JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

Department of Information Technology

Unit code: DIT 0402

Unit Name: ICT PROJECT II

Simple Socket Chat Application

Supervisor

DR DENNIS KABURU

(*PHD*)

Submitted by

KEVIN NALIANYA.

SCT121-D1-0010/2023

August 15, 2024

Submitted for partial fulfillment of the requirements of the diploma in information and technology

Table of contents

Declaration

This report is submitted as part requirement for the diploma in Information technology . It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Signatures:

Supervisor: **Sign:** ..... **Date:**	**Student:** **Sign:**  **Date:**.....**15/08/2024**.....

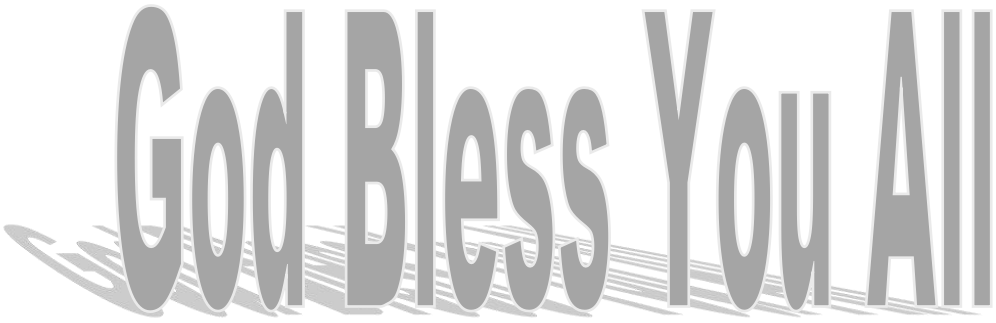
Acknowledgement

With all special regards I would like to acknowledge all the people who have contributed in one way or another to the success and completion of this project.

First of all I would like to give my thanks to the project supervisors for taking much of their time to dedicate themselves to the success of this project by giving value instructions and directions that were of great meaningful importance.

It would also be a crime to forget my classmates and friends whose moral support criticism and encouragement real helped me a great deal. I owe so much to them.

And may all the Glory and Honor go to the Almighty God for this far He has brought us.



Abstract

Live Chat applications are widely in use as far as the social aspect of man machine interaction is concerned to relay information in real time from one end to another. This project therefore is carried out with an aim of coming up with simple Live chat application that implements sockets. The goal was birthed from analyzing of existing chat applications that uses sockets and coming up with a simple chat application that implements socket programming.

Socket programming uses the client socket and server socket methods to connect the local host to the named host and port.

This project is an example of client server. It is made up of two applications the client application, which runs on the user's PC and server application, which runs on any PC on the network.

To start chatting client should get connected to server where they can practice two kinds of

Chatting, public chatting where a message is broadcasted to all connected users and private chatting which involves a chat between two users only.

Prototyping formed a basis for requirement specification elicitation and foundation for support of future works related while agile methodology became the foundation for this development.

Data analysis from the existing chat applications was done to give a clear picture of the needed chat application and the various properties that it will have. The analysis dwelt majorly on socket applications and ignored chat systems that don't apply sockets because this project was specifically for socket implementations.

The main challenges found with the currently used private chat applications is that they are very expensive to own especially for small scaled companies yet they face stiff competition from live chat that are deployable free of charge like

Google ,Gmail, Facebook and so on. Insecurity is also an issue with this kind of applications as they are open to anyone connected.

This new system was developed using C# programming language as per the initial requirements but future enhancement will involve the use of web technology languages like PHP, MySQL, JavaScript and CSS and others like Python, C++,Perl or Java.

Recommendations were then made on how the proposed system should be implemented and enhanced for other expansions and the proposed new live chat was found to be faster, efficient, secure and flexible to accommodate changing needs.

INTRODUCTION

Today people use computers to process, store, retrieve and disseminate information from one place to another. All this is possible because of the ability of programs to communicate with each other.

Sockets are the basis upon which this mystery of programs to communicate with each other over a network comes to a reality.

Our main focus will be on Chat applications and the use of socket Application Programming Interface (API) in such applications.

Live Chat applications are widely used in social networking and in e-commerce to offer flexible and affordable means to relay information in real time. Social networking sites like Google Gmail, Facebook, tweeter and yahoo are in the fore front of promoting social networking via chat.

