CENSORED LAN MESSENGER

Project Report submitted in partial fulfillment of the requirement

for the degree of

Bachelor of Technology,

In

Computer Science & Engineering

Under the Supervision of

Dr. Nitin

By

Akshay Kumar Agarwal (111234)

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled “Censored LAN Messenger”, submitted by Akshay Kumar Agarwal 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:

Dr. Nitin

Associate Professor, CSE/IT Department

Acknowledgement

I would like to express my deep and sincere gratitude and warmly acknowledge the continuous encouragement, invaluable supervision, timely suggestions and inspired guidance offered by my guide Dr. Nitin, who was the staunch supporter and motivator of this project. Right from the inception of this project work, my supervisor has guided me till the very end in the true sense of the word.

I would also like to extend my gratitude to one and all who are directly or indirectly involved in the successful completion of this project report.

Date: **Akshay Kumar Agarwal**

Table of Content

List of Figures

List of Tables

Abstract

Teleconferencing or Chatting is a method of using technology to bring people and ideas “together” despite of the geographical barriers. The technology has been available for years but the acceptance was quite recent.

There are so many messengers already developed and performing well for the Internet. In order to use messengers of this type a user always has to connect to the internet, which may cost money and time.
If the users in organizations want to pass the messages or chat within the organization they can’t use Internet always as it may cause security problems. So they should use a messenger, which may send the messages with in the network and Internet.

my project is entitled as **Censored LAN Messenger**. It is made up of two applications the client application, which runs on the user’s system and the server application, which runs on any system on the network. To start chatting client should get connected to server where they can practice two kinds of chatting, public one (message is broadcasted to all connected users) and private one (between any 2 users only).

As a matter of fact there are several varieties of chatting. The simplest computer chatting is a method of sending, receiving, and storing typed messages with a network of users. This network could be WAN (Wide Area Network) or LAN(Local Area Network). Our chatting system is made up of two applications one runs on the server side (any computer on the network you choose it to be the server) while the other is delivered and executed on the client PC. Every time the client wants to chat he runs the client application, enter his user name and password and hits the connect button and start chatting. The system is many-to-many arrangement; every–one is able to “talk” to anyone else. Messages may be broadcasted to all receivers (recipients are automatically notified of incoming messages) or sent to special individuals (private chatting through server).

Security measures have also been installed so as to make sure that no private information like the user's password is disclosed. The password is encrypted using a cipher scheme before it is sent over the network and also before storing the username and password of a new user in the database.

CHAPTER 1: AN OVERVIEW

1.1 INTRODUCTION

It has been observed that 80% of the employees in an organization or students in a college/institute having access to the computer and internet use internet to communicate among themselves which results into very high traffic over the network.

We are becoming more and more dependent on communication over network and instant messaging greatly helps us to communicate over the network. A LAN Messenger is an instant messaging program designed for use within a single Local Area Network (LAN).

LAN messenger provides user to chat with other users. It has been observed that most of the times, user want to access his terminal from other places because it is practically impossible to carry all the data of personal computer everywhere.

This application solves the purpose of communication, files transferring and also connects remotely with the end user.

There are several advantages using a LAN messenger over a normal messenger. The LAN Messenger runs inside a company or private LAN, so only people who are inside the firewall will have access to the system. Communication data doesn't leave the LAN and the system can also not be spammed from the outside.

The purpose here is to provide a secure and reliable communication and delineate the broad technology of communication in order to provide a better understanding and to provide a basis for estimating the impact of this technology.

1.2 AIM AND OBJECTIVES

The project aims at developing an interactive application that would enable a user to communicate with other people present on a Local Area Network (LAN). The application would enable a user to send messages to anyone who is present on the LAN. A user can chat privately or can communicate in a group. An added feature that this application is offering is the screening of the messages as they are sent i.e. each and every message, be it private or group, will be scanned for any offensive words that a user might use against another user. If any such offensive words are found in the messages, then they will be filtered out and the remaining message will be sent to the recipient. Since security of a user's password is a priority in order to ensure safe usage of this application, encryption schemes will also be put in place to ensure that the passwords are always in encrypted form when they are being sent over the network or being stored in a database. The application will also have the option of file sharing.

1.3 PROJECT VISION

The vision of the project is to develop an application which comes in handy for all the people/students of an organization/college/institute. It will provide remote access to files and a safe and easily accessible and usable platform for users. It will also ensure that no one is offended by a user due to his usage of foul language, thus helping in maintaining a peaceful and calm work environment.

1.4 MOTIVATION OF THE PROJECT

The best type of motivation is the one which comes from within and from our own observations and experiences. So does the motivation for this project. Being a part of an institute where, due to a lot of restrictions, one cannot use the usually available messengers, thus having to physically travel to someone else's room in order to get the required data or having to carry our own personal laptops everywhere we go in order to have access to all our data, has led me to take on this project and develop an application which would run over a LAN and won't use the internet thus allowing the use of this application in the institute. Once this application is in place, no one will have to go through all the hardships and no one will have to carry their laptops everywhere. People can then use the application to communicate and share data with each other from their own rooms.

1.5 SCOPE OF THE PROJECT

There are several varieties of chatting. The simplest computer chatting is a method of sending, receiving and storing typed messages with a network of users. This network could be WAN (Wide Area Network) or LAN (Local Area Network). The chatting application being developed is made up of two applications: one that runs on the server side (any computer on the network can be selected to be the server) and second that is delivered and executed on the client system. Every time the client wants to chat, he runs the client application, enters his username and password and hits the connect button and starts chatting. The system is many-to-many; everyone is able to communication to everyone else. Messages maybe broadcasted to all the receivers or sent to a specific person. There is no restriction on the number of people that can participate at a given time.

1.6 INTRODUCTION TO THE MODULES

The following are the modules in the multi user chat system:

1. Server Module.
2. Client Module.
3. Public Chatting.
4. Private Chatting.
5. File Sharing.

The following are the functions of Multi user chat system:

1. **Start the Server:** The server has to be started first, only then can clients login.
2. **Client Registration:** The client first has to register his/her username and password with the server before using the application.
3. **Client Login**: The client enters his username and password that he had entered during the registration, to login the application to use it.
4. **See all online people:** Whenever a user enters, he can see all the people who are online and all the online users are also notified of the new user.
5. **Send public messages:** A user can send the message to all the people who are online.
6. **Send private messages: A** user can send the message to a particular user who is online.
7. **Send file: A** user can send a file to a particular user who is online or to all those who are online.
8. **Logout:** If the user doesn't want to communicate with anyone at the moment, then he can logout from the messenger.

CHAPTER 2: APPLICATION WALKTHROUGH

To develop the Messenger I have used java as front end. Java is a powerful, object-oriented language that supports much functionality, including client/server communication through socket programming, windows-based programming, console programming, database connectivity, image, and sound programming. Java is mainly designed for Internet programming and to create small applications that can be embedded inside an HTML page, known as **Applets**.

Before coming to the actual coding of the program, some important terms need to be defined so that the concept becomes clear. The purpose of a broadcast messenger is to create a server that is responsible for receiving and responding to messages that are received from the clients to all the clients available on the network. This is called as **Broadcasting** (to send the packet or message to all the hosts).

The **server/client architecture** is basically applied here because one of the computers will act as a **Server** responding to client messages, while all other computers will behave as **Clients** that only send requests to the server to perform some task. **Sockets** are logical connections that are created to connect the computers to each other. A port number and a host IP address/host name are required to create a socket.

Multithreading means that many threads of a process can be run on a single processor, assigning them equal time slice and priority, each feeling that it is the only process running. Multithreading is implemented here because many clients connect to the same port to the server. **Threads** are that pieces of a process or program that are assigned some resources, such as file, I/O, and so forth, and are able to run individually.

2.1 JAVA OVERVIEW

Java is loosely based on C++ syntax, and is meant to be Object oriented. It however differs from C++ in many ways. Structure of Java is widely between an interpreted and compiled language. Java programs are compiled by the Java compiler into 'Bytecode', which are secure and portable across different platforms. The technique of having intermediate code i.e., Bytecode solves both the security and the portability problems. Bytecode is highly optimized set of instructions designed to be executed by virtual machine that the Java run-time system emulates. That is, the Java run time system is an interpreter for Bytecode.

These Bytecodes are essentially instructions encapsulated in a single byte, to what is known as Java Virtual Machine (JVM), which resides in standard browser, any application can have JVM built-in .JVM, which verifies these Bytecodes into machine specific instruction at run time.

2.2 JAVA FOUNDATION CLASSES (JFC)

The FC is a suite of libraries designed to assist programmers in creating enterprise applications with Java. The Swing API is only one of five libraries that make up the JFC. The JFC also consists of the Abstract Window Toolkit (AWT), the Accessibility API, the 2D API, and enhanced support for Drag and Drop capabilities.

A brief introduction to some of the elements in the JFC:

- ***AWT:***

  The Abstract Window Toolkit is the basic GUI toolkit shipped with all versions of the Java Development Kit. While Swing does not reuse any of the older AWT components, it does build on the lightweight component facilities introduced in AWT 1.1.

- ***Drag and Drop:***

  Drag and Drop (DnD) is one of the more common metaphors used in graphical interfaces today. The user is allowed to click and "hold" a GUI object, moving it to another window or frame in the desktop with predictable results. The DnD API allows implementing droppable elements that transfer information between Java applications and native applications. Although DnD is not part of Swing, it is crucial to a commercial-quality applications

Below Figure enumerates the various components of the Java Foundation Classes. Because part of Accessibility API is shipped with the Swing distribution, we show it overlapping Swing.

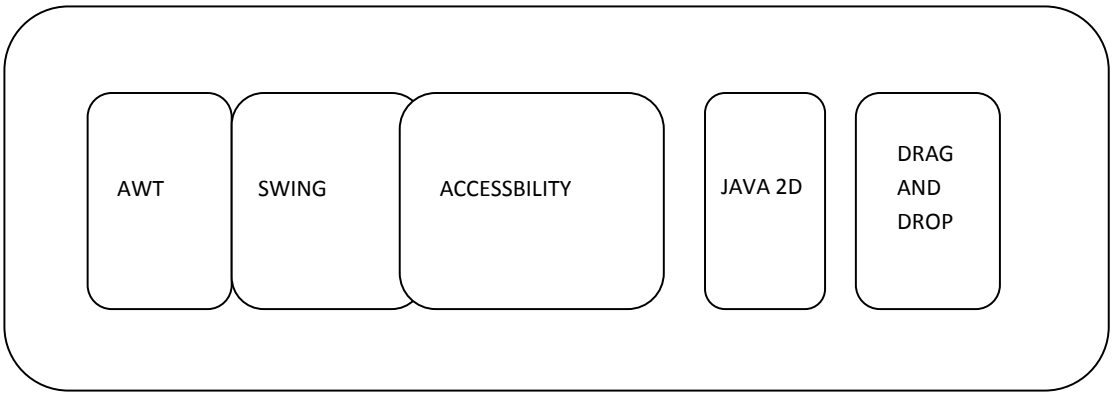

Figure 1: Java Foundation Classes

2.3 SWING

Swing can be described as a set of customizable graphical components whose look-and-feel (L&F) can be dictated at runtime. In reality, however, Swing is much more than this. Swing is the next-generation GUI toolkit that Sun Microsystems created to enable enterprise development in Java. By enterprise development, we mean that programmers can use Swing to create large-scale Java applications with a wide array of powerful components. In addition, we can easily extend or modify these components to control their appearance and behaviour.

Although Swing was developed separately from the core Java Development Kit, it does require at least JDK 1.1.5 to run. Swing builds on the event model introduced in the 1.1 series of JDK's; you cannot use the Swing libraries with the older JDK 1.0.2. In addition, you must have a 1.1-enabled browser to support Swing applets.

2.3.1 SWING REPLACES AWT?

No. Swing is actually built on top of the core AWT libraries. Because Swing does not contain any platform-specific (native) code, you can deploy the Swing distribution on any platform that implements the Java 1.1.5 or above virtual machine. In fact, if you have JDK 1.2 or higher on your platform, then Swing classes are already available, and there's nothing further to download.

Below figure shows the relationship between Swing, AWT, and the Java Development Kit in the 1.1 and higher JDK's. In JDK 1.1, the Swing classes must be downloaded separately an included as an archive file on the classpath (*swingall.jar*). JDK 1.2 (and higher) comes with a Swing distribution.

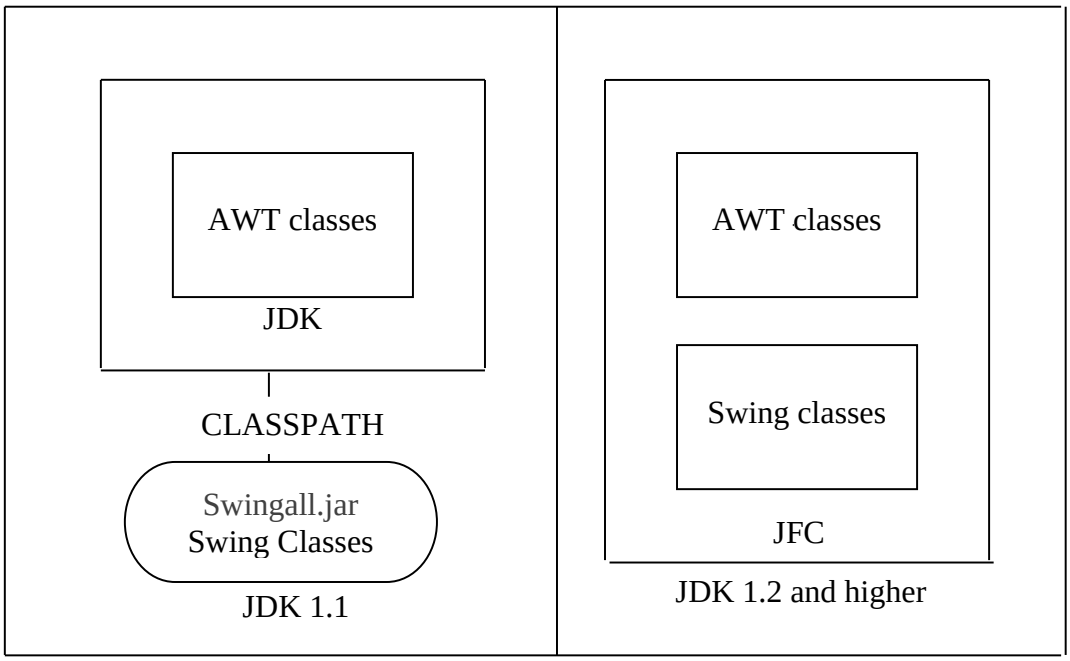

Figure 2: Relationship between swing, AWT and JDK

Swing contains many more graphical components that it's immediate predecessor, AWT 1.1. Many are components that were scribbled on programmer wish lists since Java first debuted including tables, trees, internal frames, and a plethora of advanced text components. In addition, Swing contains many design advances over AWT. For example, Swing introduced an Action class that makes it easier to coordinate GUI components with their functionality.

You'll also find that a much cleaner design prevails throughout Swing: this cuts down on the number of unexpected surprises that you're likely to face while coding.

Swing depends extensively on the event-handling mechanism of AWT 1.1, although it does not define a comparatively large amount of events for itself. Each Swing component also contains a variable number of exportable properties. This combination of properties and events in the design was no accidents. Each of the Swing components, like the AWT 1.1 components before them, adheres to the popular JavaBeans specifications. This means that you can import all of the Swing components into various GUI builder tools, which is useful for powerful visual programming.

2.3.2 SWING FEATURES

Swing provides many features for writing large-scale applications in Java. Here is an overview of some of the more popular features.

- **Pluggable Look-and-Feels**

One of the most exciting aspects of the Swing classes is the ability to dictate the L&F of each of the components; even resetting the L&Fs have become an important issues in GUI development over the past 10 years. Many users are familiar with the Motif style of user interface, which was common in Windows 3.1 and is still in wide use on Unix platforms. Microsoft created a more optimized L&F in their Windows 95/98/NT/2000 operating systems. In addition, the Macintosh computer system has its own carefully designed L&F, which most Applet users feel comfortable with.

Swing is capable of emulating several L&Fs and currently supports the Windows, Unix Motif, and "native" Java Metal L&Fs, Mac OS X comes with full support for its own L&F based on Apple's Aqua Human Interface Guidelines, although you can still access Metal if you prefer. In addition, Swing allows the user to switch L&Fs at runtime without having to close the application. This way, the user can experiment to see which L&F is best for her with instantaneous feedback. Users can create they own L&F for each one of the Swing components.

The Metal L&F combines some of the best Graphical elements in today's L&Fs and even adds a few surprises of its own. All Swing L&Fs are built from a set of base classes called the Basic L&F.

- ***Lightweight Components***

Most Swing components are lightweight. In the purest sense, this means that components are not dependent on native peers to render themselves. Instead, they use simplified graphics primitives to paint themselves on the screen and can even allow portions to be transparent.

The ability to create lightweight components first emerged in JDK 1.1, although the majority of AWT components did not take advantage of it. Prior to that, Java programmers had no choice but to subclass java.awt.Canvas or java.awt.Panel if they wished to create their own components. With both classes, Java allocated an opaque peer object from underlying operating systems to represent the component, forcing each component to behave as if were its own window, thereby taking on a rectangular, solid shape. Hence, these components earned the name "heavyweight" because they frequently held extra baggage at the native level that Java did not use.

Heavyweight components were unwieldy for two reasons:

- Equivalent components on different platforms don't necessarily act alike. A list component on one platform, for example, may work differently than a list component on another. Trying to coordinate and manage the differences between components was a formidable task.
- The L&F of each component was tied to the host operating system and could not be changed.

Programming your own L&F can get pretty complex; in fact, the source code for an entire L&F would far exceed the size of ordinary program.

2.4 NETWORK PROGRAMMING

2.4.1 NETWORK APLICATION

Network application exchange data between physically separated machines. For this to occur, the machines must be connected by a transmission media. There are many different types of communication links and new ones continue to be developed. Coaxial cables, phone lines, digital phone lines, fibre optic cable, satellite beam, and infrared waves are all used as transmission media for exchange data between computers.

A network includes a group of computers connected by a physical link allowing data to be exchanged between them. A local are network on LAN is a network of computers in close physical proximity, usually a single building, but can be a group of adjacent buildings. Over the last decades LANs have become an important component of the computer workplace.

2.4.2 PROTOCOL STACKS

Very Early in the history of computer network development the concept of separating the problem into multiple levels was adapted. With a multilevel architecture each layer can handle a different aspect of networking and provide that functionality to the above layer. TCP/IP is a specific implementation of a multi level network architecture. In both, the first and second chapter, we are always repeating the same sentence, which is TCP/IP protocol. It is now the time to dissect this sentence.

2.4.3 TCP

TCP (the "Transmission Control Protocol ") has the responsibility for breaking up the message into datagrams, reassembling them at the other end, resending anything that gets lost, and putting things back in the right order. It may seem that TCP is doing all the work. And in small network it is true. With TCP, there is no maximum message length. When a message is passed to the TCP protocol, if it is too large to be sent in one peace, the message is broken up into chunks or packets and sent one at a time to the destination address. The TCP packet contains the addressing information. The TCP message also contains a packet number and total number of packets. Because of the nature of the TCP/IP protocol, the packet may travel different paths and may arrive in a different order than sent. TCP reassemble the packets in the proper order and requests the retransmission of any missing or corrupted packets. TCP enables you to create and maintain a connection to a remote computer. By using the connection, both computers can stream data between each other.