Most social live chat applications have been developed using web technology which we will look into later. They are usually available free of charge to the user who has accounts on those sites.

E-commerce live chats are deployable at a fee and widely used in teleconferencing to bring about communication in real time though offline support is also available through chat bots that redirect users to forms to send messages that are stored.

Live chats also come handy with messaging systems like yahoo messenger, whatsup and Skype.

Live chats are also in use in customer support and in future Customer Relation Management (CRM) systems live chat can be very helpful.

This chapter will cover key project areas like the problem statement, aims and objectives and the scope of the project.

Statement of the Problem

During the initial investigation of the current chat applications it was note that:

Current chat applications are very expensive to implement at personal level or for Small and Medium Enterprises (SMEs).

Most chat applications are community chat applications like Facebook chat and they are used by most people on social media and rarely used in company activities.

Mobile Chat applications are the new trend giving better capability for mobile chatting with developers like *Google, whatsup, Android* being on the cutting edge.

Chat applications potentials have not been fully met as poor implementation strategies are executed.

Aim of the Study

The aim of this project is to design and develop a simple chat application that implements sockets.

Objectives of the Study

In fulfilling the above aim the following are the objectives of the study

- To analyze the current chat applications in the market
- To identify the challenges facing the existing chat application and develop alternative solutions to solve the problem if any.
- To establish the need for a new and improved chat application.
- To design and develop a faster and convenient chat applications.

Scope the Project

The scope of this project is to design and develop simple chat application that implements socket programming. It is basically involved with the analysis, design, development and implementation of this chat application.

LITERATURE REVIEW

Introduction

This chapter deals with the review of literature related to the system being developed. It also shows the current state of knowledge based on system being developed.

Books and the internet resources were used as the source of information as they covered the topics more extensively in timely way.

Here the literature review specifically projected and concentrated on the ***live chats*** and other chat applications that exist today.

However this area of study is concerned with the design and development of a simple chat application that makes use sockets website using C# programming language, literature on the following topics were considered relevant to the system being developed.

Key terms

Server

According to Michael 2001 the server's job is to set up a communication endpoint and passively wait for a connection from the client.
Therefore a Sever is a system (software and suitable computer hardware) that responds to requests across a computer network to provide, or help to provide, a network service.
He continues to add that servers can be run on a dedicated computer, but many networked computers are capable of hosting servers. In many cases, a computer can provide several services and have several servers running.

A server therefore handles requests form a client.

Fig 1.0 Showing Server hardware. Source: Google images

Client

According to Michael 2001 a client's job is to initiate communication with a server that is passively waiting to be contacted.

A client therefore is a user system which uses its server to get its program or action executed.

A client sends a request to the server and the server handles that request and sends a reply to the client.

Fig 1.1 Showing client server. Source: Google images

Socket

According to Oracle online definition a ***socket*** is one endpoint of a two-way communication link between two programs running on the network.

A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

An endpoint is a combination of an IP address and a port number. Internet socket APIs are usually based on the Berkeley sockets standard.

Port number

According to Bradley 2015 a *port number* is part of the addressing information used to identify the senders and receivers of messages. Port numbers are most commonly used with TCP/IP connections.

After familiarizing ourselves with such terms we can now look at the various works related to this project:

Server client Action

According to Farah 2000 to initiate a client server connection one of the programs should take the role of a client while the other must take the role of a server. Farah 2000 continues to add that the client application initiates what is called *active open*. It creates a socket and actively attempts to connect to server program. On the other hand, the server application creates a socket and passively listens for incoming connections from client, performing what is called *passive open*.

The following is a summary of how this communication is initiated as listed by Farah 2000

The server side would follow these steps:

- Create a socket
- Listen for incoming connections from clients
- Accept the client connection
- Send and receive information
- Close the socket when finished, terminating the conversation

In case of the client, these steps are followed:

- Create a socket
- Specify the address and service port of the server program

- Establish the connection with the server
- Send and receive information
- Close the socket when finished, terminating the conversation

Types of Chat

There are two types of Chat according Verma 2010

1. **Synchronous Chat**: This are the form of chat rooms that entails a user typing text message and it is synchronously viewed by multiple users connected. Messages do appear in the order they were sent example is in whatsapp.

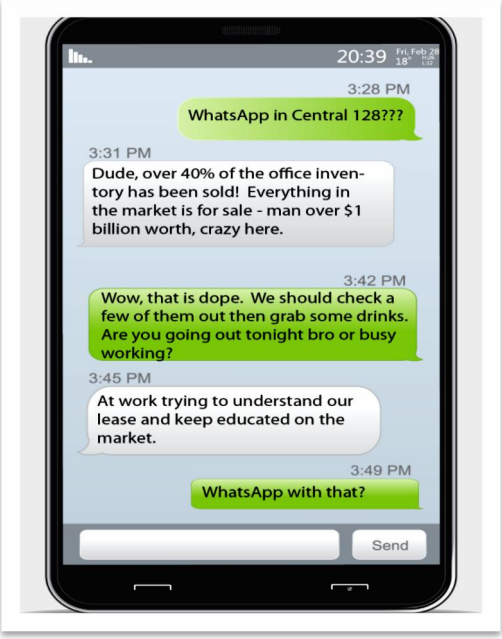


Fig 1.3 Showing whatsup chats. Source: Google images

2. **Asynchronous Chat**: Asynchronous Chat is way of saying Forum, Message Board or Bulletin Board.

Synchronous Chat

We will look at this as it is the area we are interested in mostly. There are three types of synchronous chats as described by Verma 2010.

Web-based chat rooms:

They are virtual forums dedicated to messages on particular topics written and read in real time by all individual present in the room.
Many popular search engines have their own chat platform and can be run under most operating systems, provided that the software that allows you to communicate with others in the chat room has been installed. (Verma, 2010).

Real-time chat:

Real-time chat sometimes referred "instant messaging," (IM) offers users the ability to communicate in real time with individuals who have been added to a "buddy" (friends) list.

Real-time chat is messaging software that lets you communicate with friends and colleagues in real time. Features include a quick launch bar, music file support, and a new answering service. Some versions also work with Internet Transmission Control Protocol (TCP)/Internet Protocol (IP) connections, allowing communication behind firewalls. (Verma, 2010).

Instant Messaging has been a growing trend since computers gained their power to communicate. Popular brands are Skype Messenger, eBuddy, Ms Windows Live Messenger and Yahoo! Messenger.

According to Hardikar, 2007 IM uses internet or intranet as the underlying medium. Generally IM allows synchronous textual interaction between two or more people.

IM has become a growing trend among teenagers (Grinter &Palen , 2002), though this is shifting focus to the work place but recent publications have criticized due to many dangers that it might pose to organization if not well handed.

The latest trend is the mobile instant messaging which has taken forms of text (SMS) or applications that run IM on mobile phones like whatsup, Facebook IM and Gmail hangout. Social media giants have greatly embraced this idea and almost all of them have an IM .