2.4.4 IP

As the number of computers networked become larger, a system becomes necessary to give remote computers the capability to recognize other remote computers; thus the IP addressing method was born. Therefore, simply an IP address uniquely identifies any computer connected to a network. This address is made up of 32 bits divided into 4 four bytes. But since the number of connected computers is too large and since it is difficult to remember all their IP addresses, the Domain Name Service (DNS) was designed. It has the job of transforming the unique computer names (host name) into an IP address. There for, whenever in our project we run the client application and enter the host name, this means that we are writing the IP address of the remote computer we want to connect to indirectly. In general, TCP/IP is a set of protocols developed to allow cooperating computers to share resources across the network.

2.4.5 SERVICE PORT

Till now, we have seen that TCP/IP forms the backbone for communication between computers, but do you know how these computers speak to each other? The answer is **Ports**. A port is a special location in the computer's memory that exists when two computers are communicating via TCP/IP. Application uses a port number to communicate and the sending and receiving computers use this same port to exchange data.

2.4.6 SOCKETS

The world is defining itself as a largely Intel-processor, windows-based set desktops communicating with back end servers of various types. Hardware and software technology advances are pushing PC's into the role of every where communications devices. For software applications to take advantage of increasingly sophisticated and feature-rich communications technology, they require an Application Programming Interface (API) which provides a simple and uniform access to this technology. WinSock has been this interface for TCP/IP on windows systems for the last 3 years. It is now set to become the definitive applications interface for all windows-based communication-capable applications.

CHAPTER 3: APPLICATION DESIGNING

3.1 UML DIAGRAMS

3.1.1 USE CASE MODELING

It helps us to describe the functional requirements of the application developed in the overall system.

Use Case Description:

Use Case: SERVER

Actor: Administrator

If the administrator selects the start option then the server should start running until

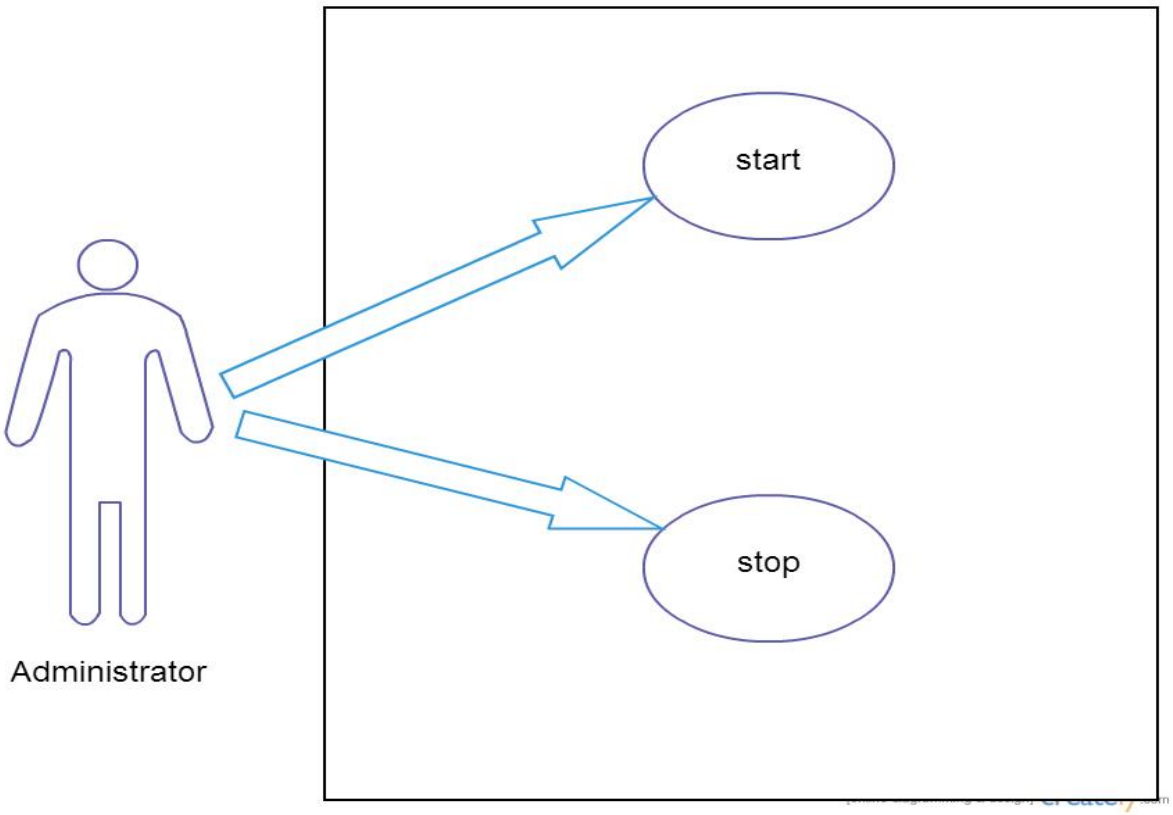

Figure 3: Use case Diagram of server module

Use case description:

Use Case : CHOICE

Actor: Client

The client is given the option to either " LOGIN " or " REGISTER " when he runs the client application.

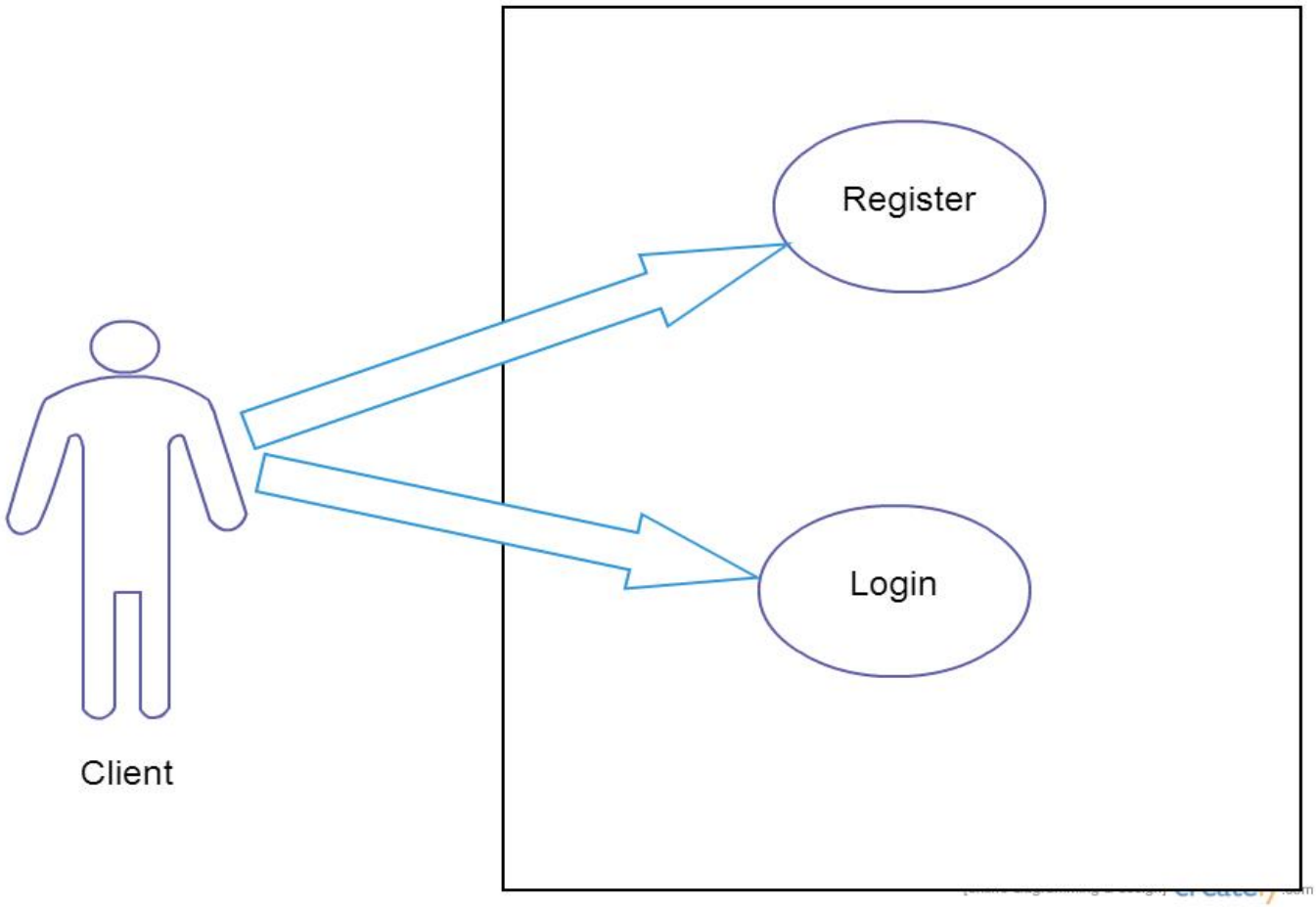

Figure 4: Use Case Diagram of Choice Module

Use case description:

Use Case : LOGIN

Actor: Client

The following diagram shows that a client can send private messages or send messages to all the other clients. A client can also send a file. The client has to login to the applicaition using a username and password which he had used for registering himself.

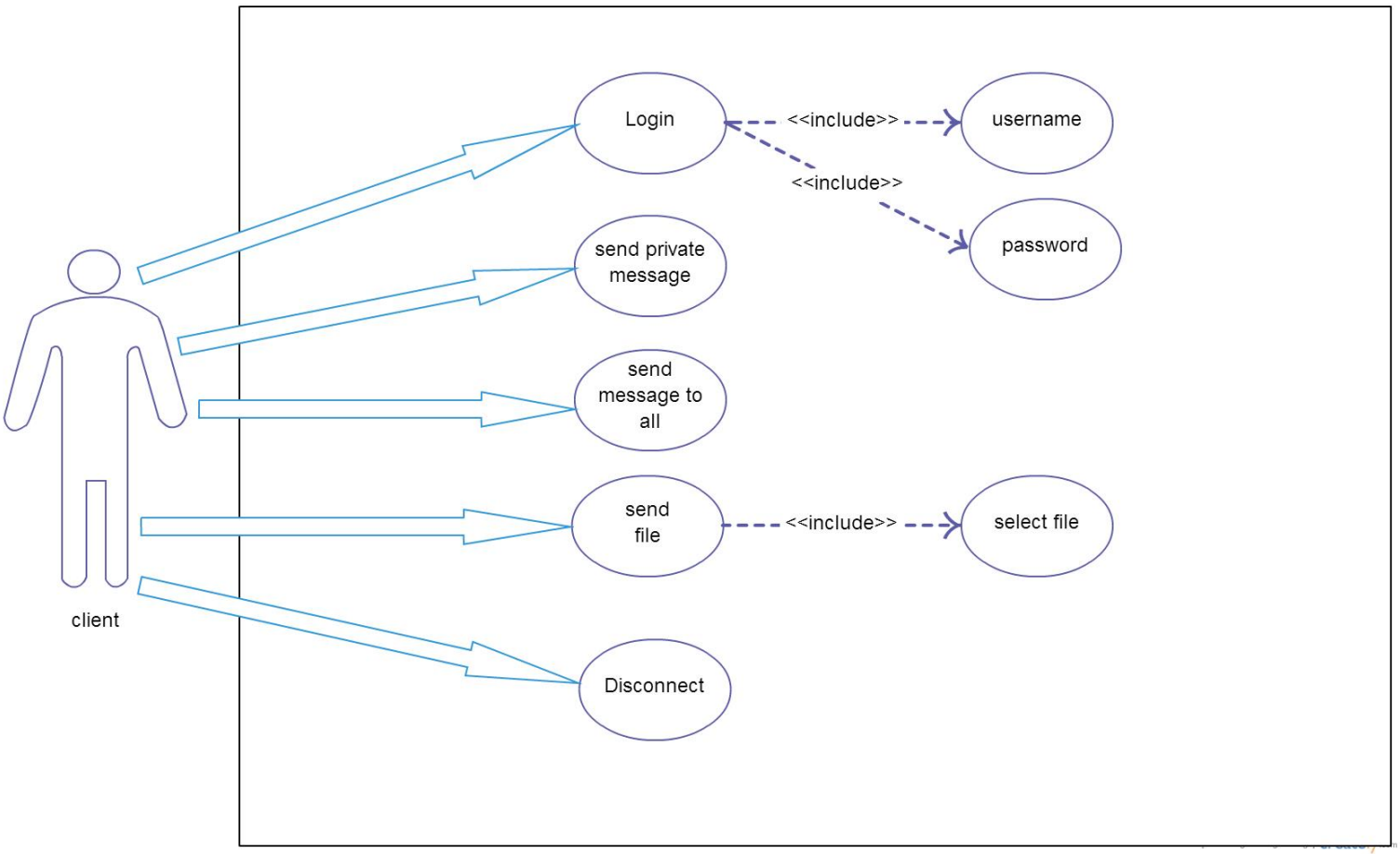

Figure 5: Use Case Diagram of Login module

Use case description:

Use Case : REGISTER

Actor: Client

A client is asked to enter a username and a password when he decides to register. Once done, the client clicks on the register button and if all checks out to be correct, a success message is displayed, else an error message is displayed.

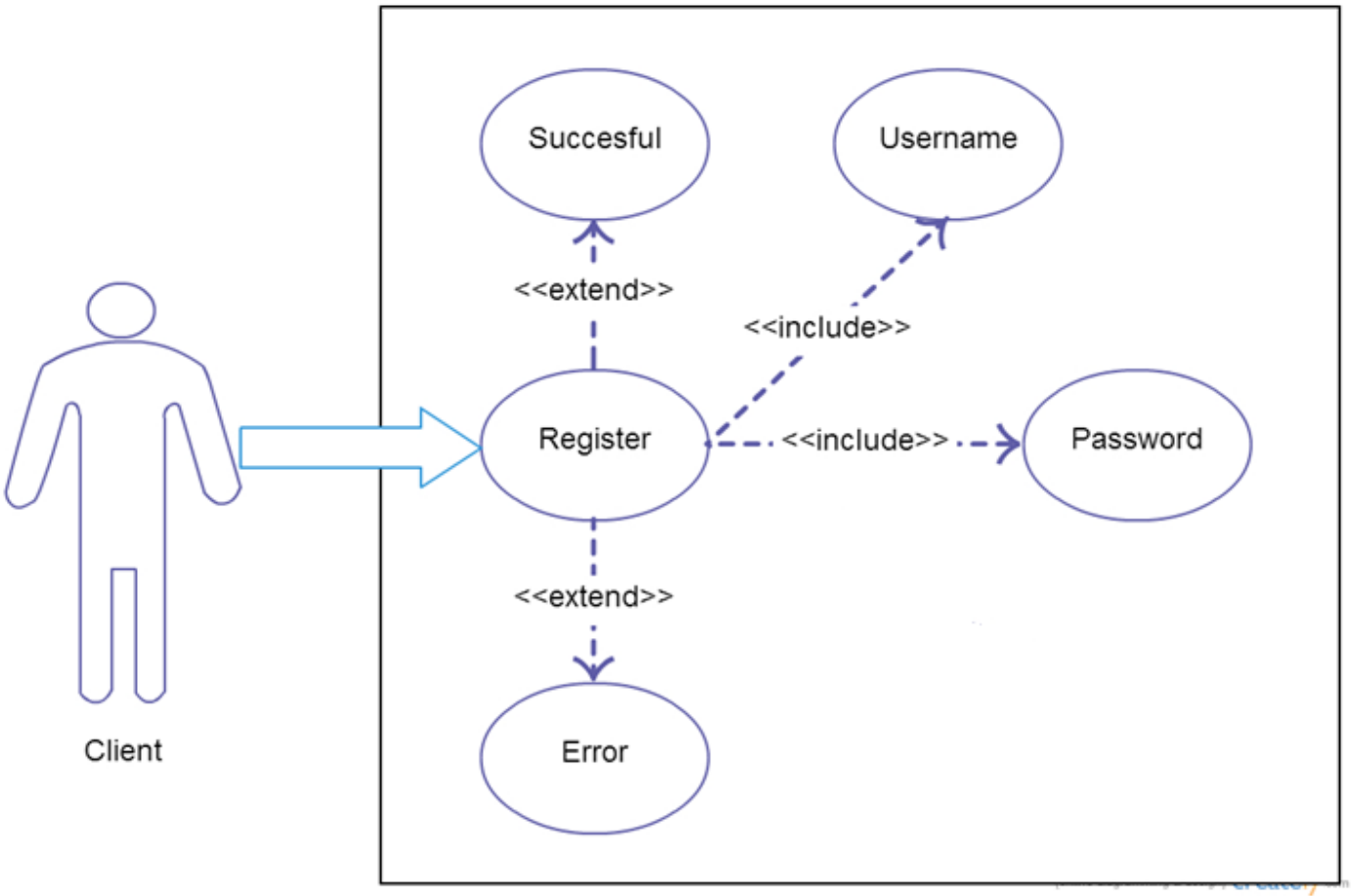

Figure 6: Use case diagram of Register module

3.1.2 SEQUENCE DIAGRAM

A sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence.

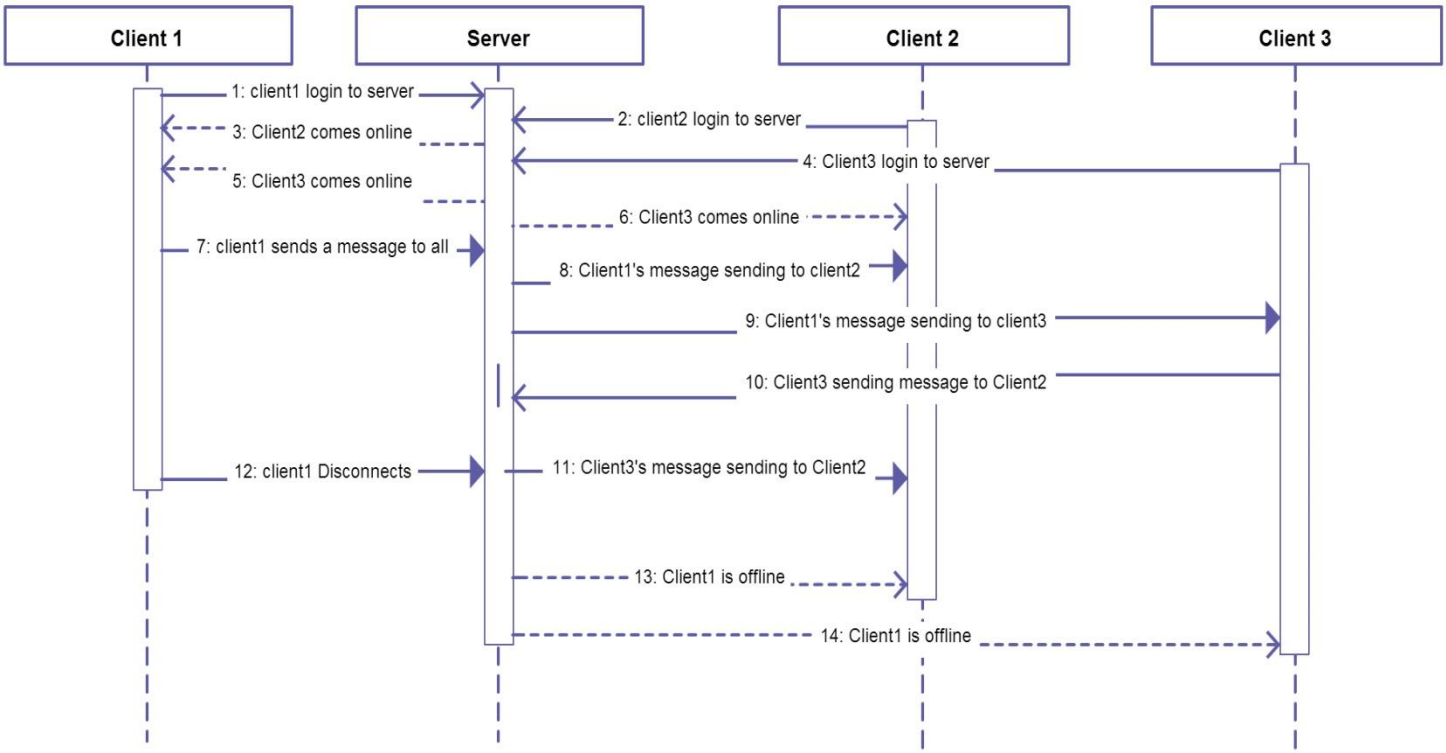

Figure 7: sequence diagram of LAN Messenger

3.1.3 DATA FLOW DIAGRAM

These diagrams depict the flow of data from one point of the system to another point.