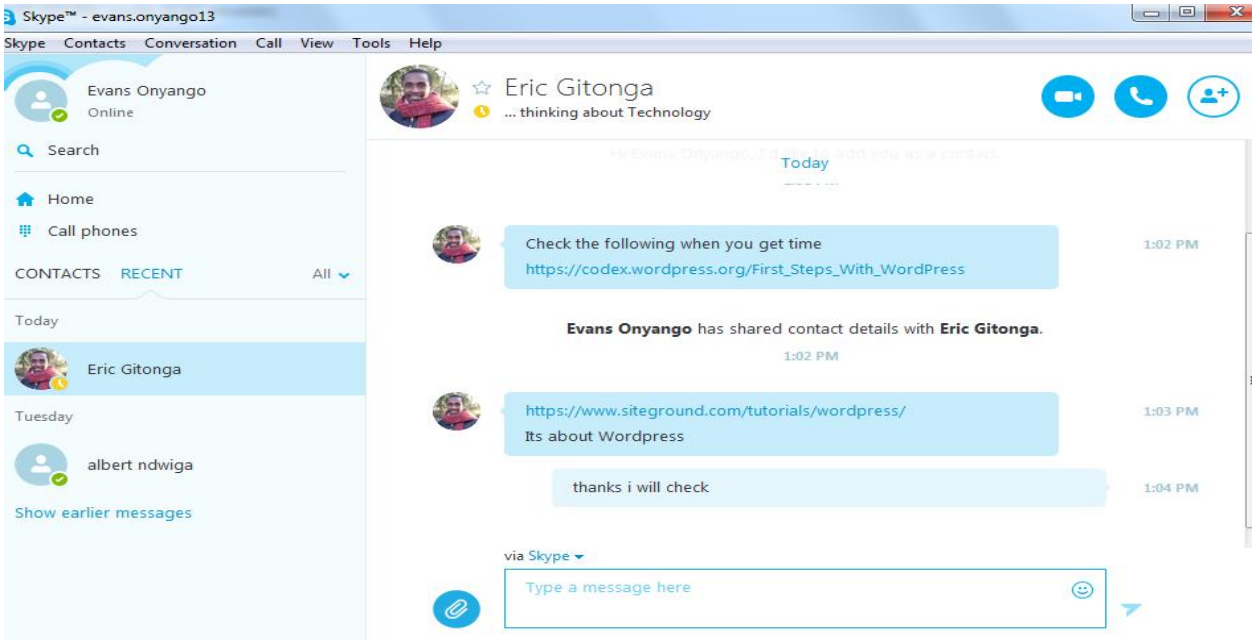


Fig 1.4 Showing Skype IM. Source: Skype

Internet Relay Chat (IRC)

IRC enables users throughout connected to the same network to communicate in real time.

For anyone to be connected, he/she must install an IRC client on his/her PC in order to communicate to the IRC server that serves each client.

IRC differs from *real-time chat* in that conversations are centered on a particular theme within a network of people s/he is not likely to encounter in real space. (Verma, 2010) .

Advantages of Internet chatting:

- Faster way of communicating in real time
- Expressing your own opinion on different matters
- Further clarification can be sort
- It can be an easier way of getting in touch with friends
- Chance of creating networks.

Disadvantages of Internet Chatting:

- Identity theft cases are rampant where by people use the identity of others
- Insecurity issues where human trafficking is involved.
- Mistrust is prevalent you may be chatting to a machine or a fake.
- They are places where idleness and laziness can be perfected.
- Cyber stalking Cyber bullying is at its peak especially among teens.
- It promotes machine dependent social traits whereby individuals shy away from physical social relationships.

Differences between the three

Here are the differences between IM, IRC and Web-based Chat according to Verma, 2010

IRC	IM	WEB-Based
• A user client must connect to the server in order to view the other users connected. • It has its chat moderator	• In IM one has to build one's own list of users or chat-buddies by inviting them to join. There is no need to log onto any server or chat-room and chat is not moderated. • With certain IM clients one can even leave an offline message (e.g. ICQ)	• For this kind of chat one has to enter a web-site that offers this service. • Such sites are Yahoo.com, Facebook and other sites that offer chat as an add-on to attract traffic.

Fig 1.5 Table showing differences of IRC, IM, Web-based chat

Live chat support system

These are systems that are used to offer support services to end users of a site or customers.

According to Ross 2006 a live chat service is either initiated by a client or a live chat agent.

They are usually deployed on E-commerce sites and are usually used to respond to user queries to offer help and support.

Fig 1.6 Showing live chat support. Source: Google images

Video chatting/Video conferencing

Video conferencing is a set of telecommunication technologies which allows two or more locations to communicate by simultaneous two-way video and audio transmissions. It is a type of groupware.

According to Rhodes 2001 there are two factors that unify these disparate technologies and applications:

- They use moving video images (of people, places and things) to enhance the communications process.
- The transmission of this video (and accompanying audio) involves two-way interactive communication.

In the corporate settings they are used to form virtual meetings and conferences which saves on time and cost of travel and boarding.

Fig 1.7 Showing video conferencing. Source: Google images

Conclusion

A port is a special location in the computer's memory that exists when two computers are communicating via TCP/IP. (Farah, 2000).
Computers communicate with each using socket API provided by the operating system. A socket is composed of an IP address and a port number and this forms a unique number that delivers incoming data packets to the appropriate application process or thread.
Chat applications are based on sockets where real time communication between different users takes place.
There are different chat applications that serve different users and offered by different developers or operators.

REQUIREMENTS AND ANALYSIS

System Requirements

In this chapter we will try to bring out what the system will do from the user point of view. These requirements will entail specifications that involve:

Network requirements

There must be a connection between the server and the client as in the internet. Local host connection can also work in purposes of testing the system.