Level '0' DFD or Context Diagram:

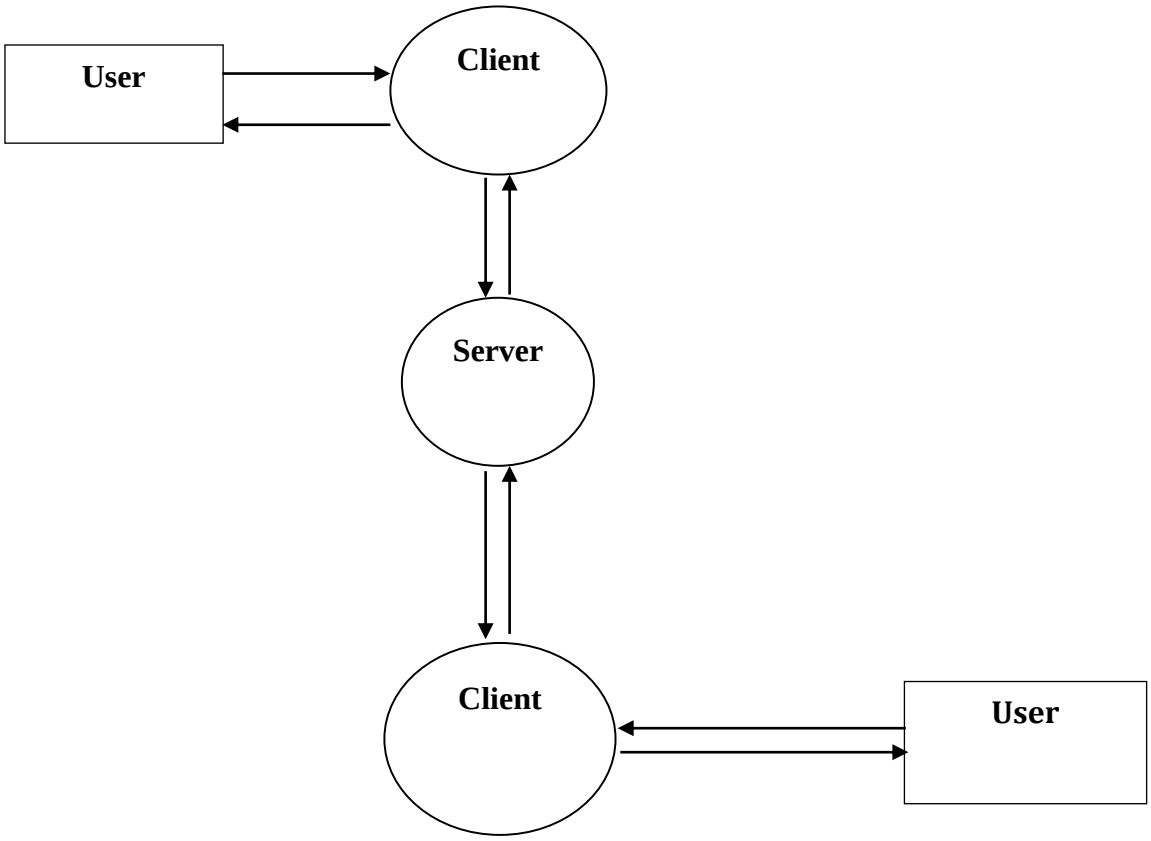

Figure 8: Context diagram of Client

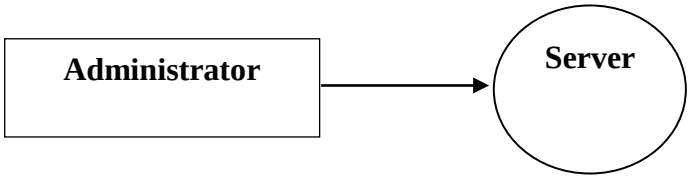

Figure 9: Context Diagram of administrator

Level '1' DFD:

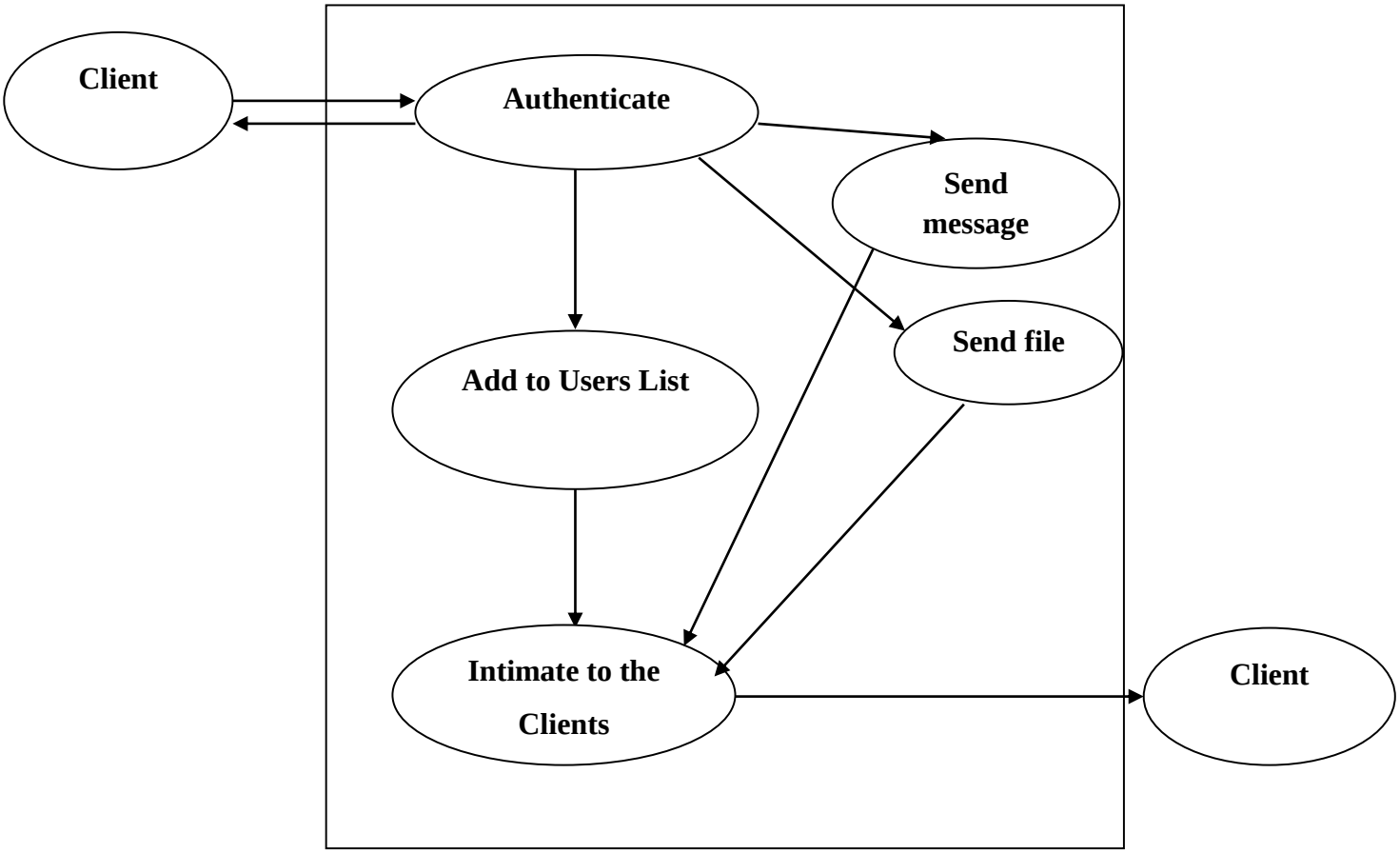

Figure 10: level '1' DFD of Client

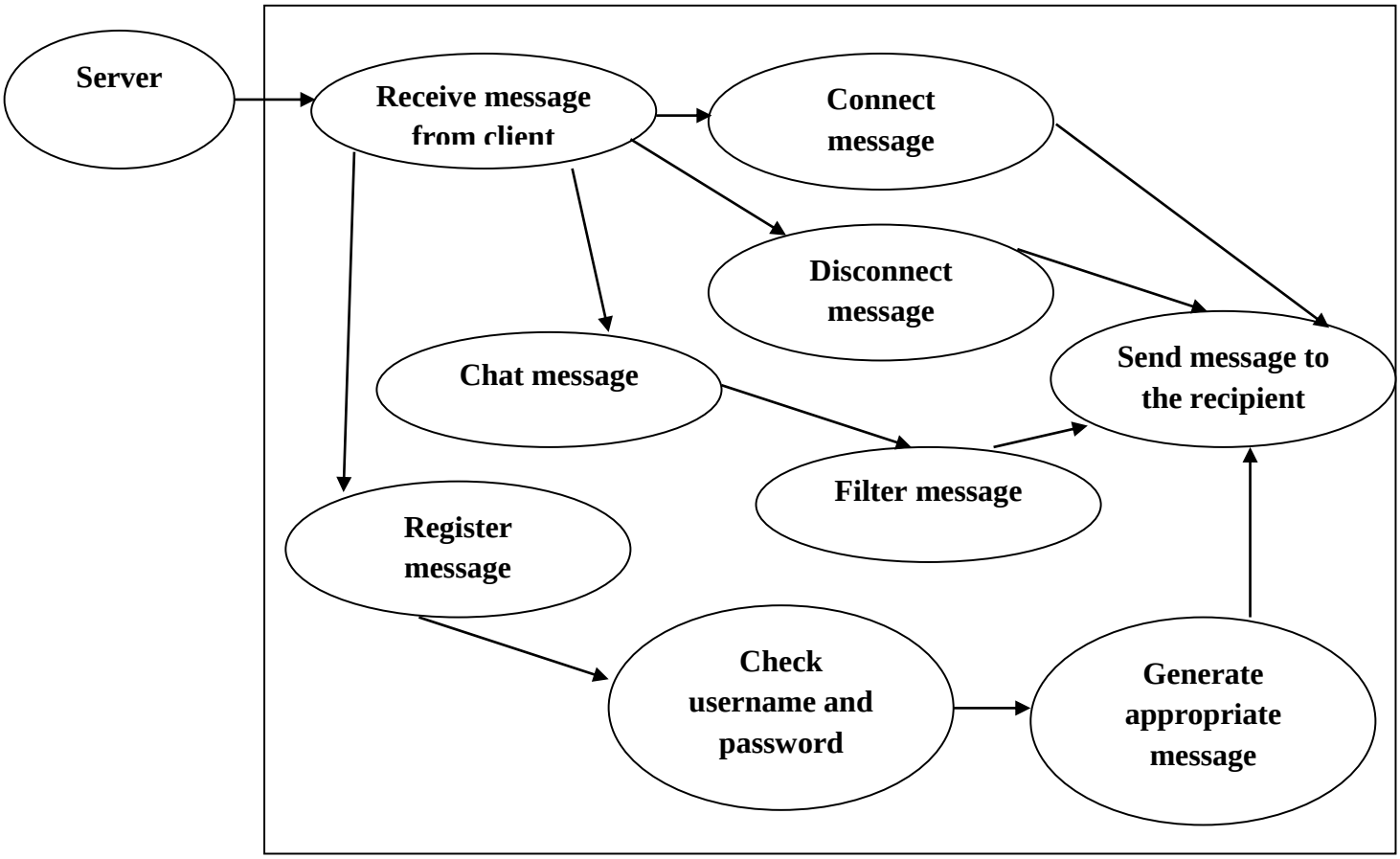

Figure 11: level '1' DFD of Server

3.2 FEATURES OF THE APPLICATION

Censored LAN Messenger is an application designed for the sole purpose of using it for communication within an organization/institute. It is developed so that users can have a safe and clean communication with each other over a single LAN. Some of the features of the Censored LAN Messenger application are:

i. ***Safe Communication:*** The application uses an encryption scheme in order to encrypt the password that a user registers and uses.
ii. ***File Transfer:*** The application allows a client to transfer files from his system to another currently online client's system.
iii. ***Message Screening:*** The application screens each and every text message that a client sends and ensures that no offensive words are used. If any offensive words are found, then they are filtered out and the filtered message is then sent to the recipient. The application currently supports screening of more than 200 offensive words.
iv. ***Private Messages:*** The application allows a client to send private messages to another client. These messages are only received by the person who the client has selected.
v. ***Group Messages:*** The application allows a client to send a group message which is broadcasted to all the currently online users.
vi. ***Emoticons:*** The application allows a client to send multiple emoticons. The application supports over 20 everyday emoticons**.**

3.3 APPLICATION USER INTERFACE

The first window that a user gets when he executes the client module is shown in **figure 12.** The user has the option of either registering as a new user or login using the credentials that he had used while signing up. **Figure 14** shows the window displaying successful registration message and logging in the user that has just registered with the new (unique) username and password. **Figure 15** shows the window displaying error when a user enters invalid inputs while registering. **Figure 17** shows the window displaying error message when user enters a wrong login credential. A client needs to connect to the server before he can do anything else. The application provides the user the option to enter the address of the host and also the port. **Figure 18** shows the window displaying an error message when a client is unable to connect to the server. **Figure 19** shows the application displaying a successful login message on client login. **Figure 20** shows the addition of new users to the online users list as and when they login. **Figure 21** shows users communicating to each other privately. Private messages can be done by simply clicking on the name of the user with whom one wants to communicate privately and then simply sending the message. **Figure 22** shows users communicating in a group by sending a group message. A group message can be sent by clicking on the **all** option in the online users list. **Figure 23** shows the file selection window that is displayed when a user clicks on the file selection button. **Figure 24** shows the application notifying each user when another user has logged out of the application. **Figure 25** shows the server window. **Figure 26 & 27** shows the various messages that a server windows displays when a connection is received. **Figure 28 & 29** files the file acceptance and file save location dialogue boxes that are displayed when a client receives a file and accepts the file transfer. **Figure 30** shows that the sender is notified by a upload complete message when the file is completely uploaded and the recipient is notified by a download complete message once the download is completed. **Figure 31, 32 & 33** shows a user sending an offensive message and the application censoring the offensive part of the message and displaying the rest of the message to the recipient/s. The application currently supports over 200 offensive words. **Figure 34 & 35** show the use of emoticons by the user. The application supports over 20 general everyday emoticons. These emoticons can be displayed by simply typing the keyboard shortcuts of them. Eg. The normal smiley face can be displayed using **:) or :-).** The application converts these symbols on its own to the respective emoticon.