Several users can connect to the server that connects them via the internet for example in Local Area Network setup many users in that learn can chat together using one dedicated server.

The dedicated server ought to be installed in the server room with all the networking components installed.

Hardware requirements

All users should have dedicated access to a computer using a modern operating system such as Windows 10, XP or Mac OS X or Linux based Operating Systems.

The Chat Server computer should hold a higher processor speed, i.e. core i5 for windows and larger Random access Memory (RAM) of minimum of 8GB to cater for all the users that may be connected. The Hard- disk capacity ought to be large to hold more data like 1 Terabyte for startup.

A Universal Power Server (UPS) can also be of benefit to facilitate the server to ensure no downtime.

The chat client computer can have a minimum of 4 GB RAM, Pentium 4 processor and 256GB Hard-disk.

CD's DVD's flash disk and External Hard-disk are of value when offsite data backup may be needed.

The chat server can be hosted in a server room 10m by 10m housing a cabinet.

Software Requirements

The chat server computer ought to have a running server operating system; example windows server 2008 and the chat client computer should have an operation operating system.

The chat server application has to be installed in the server computer to make the chat server and the chat client application to be installed in the user computer to make the chat client.

Visual studio 2019 is needed to build and run this application especially for the demo part.

Functional Requirements

The chat client application should be able to send and receive chat messages to and fro the chat server application.

The chat client application should alert when message has been sent and received.

The chat server application should be able to receive and send shat messages to and fro any chat client application.

The chat server application should be able flexible to accommodate users as they come and go.

Developing environment requirement

Microsoft Visual studio 2019 was the choice environment to develop run and test the application.

Operating Environment

This live chat server will be mainly operated on windows or UNIX. The operator client can be operated on both UNIX and Windows.
We chose to stick with MS windows operating system

Design and Implementation Constraints

The language was restricted to English however future translations are available

Assumptions and Dependencies

It is assumed that the requirements described in this document have different levels of priority.

User Classes and Characteristics

The application assumes no special knowledge of user. Commands shall be provided via menus, tool palettes, or help screens. Users shall be protected from data loss.

Summary of tools requirement and estimated prices in Kenya shillings

Hardware Requirements	Price(KSH)	Software Requirements	Price(KSH)	Other	Price(KSH)
2 Laptops or PC	70000	Windows 10 installed in both laptops	NULL	Skills	NULL
4 GB Flash disk	1000	Microsoft Visual Studio	NULL	Internet connection	NULL
DVD's and CD's	100	QT software	NULL	Data Bundles	1000
500 GB External Hard-Disk	5000				
4G Modem	3000				

Fig 1.8 Table showing tools required to develop the application

SYSTEM DESIGN

This chapter is concerned with the developing the system by using the user specifications and also putting into consideration all the inputs, outputs and process expected.

The design document describes a plan to implement the requirements – the architecture of the system. (Austin, 2014).
The main tasks involved here are user interface design, data design, and system architecture.

Design Considerations

The best design consideration our chat application ought to achieve is the ability of the system to be effective, reliable, and maintainable.
Rosen 2014 gives features a system has to carter for it to qualify as such:

- A system is reliable if it handles input errors, processing errors, hardware failures, or human mistakes. A good design will anticipate errors, detect them as early as possible, make it easy to correct them, and prevent them from damaging the system itself.
- A system is maintainable if it is flexible, scalable, and easily modified. Changes might be needed to correct problems, adapt to user requirements, or take advantage of new technology.
- A system is effective if it supports business requirements and meets user needs.

User Interface Design

A user interface (UI) describes how users interact with a computer system, and consists of all the hardware, software, screens, menus, functions, output, and features that affect two-way communications between the user and the computer.(Rosen ,2014).

Fig 1.9 showing example of UI chat application. Source: Google images

Human-Computer Interaction

According to Rosen Blatt 2014 Human Computer Interaction (HCI) describes the relationship between computers and people who use them to perform their jobs.

The chat application ought to be HCI compliant in that it must bring out a nice user experience when interacting with the system.