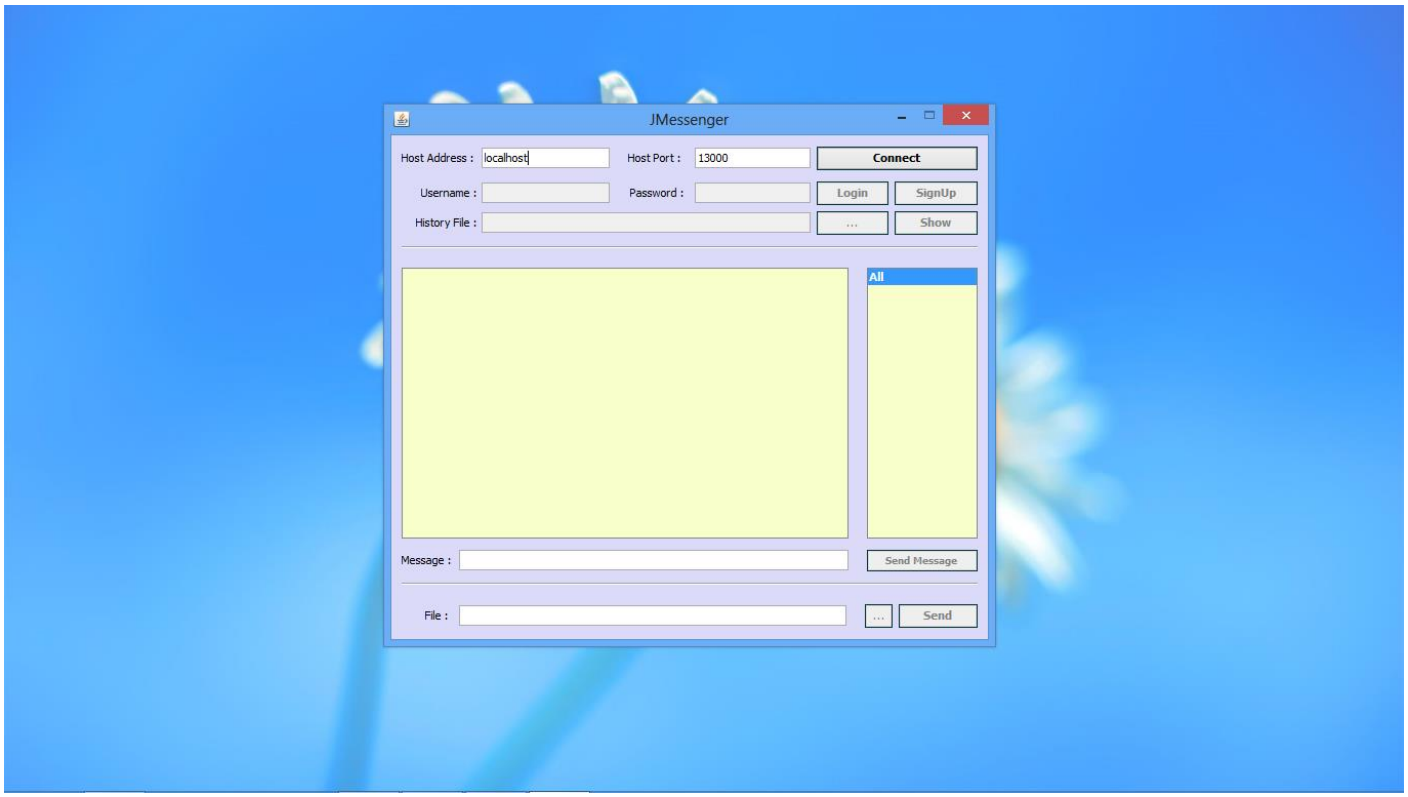

Figure 12: Initial Window

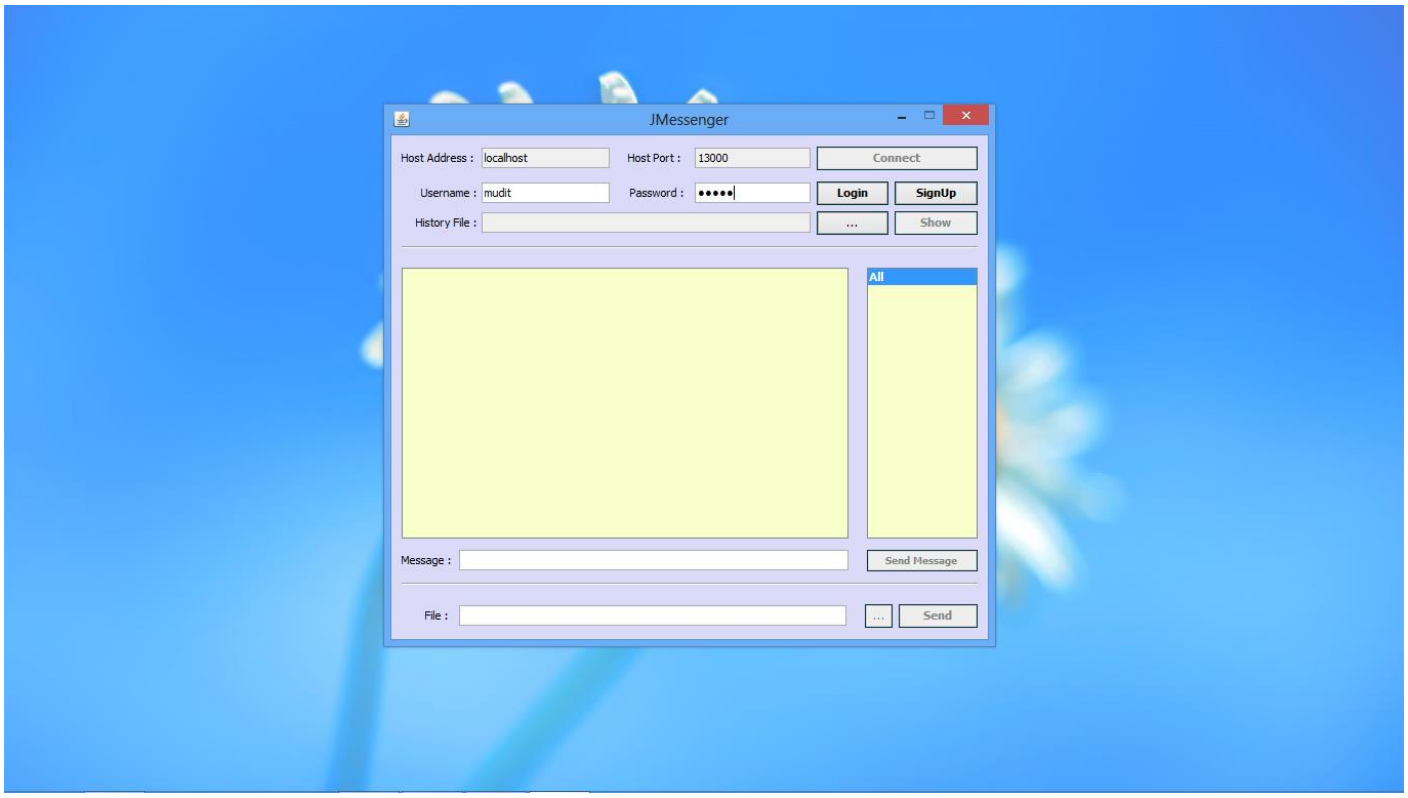

Figure 13: Regsitration Window

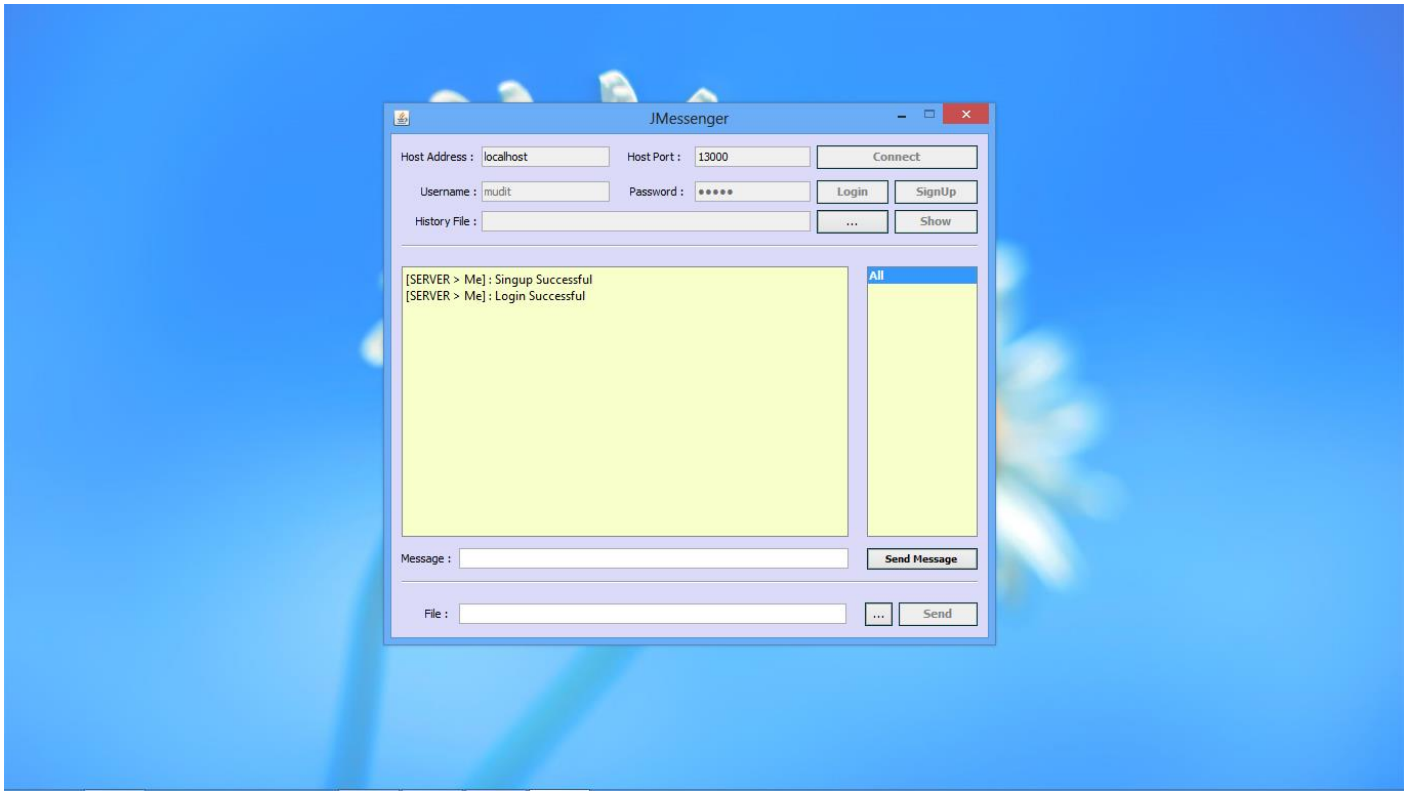

Figure 14: Successful Registration message

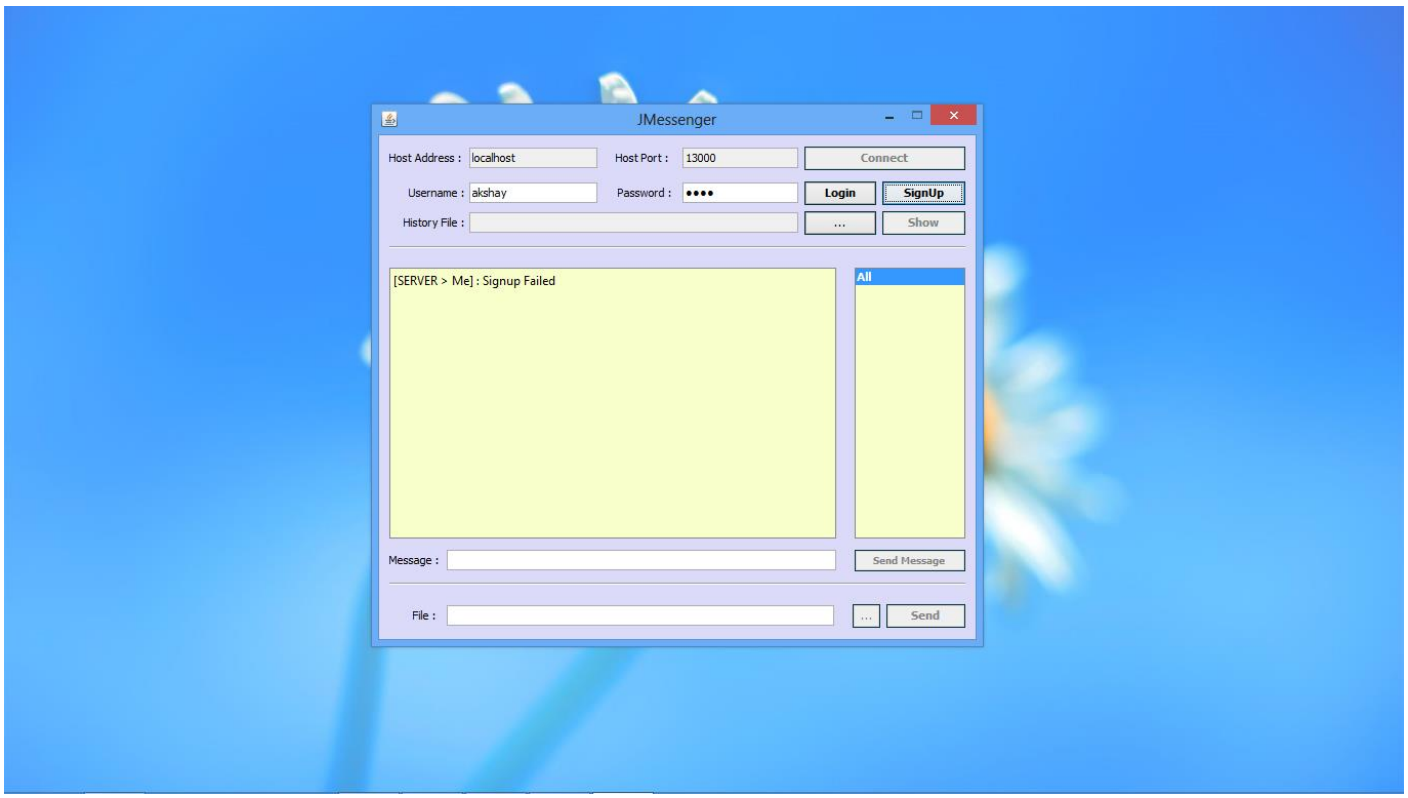

Figure 15: Error Message while registering

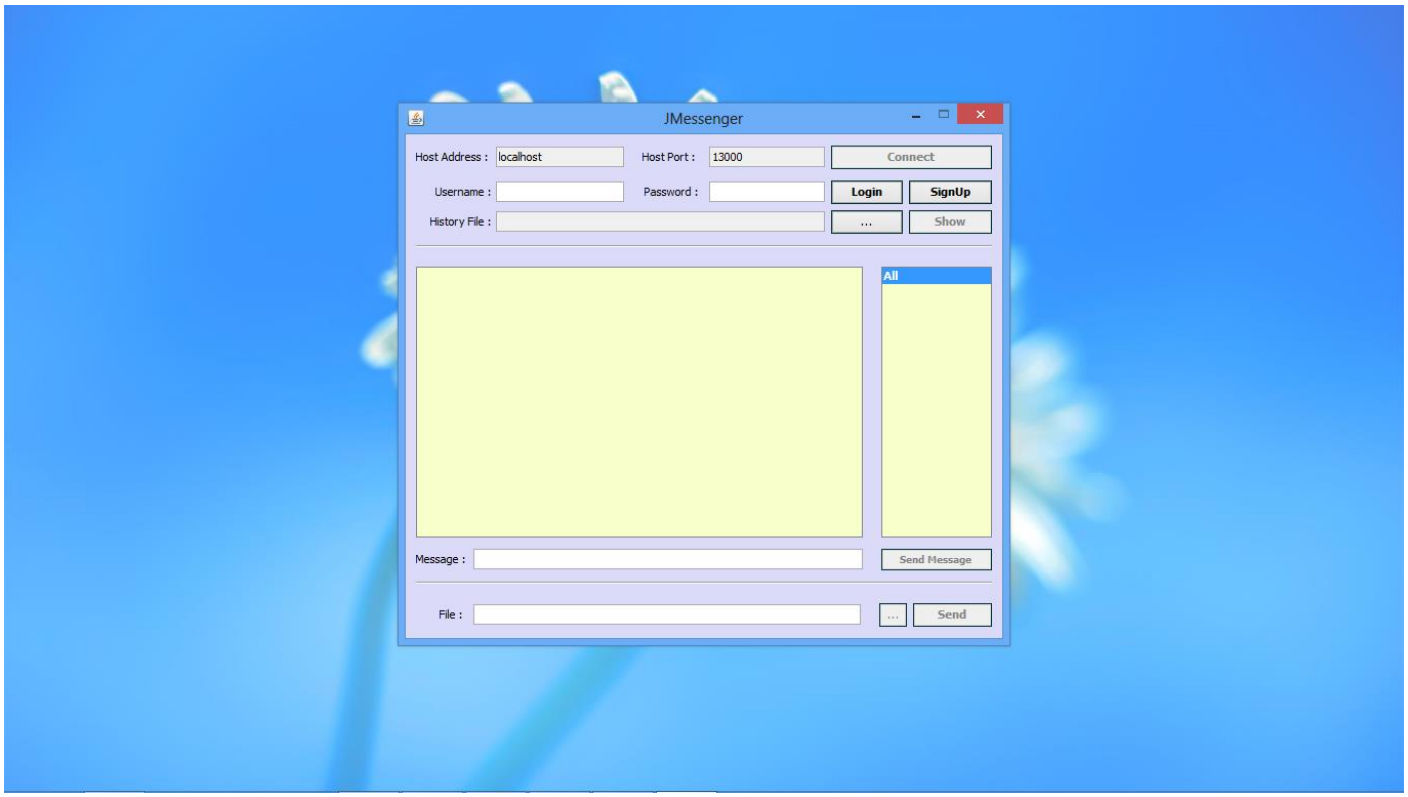

Figure 16: Login Window

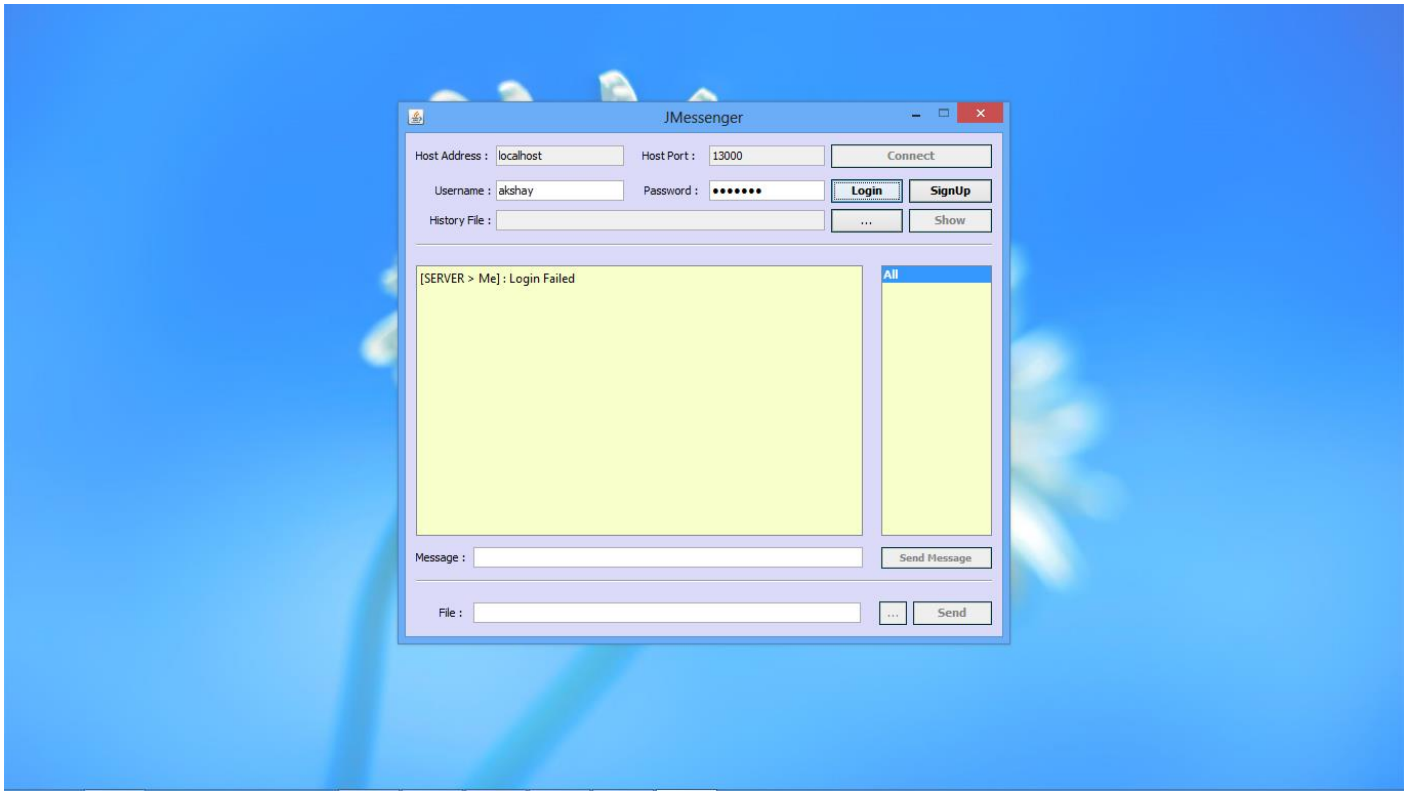

Figure 17: Error Message while login

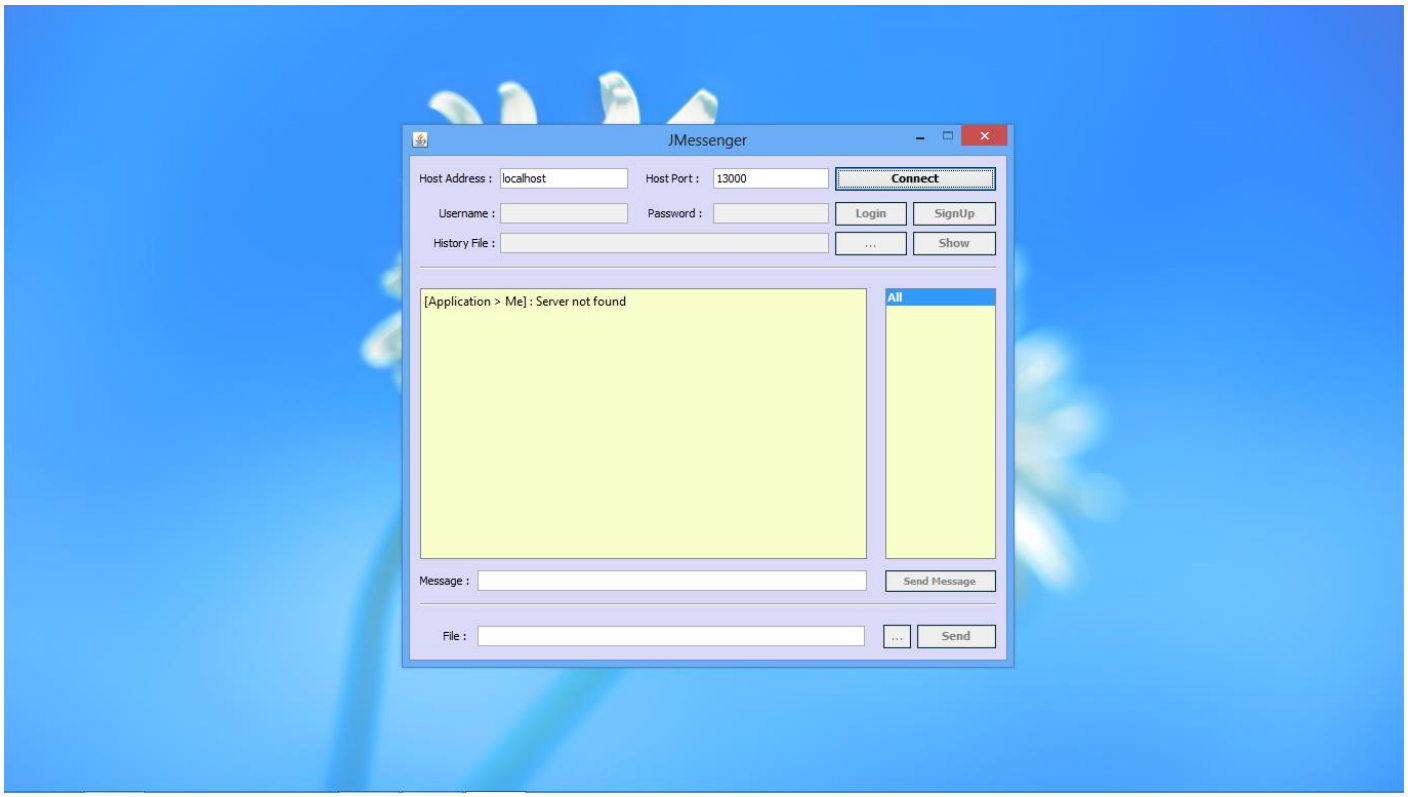

Figure 18: Error Message while connecting to server

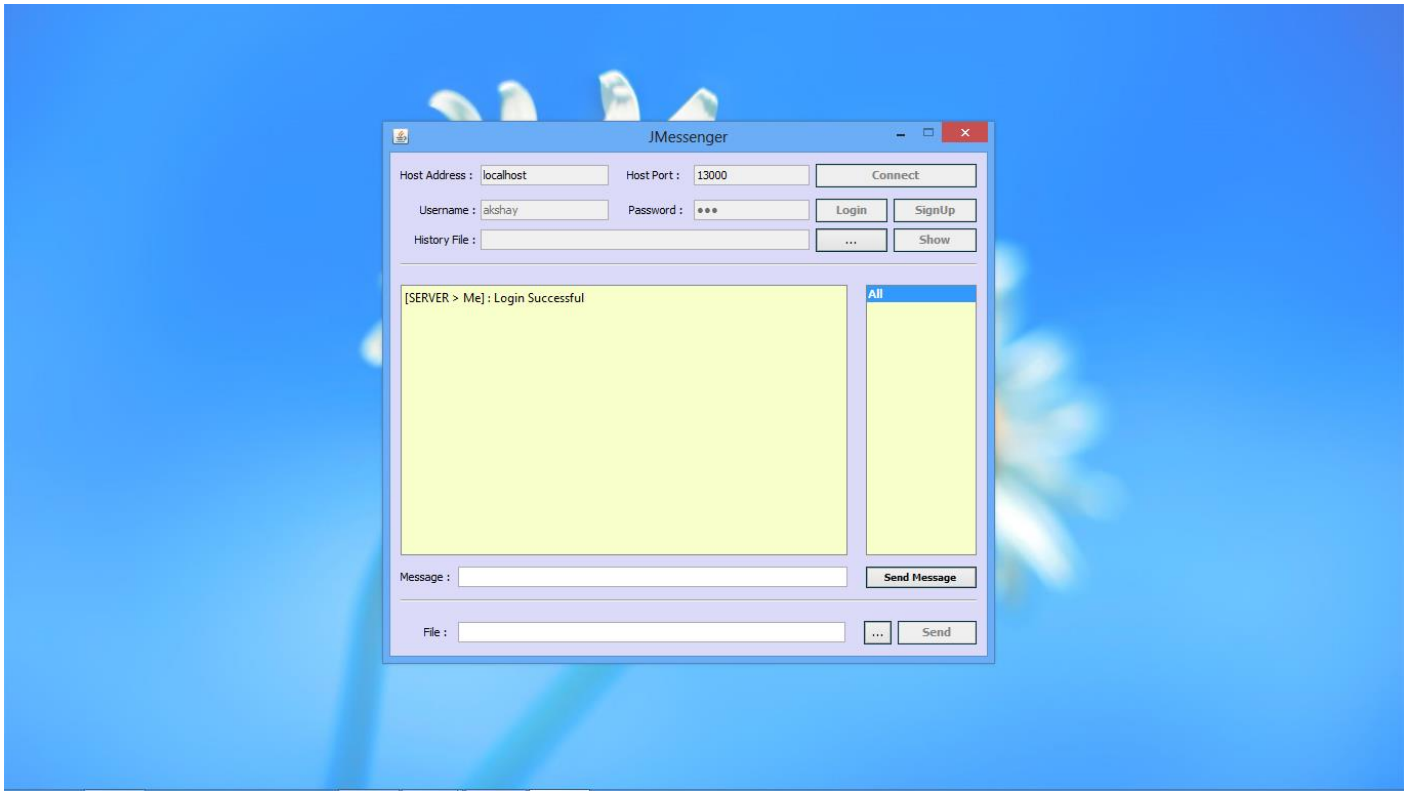

Figure 19: Successful Login

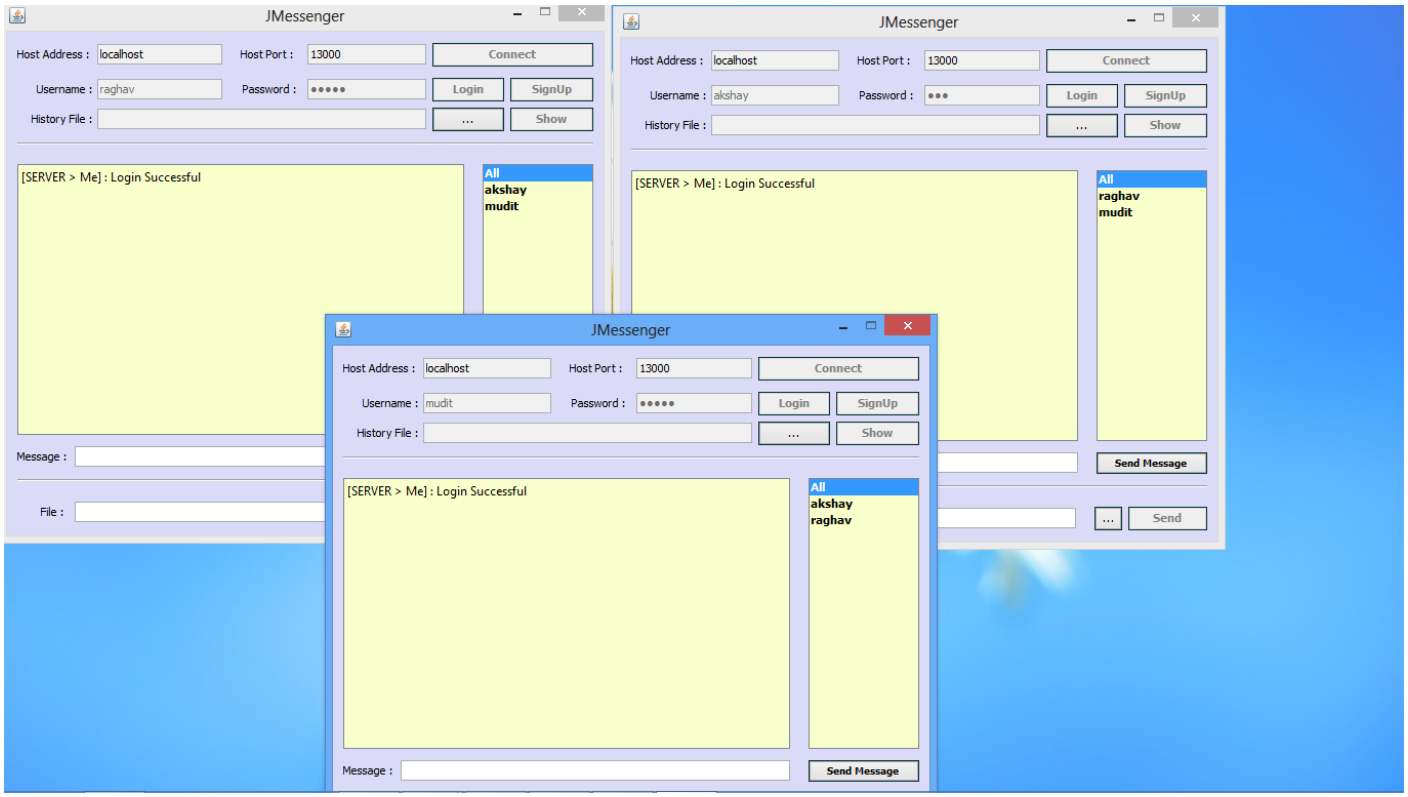

Figure 20: Login Notification

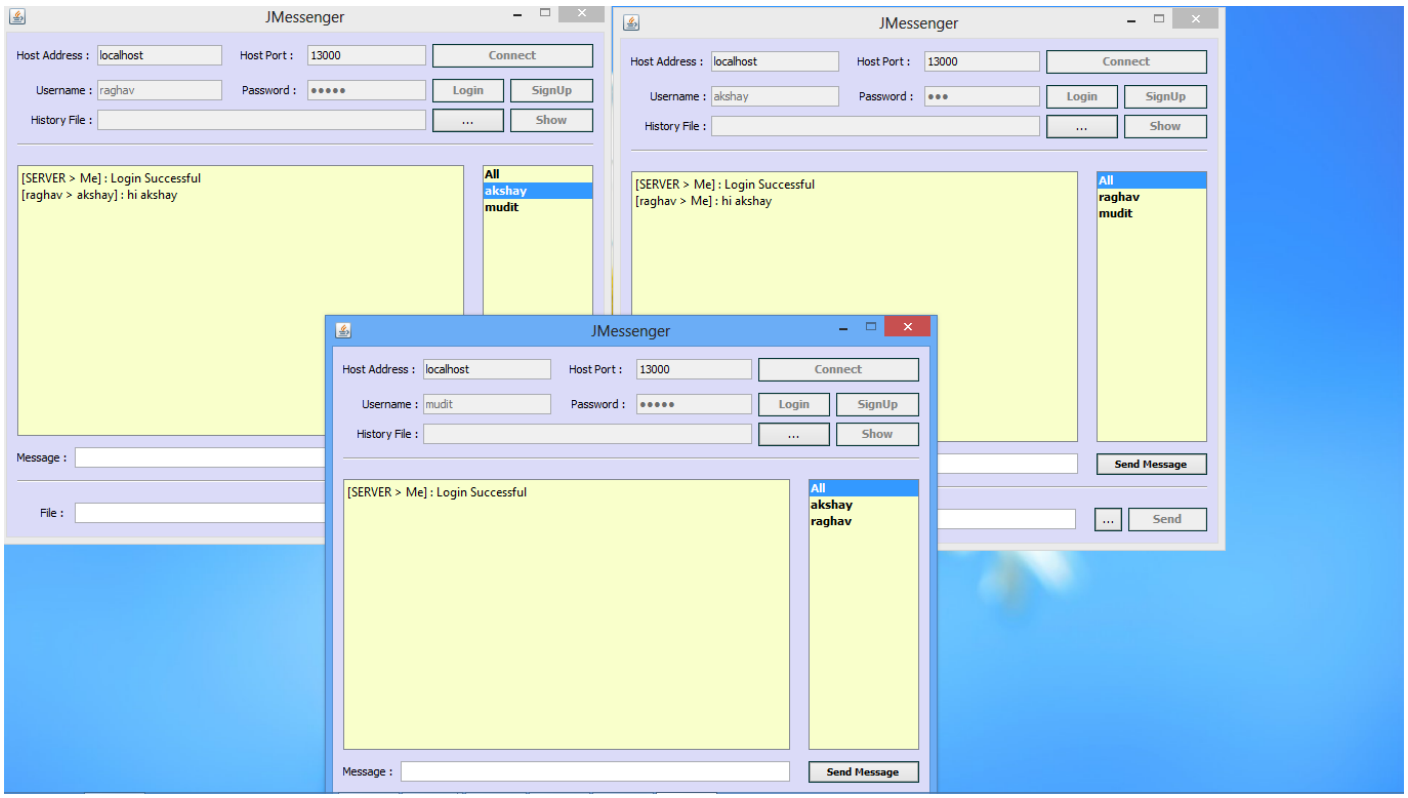

Figure 21: Private Message

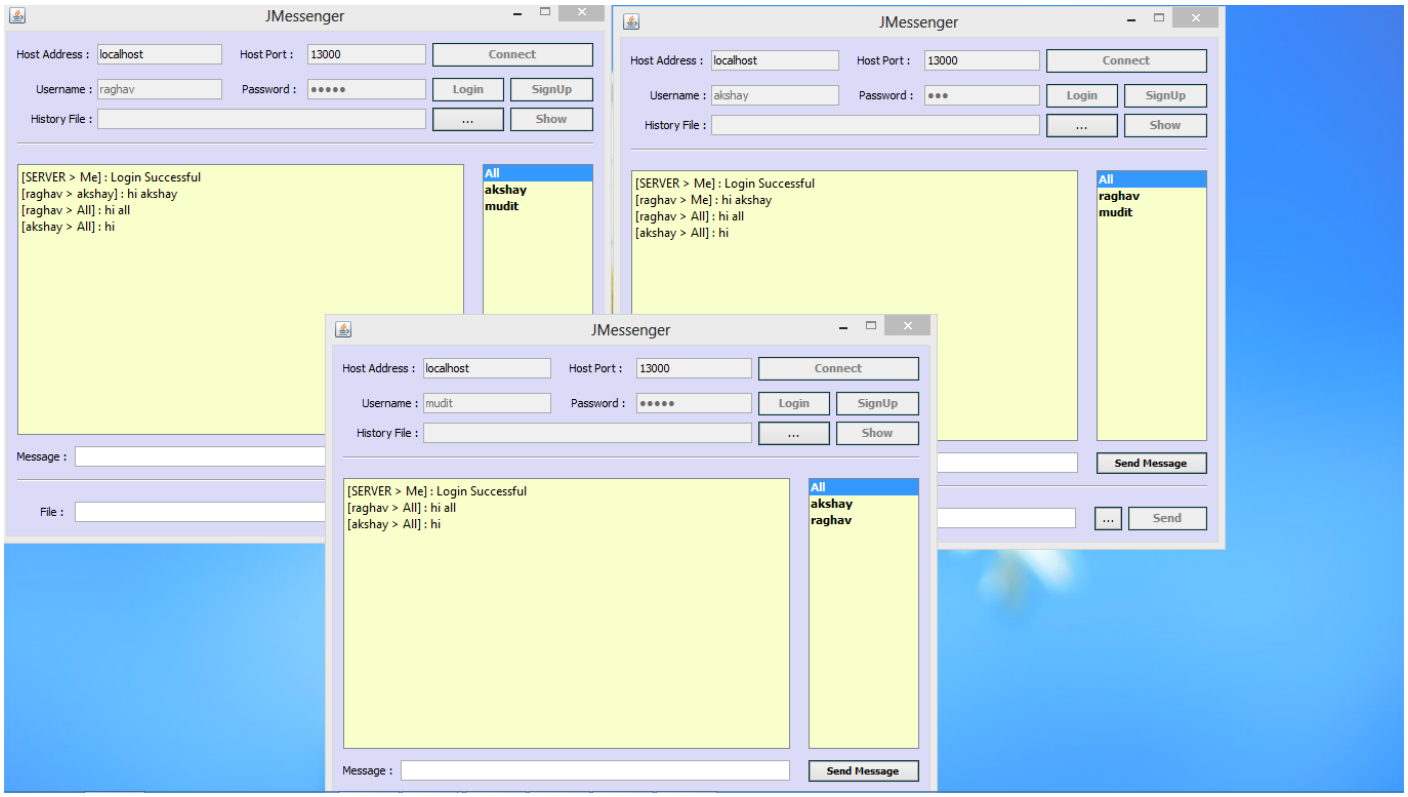

Figure 22: Group Message

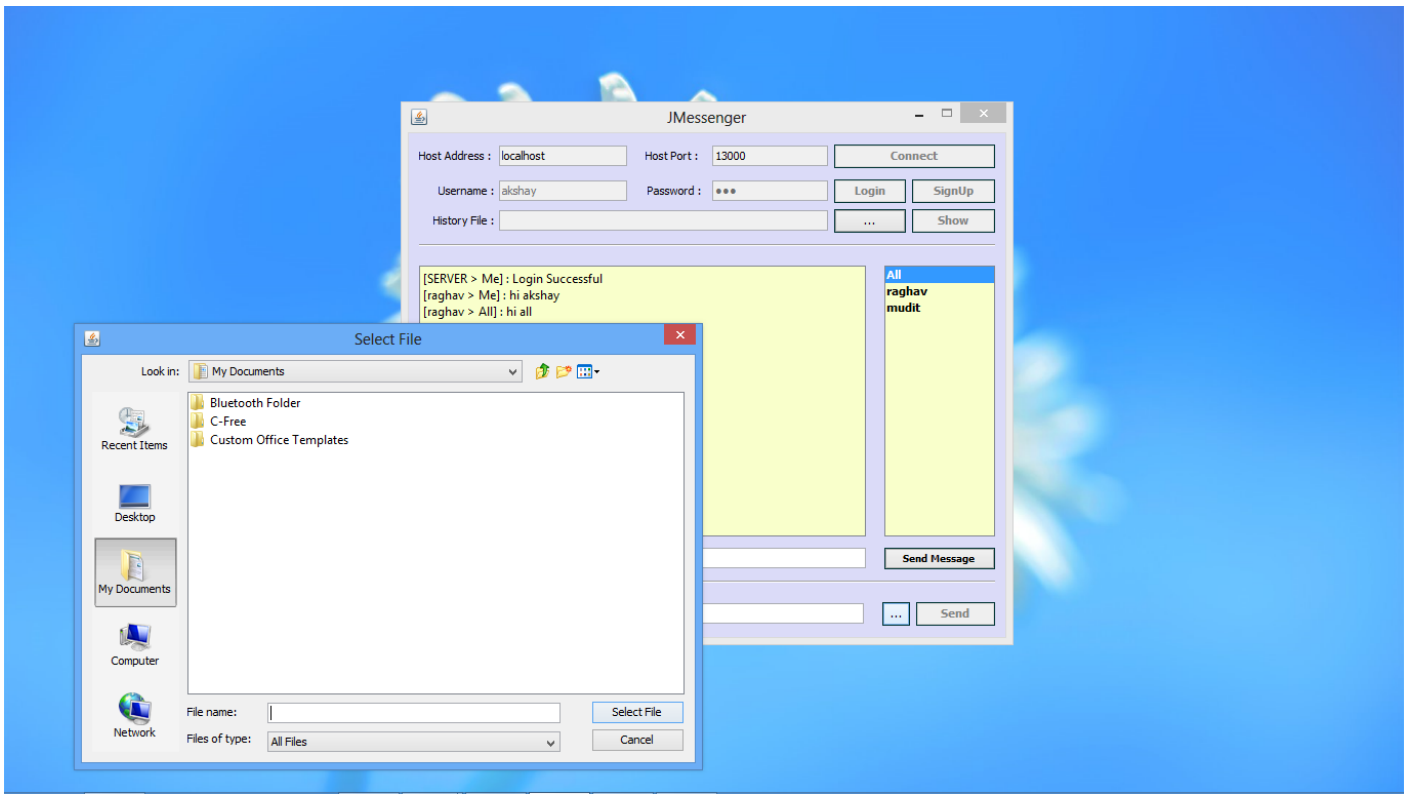

Figure 23: File Selection Window

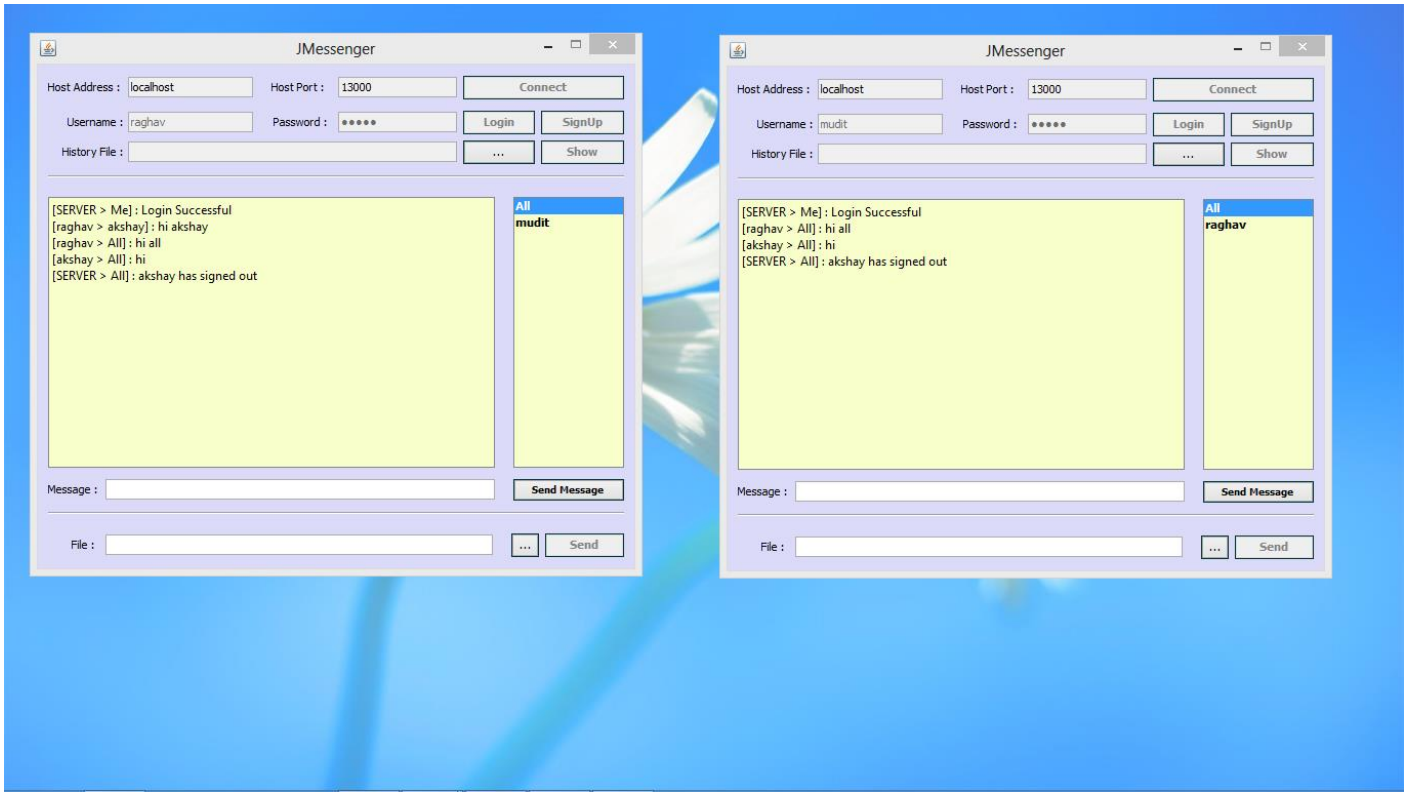

Figure 24: Disconnect Notification

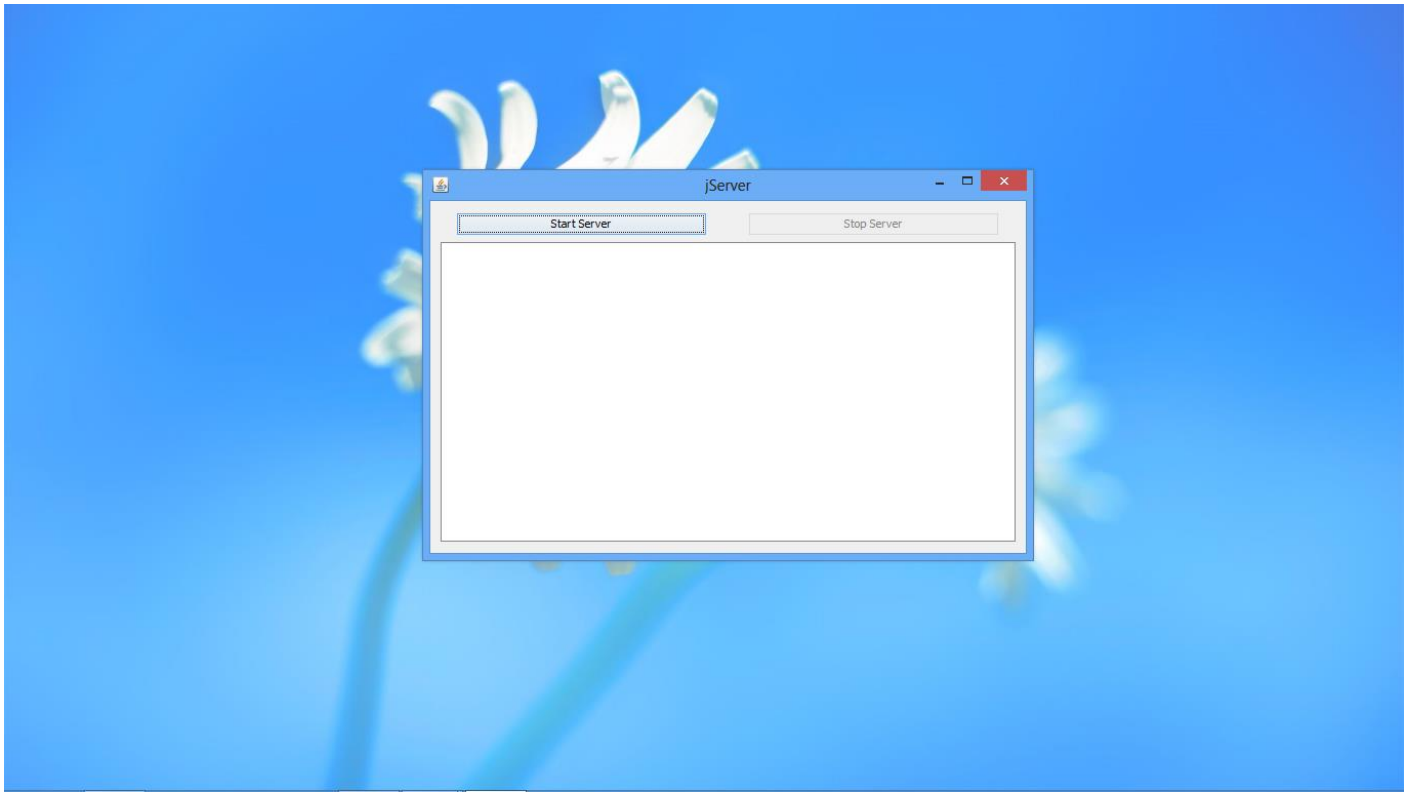

Figure 25: Server Window

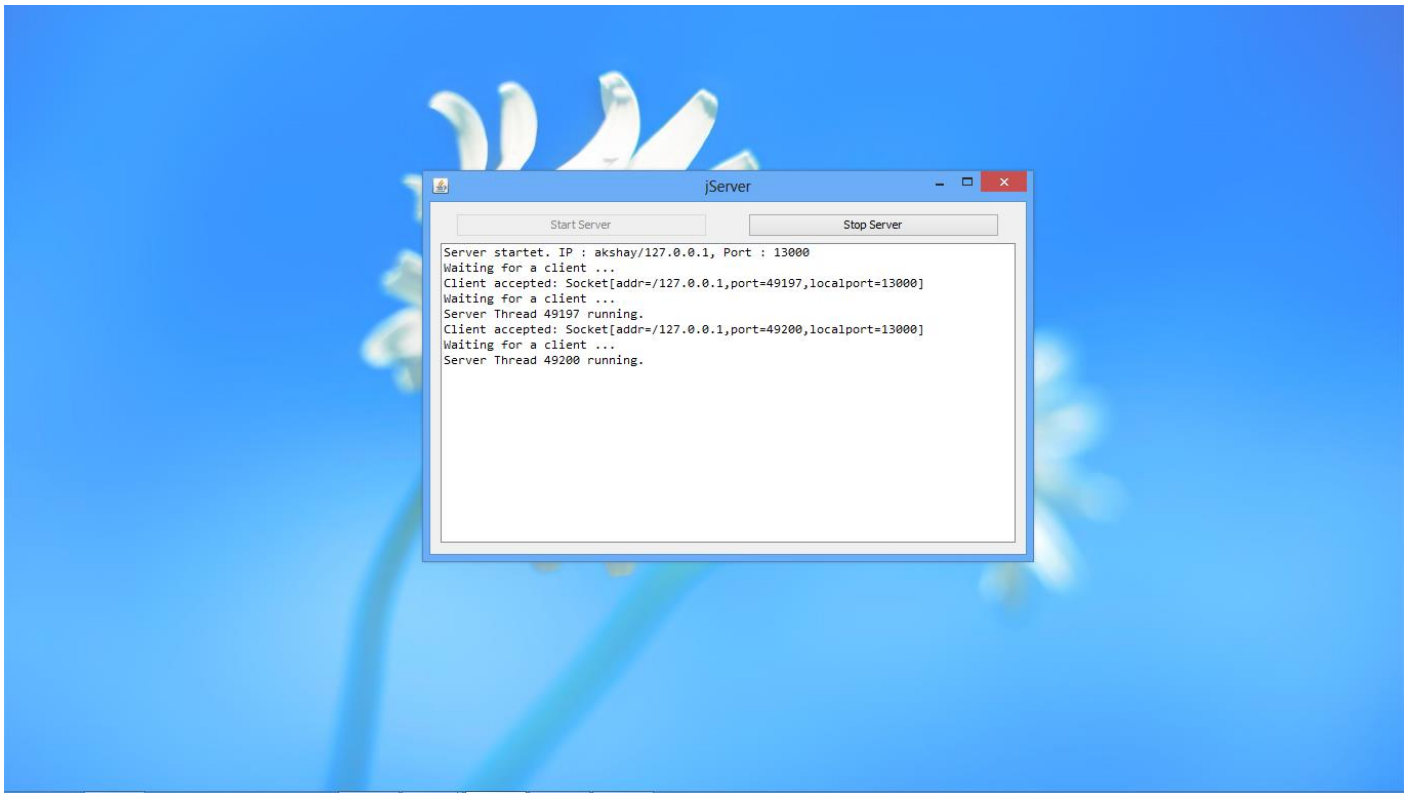

Figure 26: Server window on receiving connections

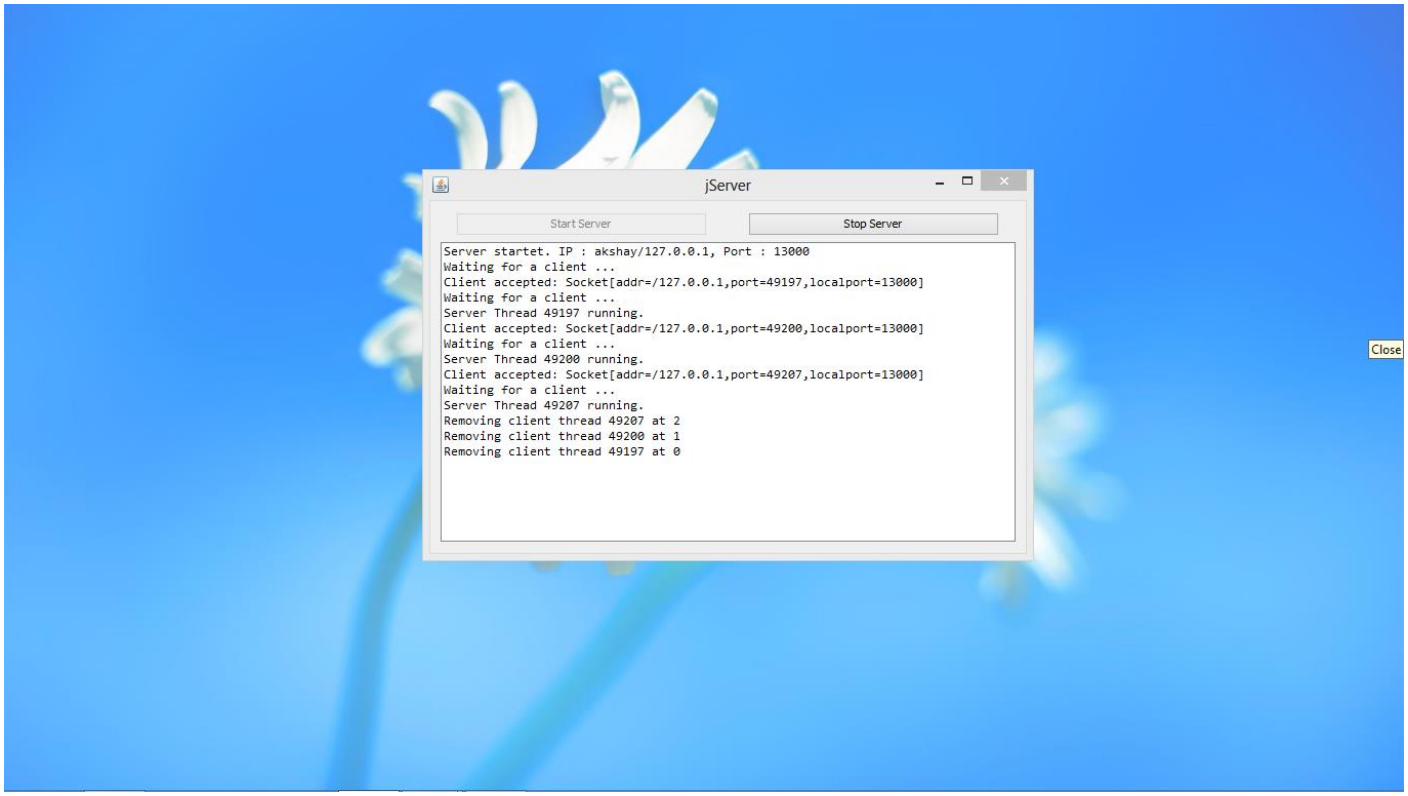

Figure 27: Server window on receiving messages and connections from Clients

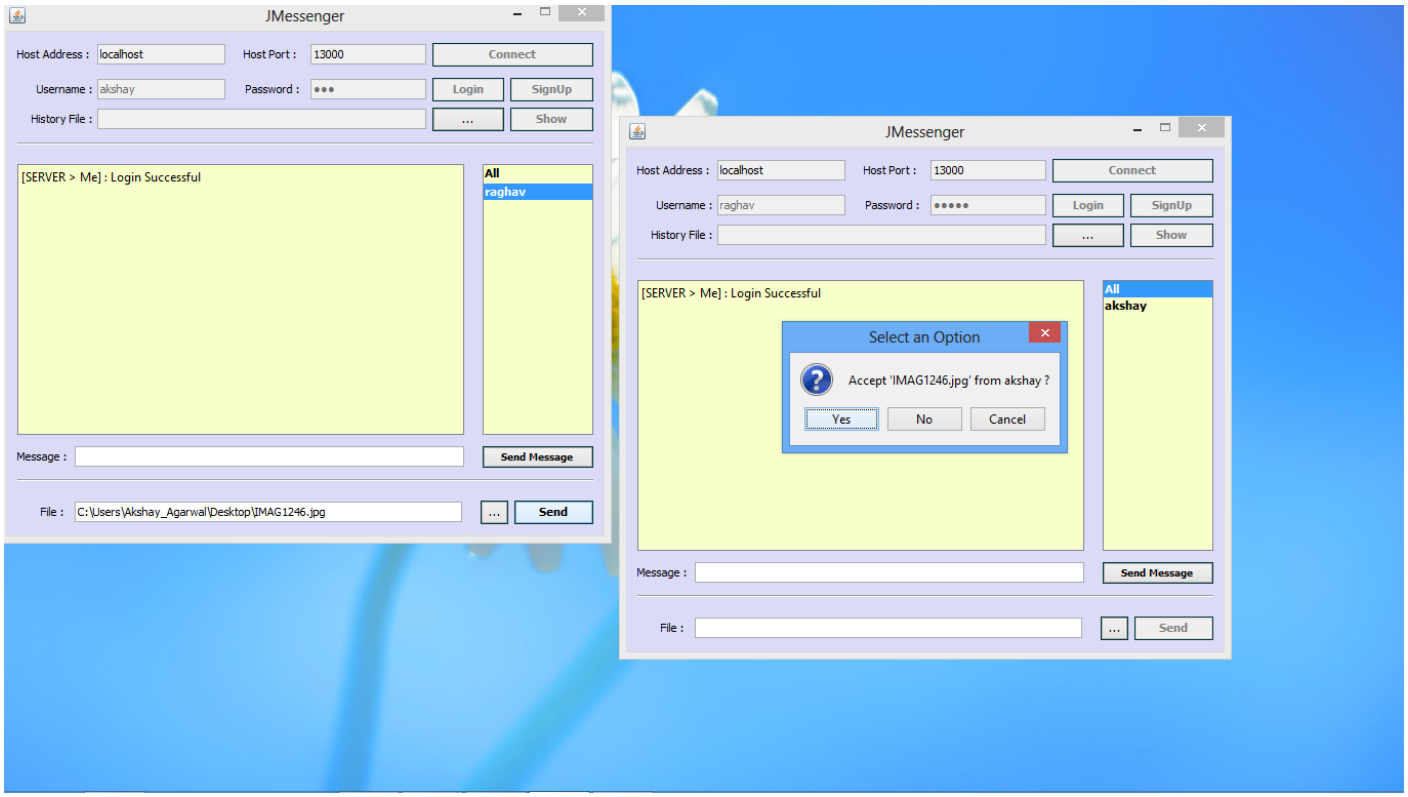

Figure 28: File acceptance dialogue box

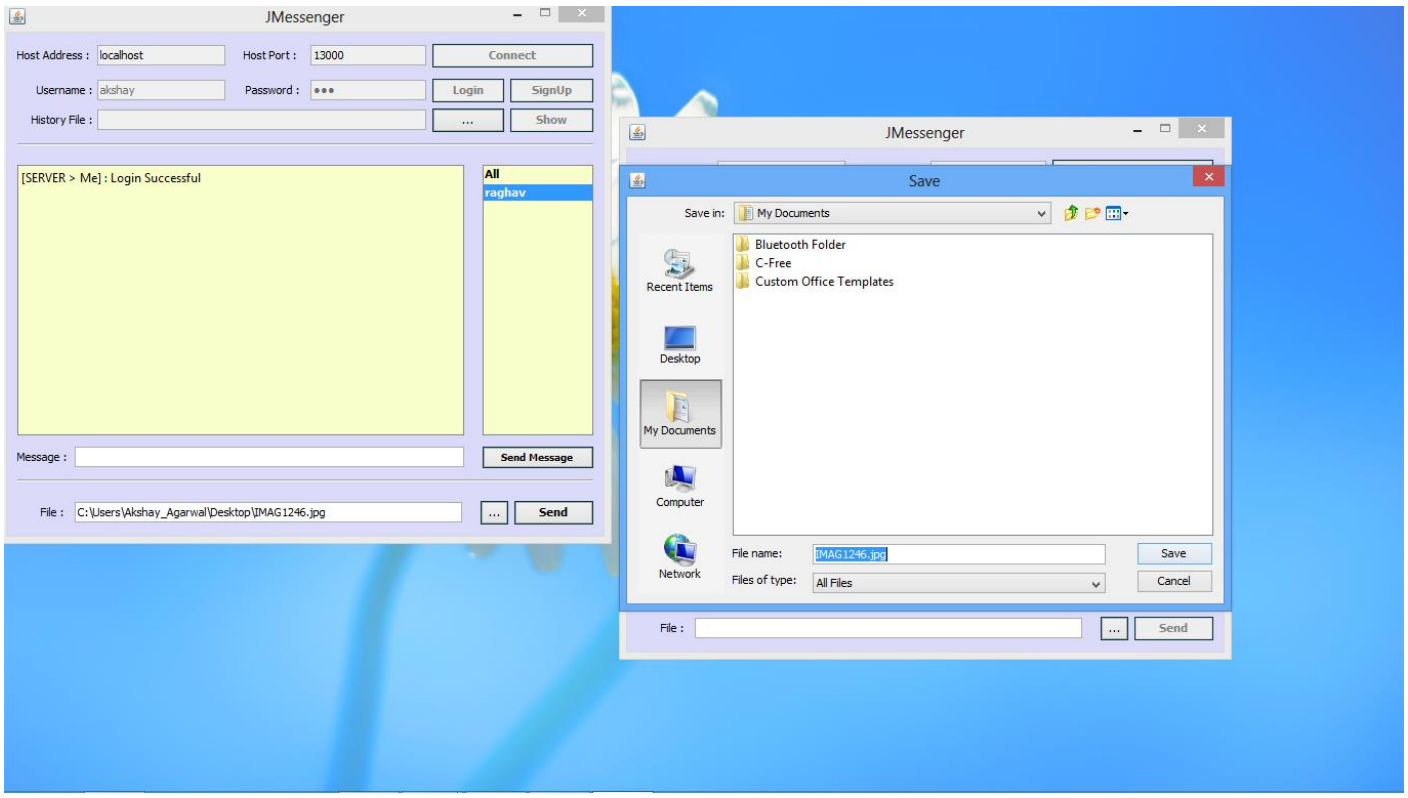

Figure 29: File saving dialogue box

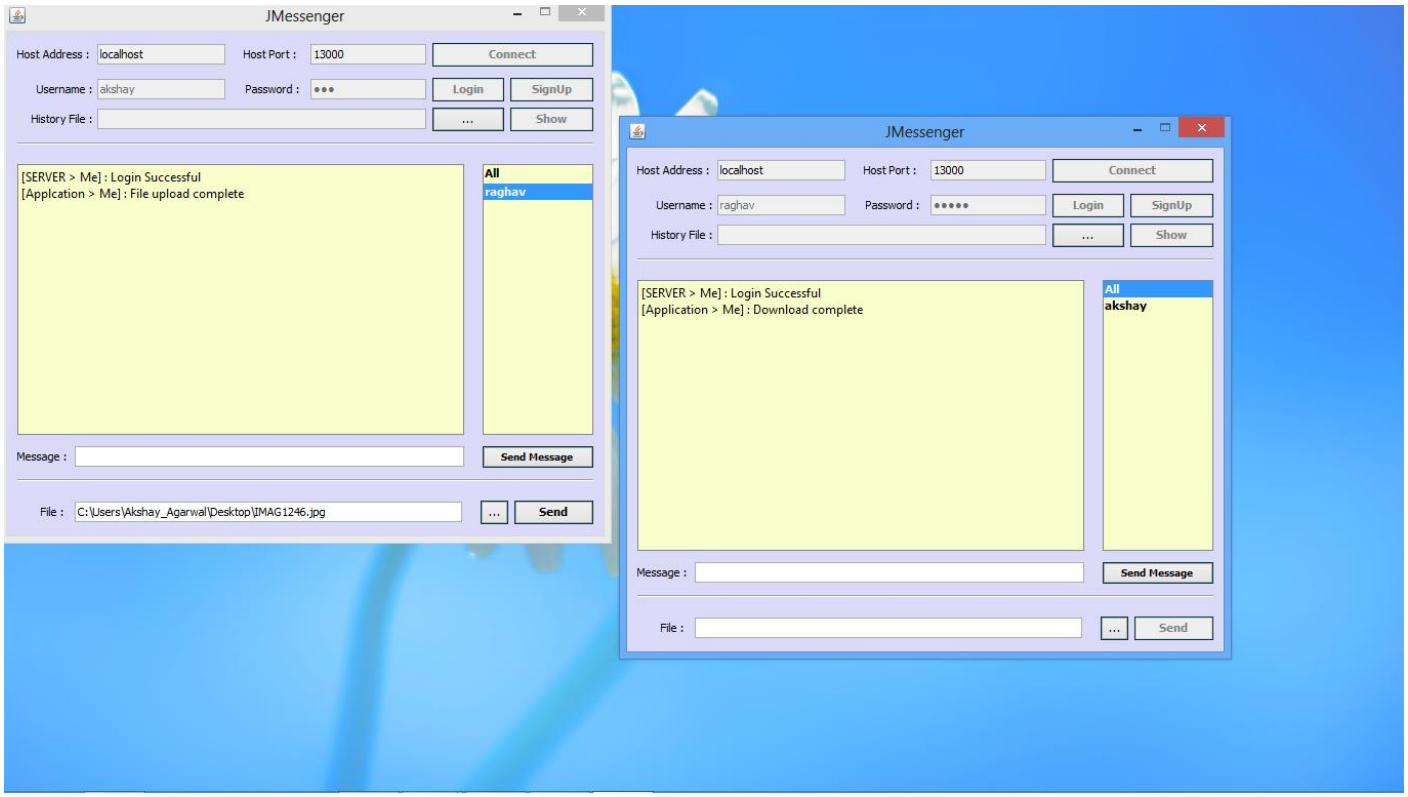

Figure 30: File upload and download complete notification

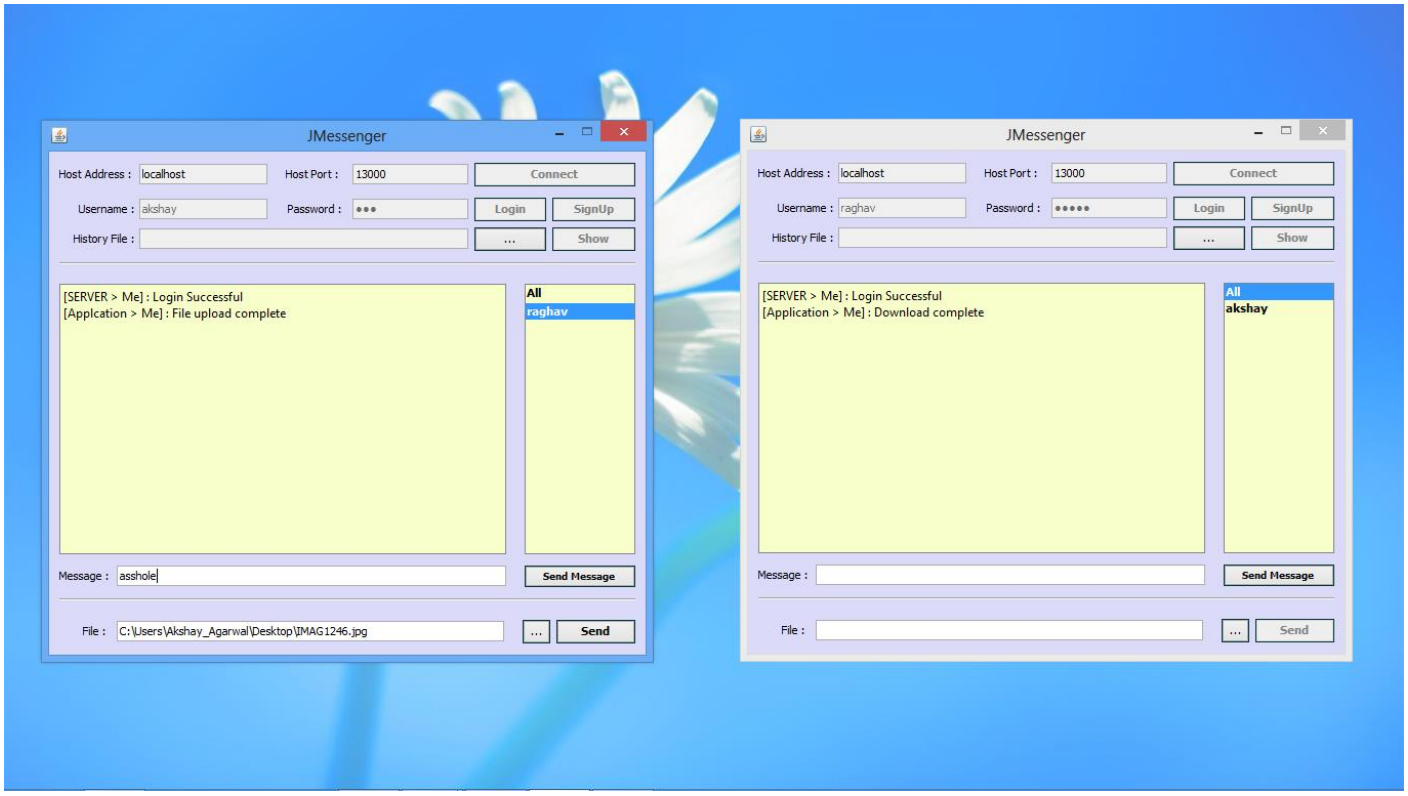

Figure 31: Sending an offensive message

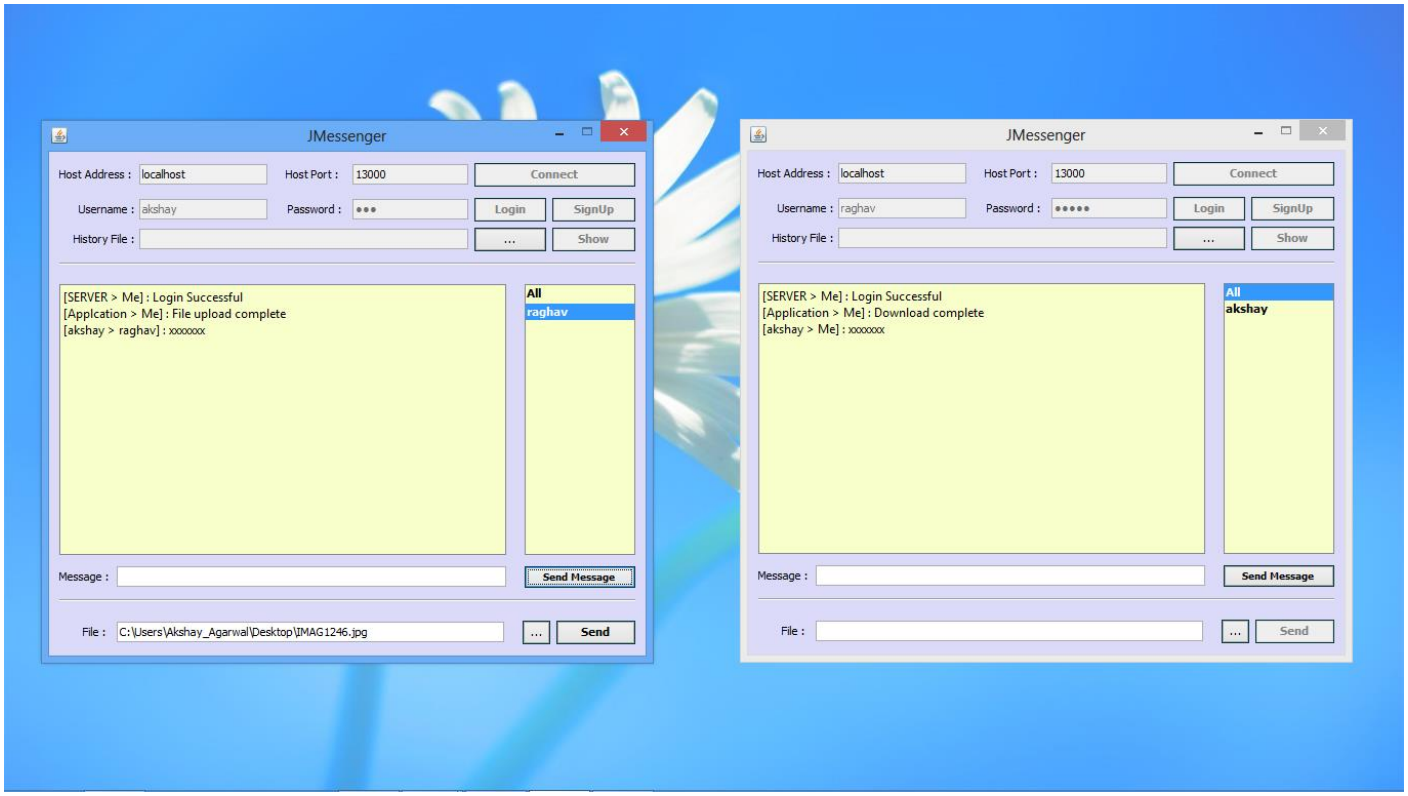

Figure 32: Censoring of the offensive message

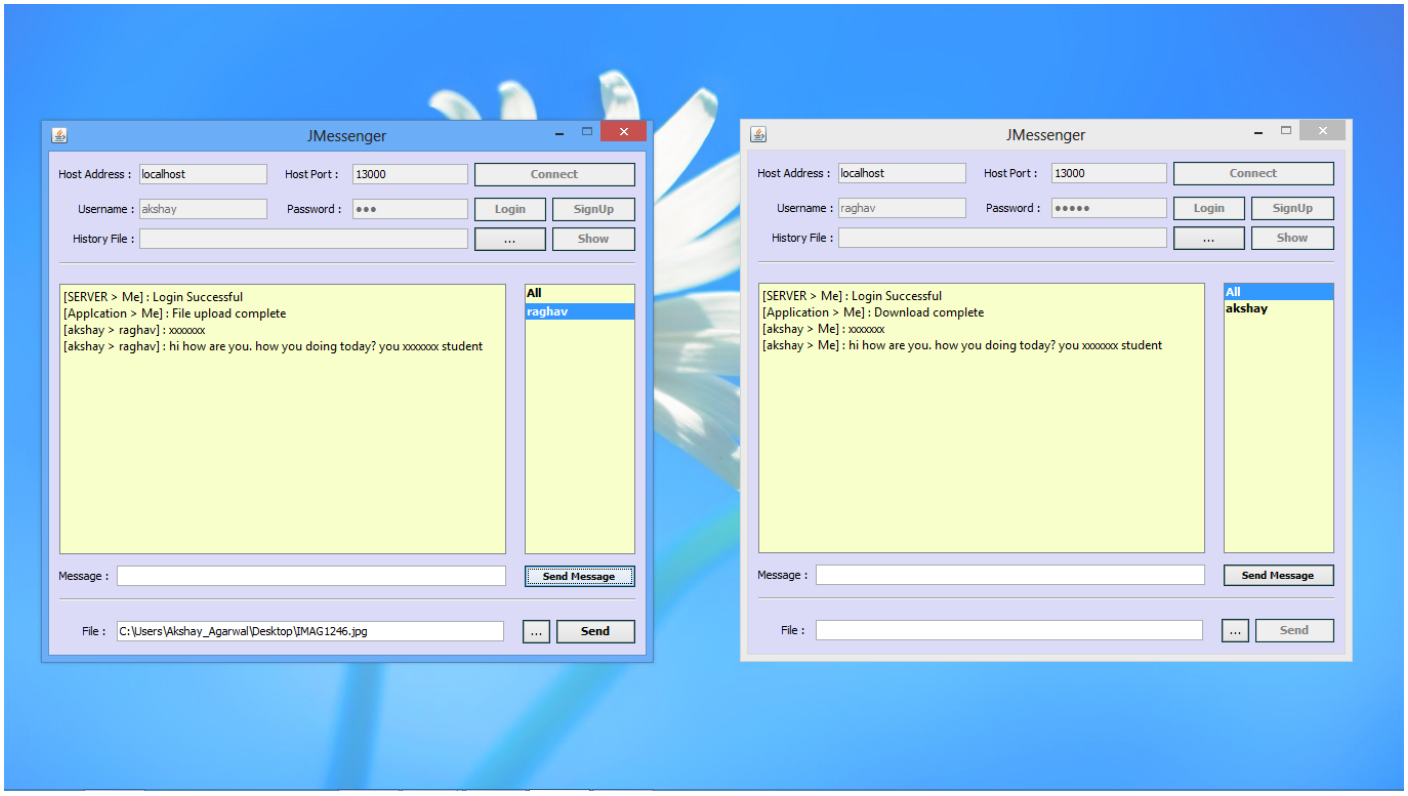

Figure 33: Censoring of the offensive message

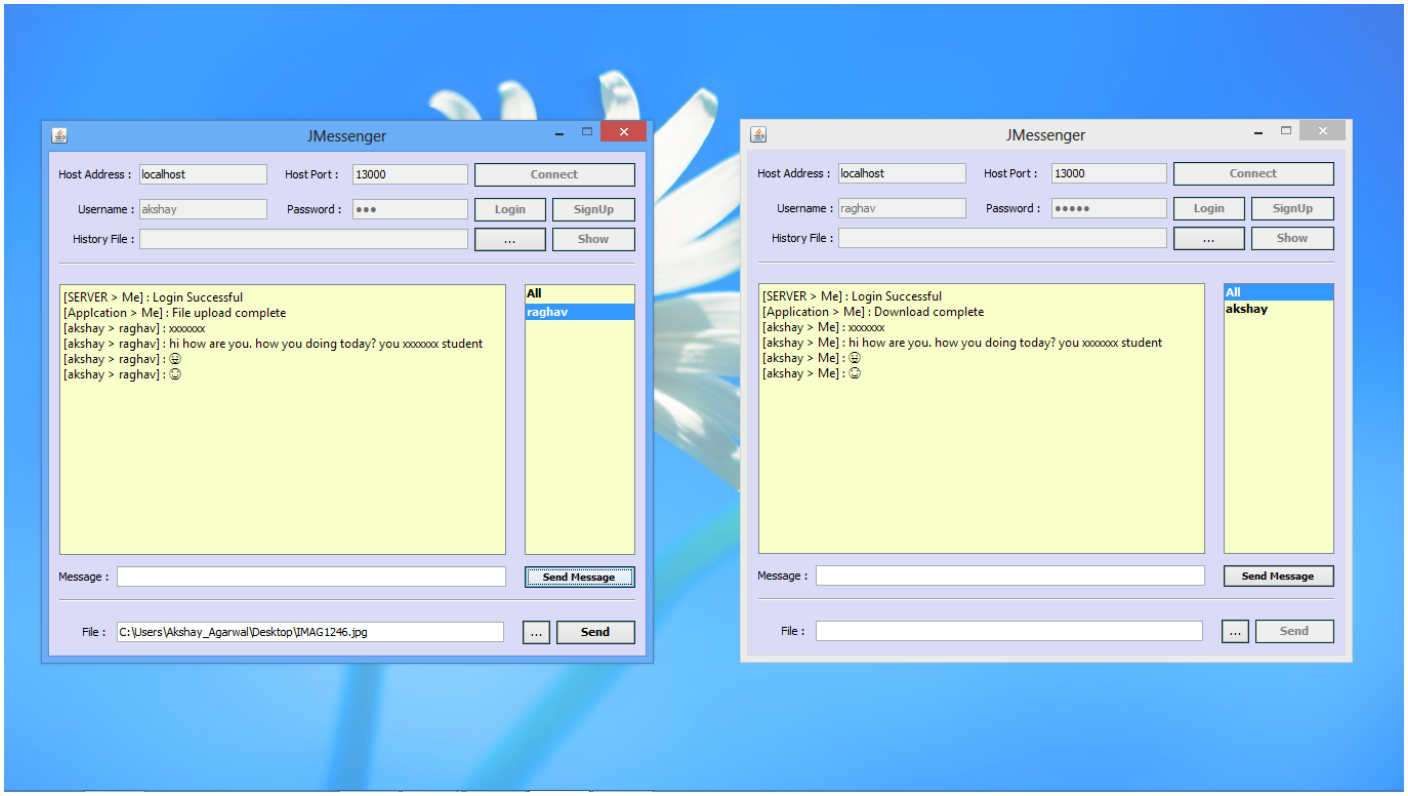

Figure 34: Messages displaying Emoticons

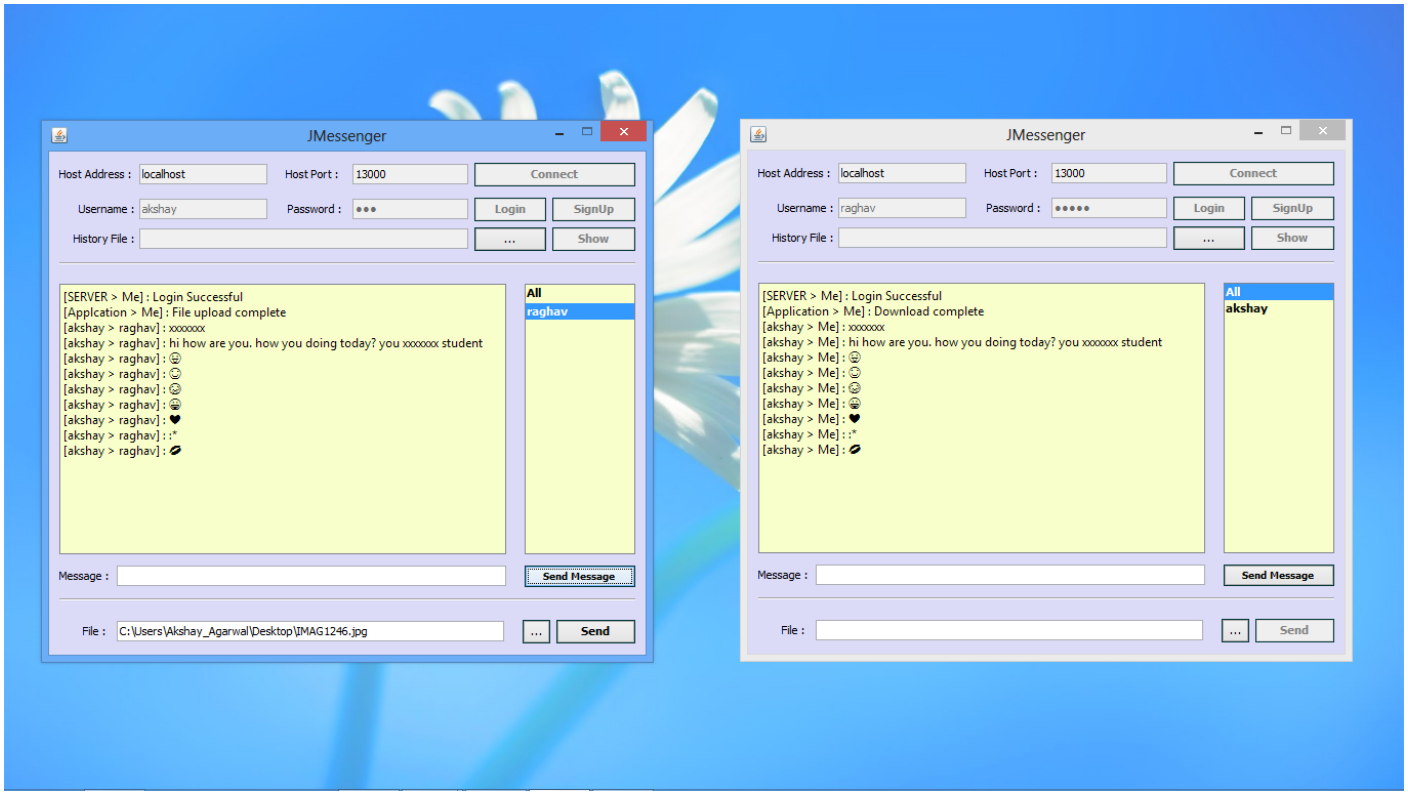

Figure 35: Messages displaying various Emoticons

3.4 AUTOKEY CIPHER

An **autokey cipher** (also known as the **autoclave cipher**) is a cipher which incorporates the message (the **plaintext**) into the key. It is a **polyalphabetic substitution cipher.** It is closely related to the **Vigenere cipher** but uses a different method of generating the key. It was invented by **Blaise De Vigenere** in 1586.

This cipher scheme has been used in the development of this application. It is used for the ensuring the secrecy of the password over the network and also in the database.

3.4.1 THE ALGORITHM

The "**Key**" for the autokey cipher is a key word, e.g."FORTIFICATION".

The autokey cipher uses the following tableau(the "**tabula recta**") to encipher the plaintext:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Table 1: Tabula Recta

3.4.1.1 ENCRYPTION

To encipher a message, place the keyword above the plaintext. Once all of the key characters have been written, start writing the plaintext as the key. E.g.:

Key: KING

Plaintext: meet me at the corner

Plaintext	m	e	e	t	m	e	a	t	t	h	e	c	o	r	n	e	r
Keystream	K	I	N	G	M	E	E	T	M	E	A	T	T	H	E	C	O

Figure 36: key Generation

With the keystream generate, we use the Tabula Recta, just like for the Vigenere Cipher. We find K across the top, and M down the left side. The ciphertext letter is "W". For the second letter "e", we go to I across the top and E down the left to get the ciphertext letter "M". Continuing in this way, we get the ciphertext "WMRZYIEMFLEVHYRGF".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Table 2: Tabula Recta showing encryption

So the ciphertext for the above plaintext is:

Plaintext	m	e	e	t	m	e	a	t	t	h	e	c	o	r	n	e	r
Keystream	K	I	N	G	M	E	E	T	M	E	A	T	T	H	E	C	O
Ciphertext	W	M	R	Z	Y	I	E	M	F	L	E	V	H	Y	R	G	F

Figure 37: Ciphertext

3.4.1.2 DECRYPTION