Data Flow diagram for the Proposed System

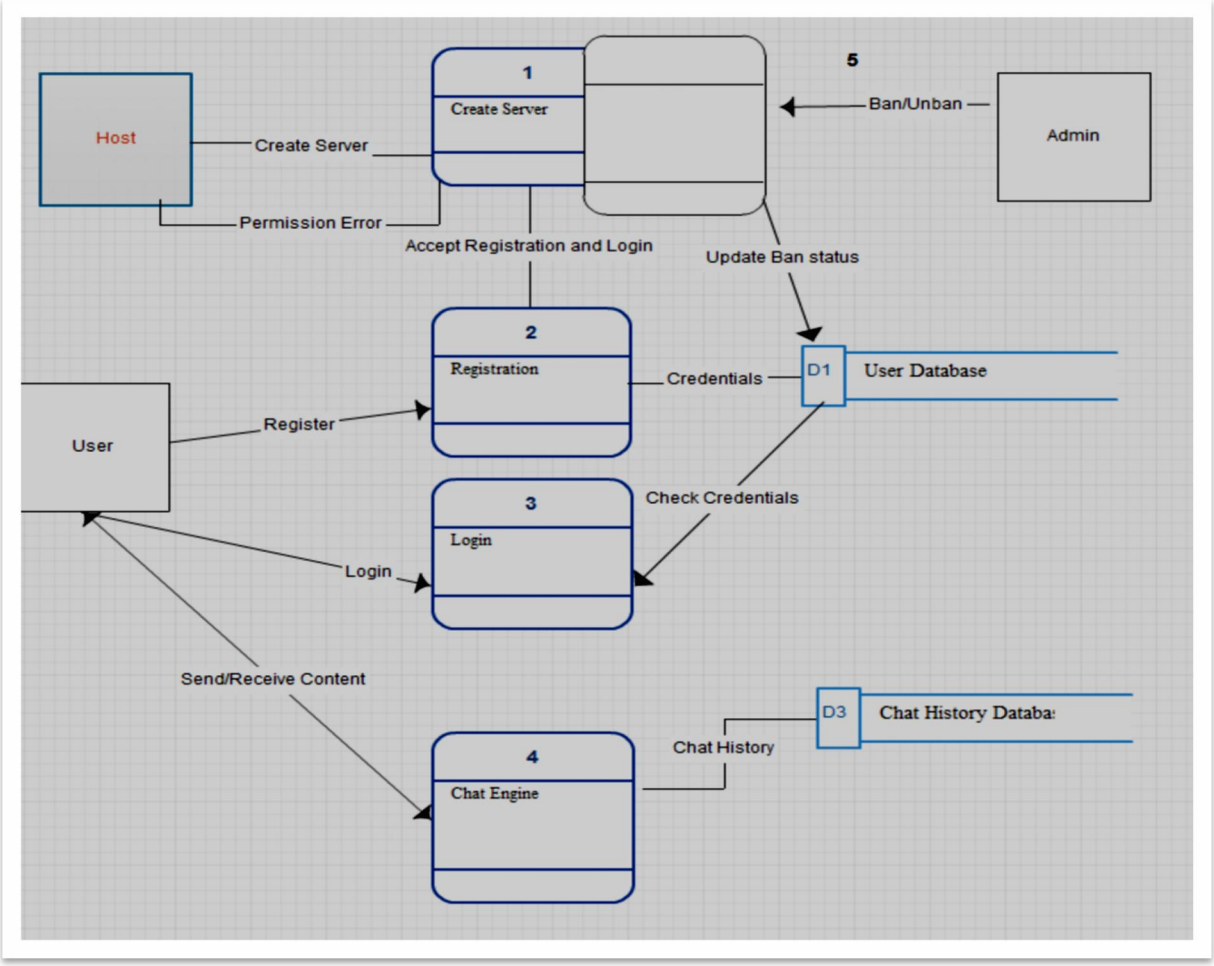


Fig 2.0 showing example of UI chat application. Source: Owner Created

Data Modeling

According to Simsion 2005 a data model is a design discipline that meets a set of business requirements.

It involves analysis of business requirements followed by design in response to those requirements.

Ponniah 2007 defines data modeling as an integral part of the process of designing and developing a data system.

Our data model will serve as critical tool to communicate to users as well as offering a blueprint of our application to be used in development phase.