To decrypt a ciphertext using Autokey Cipher, we start just as we do for the Vigenere Cipher and find the first letter of the key across the top, find the ciphertext letter down that column, and take the plaintext letter at the far left of this row. As well as being the plaintext letter, we now need to add this letter to the end of the keystream as we shall need it later. Continuing to decode each letter, we add them to the end of the keystream each time.

We shall decrypt the ciphertext "QNXEPKMAEGKLAAELDTPDLHN" which has been encrypted using the keyword "queen". We start with the information shown below:

Plaintext																						
Keystream	Q	U	E	E	N																	
Ciphertext	Q	N	X	E	P	K	M	A	E	G	K	L	A	A	E	L	D	T	P	D	L	H	N

Figure 38: Decryption

We look along the top row to find the letter from the keystream, Q. we look down this column (in yellow) and find the ciphertext letter "Q" (in green). We then go along this row (in blue) to the left hand edge, and the letter here (in purple) is the plaintext letter. In this case it is "a".

We now add this to the end of the keystream, as well as to the plaintext row.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Table 3: Tabula Recta showing Decryption

Plaintext	a																						
Keystream	Q	U	E	E	N	A																	
Ciphertext	Q	N	X	E	P	K	M	A	E	G	K	L	A	A	E	L	D	T	P	D	L	H	N

Figure 39: Generation of the plaintext

In the same way as above, we find the keystream letter U, and find the ciphertext letter "N" in this column. We then follow this row to find the plaintext letter "t". Again we add this plaintext letter to the end of the keystream.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Table 4: Tabula Recta showing Decryption

Plaintext	a	t																					
Keystream	Q	U	E	E	N	A	T																
Ciphertext	Q	N	X	E	P	K	M	A	E	G	K	L	A	A	E	L	D	T	P	D	L	H	N

Figure 40: Generation of the plaintext and keystream

We then continue in the same way to retrieve the plaintext "attack the east wall at dawn".

Plaintext	a	t	t	a	c	k	t	h	e	e	a	s	t	w	a	l	l	a	t	d	a	w	n
Keystream	Q	U	E	E	N	A	T	T	A	C	K	T	H	E	E	A	S	T	W	A	L	L	A
Ciphertext	Q	N	X	E	P	K	M	A	E	G	K	L	A	A	E	L	D	T	P	D	L	H	N

Figure 41: Plaintext

3.5 CODE SNIPPETS

3.5.1 Code for censoring of the message and converting emoticon symbols to emoticons

```
public void Check()
   {
      try
      {
         String connectionURL ="jdbc:derby://localhost:1527/offensive words";
         String connectionURL1 = "jdbc:derby://localhost:1527/images";
         Connection conn = DriverManager.getConnection(connectionURL);
         Connection conn1 = DriverManager.getConnection(connectionURL1);
         PreparedStatement stmt = conn.prepareStatement("SELECT WORDS FROM
WORDLIST WHERE WORDS = ?");
         PreparedStatement stmt1 = conn1.prepareStatement("SELECT LINK FROM LINKS
WHERE EMOTICON = ?");
         ResultSet rs,rs1;
         int begin = 0;
         int end;
         int length = content.length();
         String newcontent = "";
         String temp = content;
         while ( begin < length )
         {
            end = temp.indexOf(' ');
            if(end == -1)
            {
               end=temp.length();
            }
            String checkword = temp.substring(0, end);
            int check = 0;
            stmt.setString(1, checkword);
            stmt1.setString(1, checkword);
            rs = stmt.executeQuery();
            rs1 = stmt1.executeQuery();
            if(rs.next())
            {
               //rs = stmt.executeQuery();
               for(int i=0;i<end;i++)
               {
                  newcontent+='x';
```

```
                }
                newcontent+=' ';
                rs.close();
            }
            else if(rs1.next())
            {
                String img = rs1.getString("link");
                 newcontent+=checkword.replace(checkword,img);
                newcontent+=' ';
                rs1.close();
            }
            else
            {
                newcontent+=checkword+' ';
            }
            begin += end+1;
            if(begin<length)
                temp = temp.substring(end+1,temp.length());
        }
        content = newcontent;
    }
    catch(Exception e)
    {
        System.err.println(e.getMessage());
    }
}
```

3.5.2 Code for decryption

```
public String decrypt(String password)
{
    /*---------------------------------------------------------------------
        code for checking the username and password here
        and if correct then proceed else return error and convey the same
        to the user as well.
        ---------------------------------------------------------------------
        */

  String inputPassword = password;
  String decClientPassword = "";
  try
  {
    /*
        ---------------------------------------------------------------------
        applying decryption to the password obtained from user
        ---------------------------------------------------------------------
                */
  String ClientKey = "sensor";
  int ClientKeyLen = ClientKey.length();
  int ClientPassLen = inputPassword.length();
  int i = 0;
  int temp = 0;
  while(i<ClientPassLen)
  {
     for(int j=0;j<26;j++)
     {
        int temp1 = ((int)ClientKey.charAt(i))-97;
        char a = table[j][temp1];
```

```
                char b = inputPassword.charAt(i);
                if(a == b)
                {
                   int t = j+97;
                   decClientPassword += Character.toString((char) t);
                   ClientKey += (char)j;
                   break;
                }
             }
             i++;
          }
       }
       catch(Exception e)
       {
          System.err.println(e.getMessage());
       }
       return decClientPassword;
    }
```

3.5.3 Code for encryption

```
String inputUsername = username;
        String inputPassword = password;
        try
        {
           String connectionURL ="jdbc:derby://localhost:1527/Users";
           Connection conn = DriverManager.getConnection(connectionURL);
           Statement stmt = conn.createStatement();
           ResultSet rs;
           rs = stmt.executeQuery("SELECT * FROM USERLIST ");
           boolean success = true;
           String ClientKey = "sensor";
           String decClientPassword = "";
           int ClientKeyLen = ClientKey.length();
           int ClientPassLen = inputPassword.length();
           int i = 0;
           int temp = 0;
           while(i<ClientPassLen)
           {
              for(int j=0;j<26;j++)
              {
                 int temp1 = ((int)ClientKey.charAt(i))-97;
                 char a = table[j][temp1];
                 char b = inputPassword.charAt(i);
                 if(a == b)
                 {
```

```
            int t = j+97;
            decClientPassword += Character.toString((char) t);
            ClientKey += (Character.toString((char) t));
            break;
        }
    }
    i++;
}
/*
-----------------------------------------------------------
applying encrytion to the password
in order to prevent password disclosure
-----------------------------------------------------------
*/
String encPassword="";
String key = "chatclient";
int passLen = decClientPassword.length();
int keyLen = key.length();
while(keyLen < passLen)
{
    key+=Character.toLowerCase(decClientPassword.charAt(keyLen));
    keyLen = key.length();
}
i = 0;
temp = 0;
while(i<passLen)
{
    temp = ((int)key.charAt(i)-97)+((int)decClientPassword.charAt(i)-97);
```

```
            temp%=26;
            temp = temp + 97;
            encPassword+=(char)temp;
            i++;
        }
        PreparedStatement prep = conn.prepareStatement("INSERT INTO USERLIST
VALUES(?,?)");
        prep.setString(1,inputUsername);
        prep.setString(2,encPassword);
        int a=prep.executeUpdate();
    }
    catch(Exception ex)
    {
        System.err.println(ex.getMessage());
    }
```

3.5.4 Code for file uploading

```
public Upload(String addr, int port, File filepath, ChatFrame frame)
{
      super();
      try
      {
         file = filepath; ui = frame;
         socket = new Socket(InetAddress.getByName(addr), port);
         Out = socket.getOutputStream();
         In = new FileInputStream(filepath);
       }
      catch (Exception ex)
      {
         System.out.println("Exception [Upload : Upload(...)]");
       }
   }

   @Override
   public void run()
   {
      try
      {
         byte[] buffer = new byte[1024];
         int count;

         while((count = In.read(buffer)) >= 0)
         {
            Out.write(buffer, 0, count);
         }
         Out.flush();

          try
          {
                    Document doc = ui.jTextArea1.getDocument();
                    doc.insertString(doc.getLength(),"[Applcation > Me] : File upload
complete\n", null);
                }
                catch(BadLocationException exc)
                {
                    exc.printStackTrace();
                }
```

```
ui.SelectFile.setEnabled(true); ui.SendFile.setEnabled(true);
            ui.jTextField5.setVisible(true);
            if(In != null)
            {
                        In.close();
            }
            if(Out != null)
            {
                        Out.close();
            }

             if(socket != null)
            {
                        socket.close();
            }
        }
       catch (Exception ex)
      {
          System.out.println("Exception [Upload : run()]");
          ex.printStackTrace();
       }
    }
```

3.5.5 Code for file downloading

```
public Download(String saveTo, ChatFrame ui){
   try {
      server = new ServerSocket(0);
      port = server.getLocalPort();
      this.saveTo = saveTo;
      this.ui = ui;
   }
   catch (IOException ex) {
      System.out.println("Exception [Download : Download(...)]");
   }
}

@Override
public void run() {
   try {
      socket = server.accept();
      System.out.println("Download : "+socket.getRemoteSocketAddress());

      In = socket.getInputStream();  // inputstream
      Out = new FileOutputStream(saveTo);  // fileoutputstream

      byte[] buffer = new byte[1024];
      int count;

      while((count = In.read(buffer)) >= 0){
         Out.write(buffer, 0, count);
      }

      Out.flush();

      //ui.jTextArea1.setText("[Application > Me] : Download complete\n");
            try
            {
```

```
                    Document doc = ui.jTextArea1.getDocument();
                    doc.insertString(doc.getLength(),"[Application > Me] : Download
complete\n", null);
                }
                catch(BadLocationException exc)
                {
                    exc.printStackTrace();
                }


            if(Out != null){ Out.close(); }
            if(In != null){ In.close(); }
            if(socket != null){ socket.close(); }
        }
        catch (Exception ex) {
            System.out.println("Exception [Download : run(...)]");
        }
            }
```

3.5.6 Code for saving the chat history

```
public class History {

   public String filePath;

   public History(String filePath){
      this.filePath = filePath;
   }

   public void addMessage(Message msg, String time)
   {
      try
      {
         DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
         DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
         Document doc = docBuilder.parse(filePath);

         Node data = doc.getFirstChild();

         Element message = doc.createElement("message");
         Element _sender = doc.createElement("sender");
         _sender.setTextContent(msg.sender);
         Element _content = doc.createElement("content");
         _content.setTextContent(msg.content);
         Element _recipient = doc.createElement("recipient");
         _recipient.setTextContent(msg.recipient);
         Element _time = doc.createElement("time");
         _time.setTextContent(time);

         message.appendChild(_sender);
         message.appendChild(_content);
         message.appendChild(_recipient);
         message.appendChild(_time);
         data.appendChild(message);
```

```
            TransformerFactory transformerFactory = TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(new File(filePath));
            transformer.transform(source, result);

            }
        catch(Exception ex)
        {
            System.out.println("Exceptionmodify xml");
            }
    }

    public void FillTable(HistoryFrame frame)
    {
        DefaultTableModel model = (DefaultTableModel) frame.jTable1.getModel();
        try
        {
            File fXmlFile = new File(filePath);
            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(fXmlFile);
            doc.getDocumentElement().normalize();

            NodeList nList = doc.getElementsByTagName("message");

            for (int temp = 0; temp < nList.getLength(); temp++)
            {
                Node nNode = nList.item(temp);
                if (nNode.getNodeType() == Node.ELEMENT_NODE)
                {
                    Element eElement = (Element) nNode;
                    model.addRow(new Object[]{getTagValue("sender", eElement),
getTagValue("content", eElement), getTagValue("recipient", eElement), getTagValue("time",
eElement)});
                }
            }
        }
```

```
 catch(Exception ex)
       {
          System.out.println("Filling Exception");
       }
    }

    public static String getTagValue(String sTag, Element eElement) {
          NodeList nlList = eElement.getElementsByTagName(sTag).item(0).getChildNodes();
       Node nValue = (Node) nlList.item(0);
          return nValue.getNodeValue();
  }
}
```

CHAPTER 4: CONCLUSION AND FUTURE WORK

The application building part of the overall application has come to an end, where I have pointed out the major applications which is desired by general audience. The final Censored LAN Messenger application is an application where user will be able to communicate with any person who is connected to the same Local Area Network. The person will be able to send group messages as well as private message to other online users. The person will also be able to share files with other users and there is no limit to the size of file that a user can share. The use of emoticons is also possible with the final version of the application.

The future work for this application will involve upgrading the security algorithm used and expanding the ability of the application to censor more offensive words. Also, the next version would support other new emoticons that haven't been included at this stage.

BIBLIOGRAPHY

Davie, L. L. (2000). Computer Networks A Systems Approach, Harcourt Asia PTE LTD, second edition. Harcourt Asia/Morgan Kaufman.

Lyons, J. (n.d.). Cryptography. Retrieved from Practical Cryptography: www.practicalcryptography.com

Marc Loy, R. E. (2002). Java Swing. O'Reilly Media, Inc.

Oracle. (n.d.). Java Tutorial. Retrieved from Sun Java Tutorial: www.java.sun.com

R.S., P. (1997). Software Engineering: A Practitioner's Approach, Fourth Edition. McGraw Hill.

Schildt, P. N. (2002). The Complete Reference Java 2, Fifth Edition. McGraw-Hill.

Singh, S. (n.d.). Autokey Cipher. Retrieved from Interactive Maths: www.crypto.interactive-maths.com/autokey-cipher

Stallings, W. (2005). Cryptography and network security principles and practices 4th edition. Pearson.