Entities identified included the following:

- **Server**
- **Clients**
- **Message**

Attributes

Each entity list above has properties or characteristics. These characteristics are named to allow classification and identification of entities. The characteristics which are important are called attributes and include:

Entity	Properties
Server	IP, Location, Authentication Key
Client	IP, Location, Authentication Key
Chat	Chat Date, Chat _ID, Sender_ID, History
Users	User_ID, User Name, User Pass Key

Fig 2.1 Showing Entity attributes. Source: Owner Created

Entity Relationship Diagram

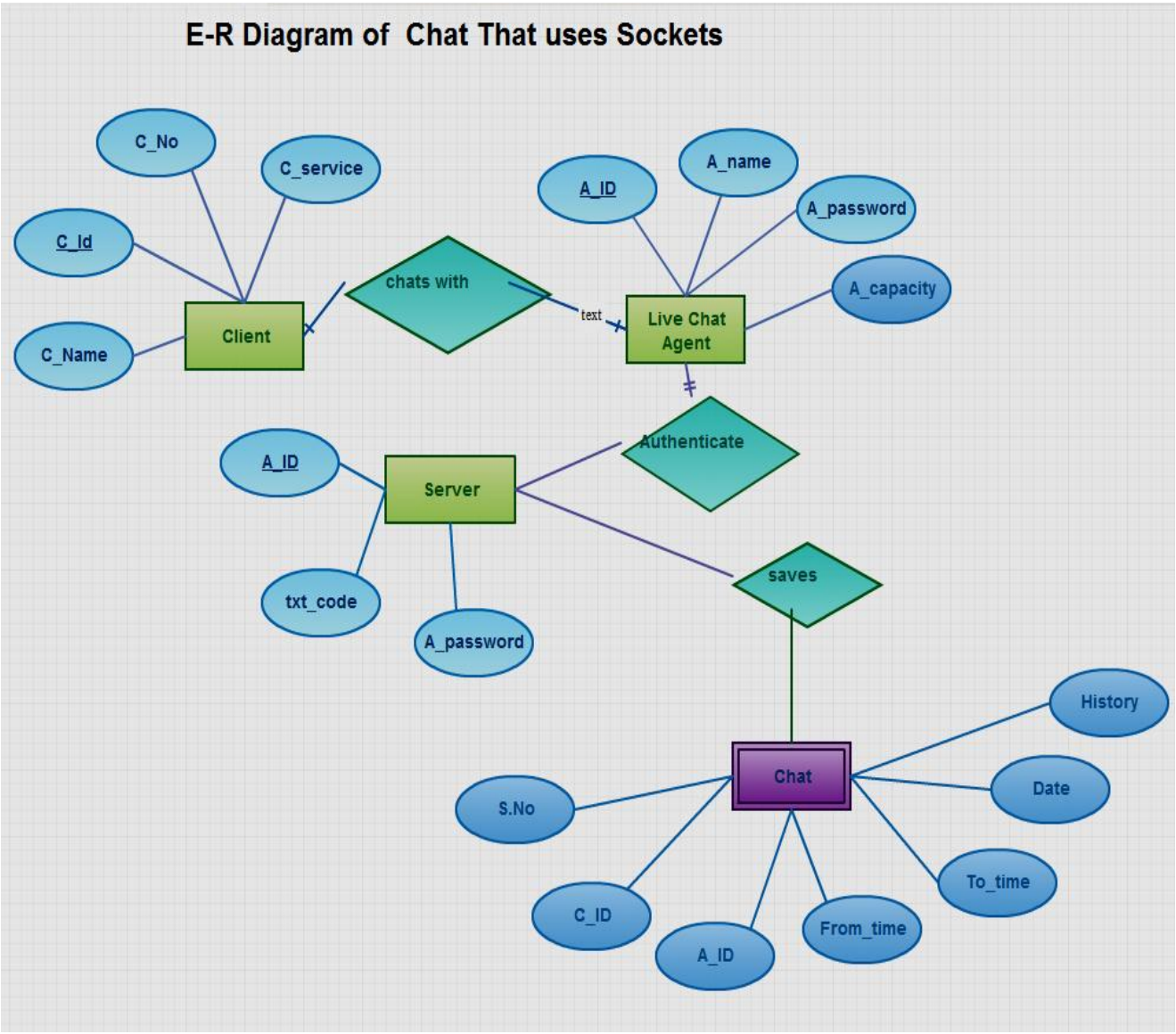


Fig 2.2 Showing Entity Relationship Diagram. Source: Owner Created

Use case 1

Client IPv4 use case

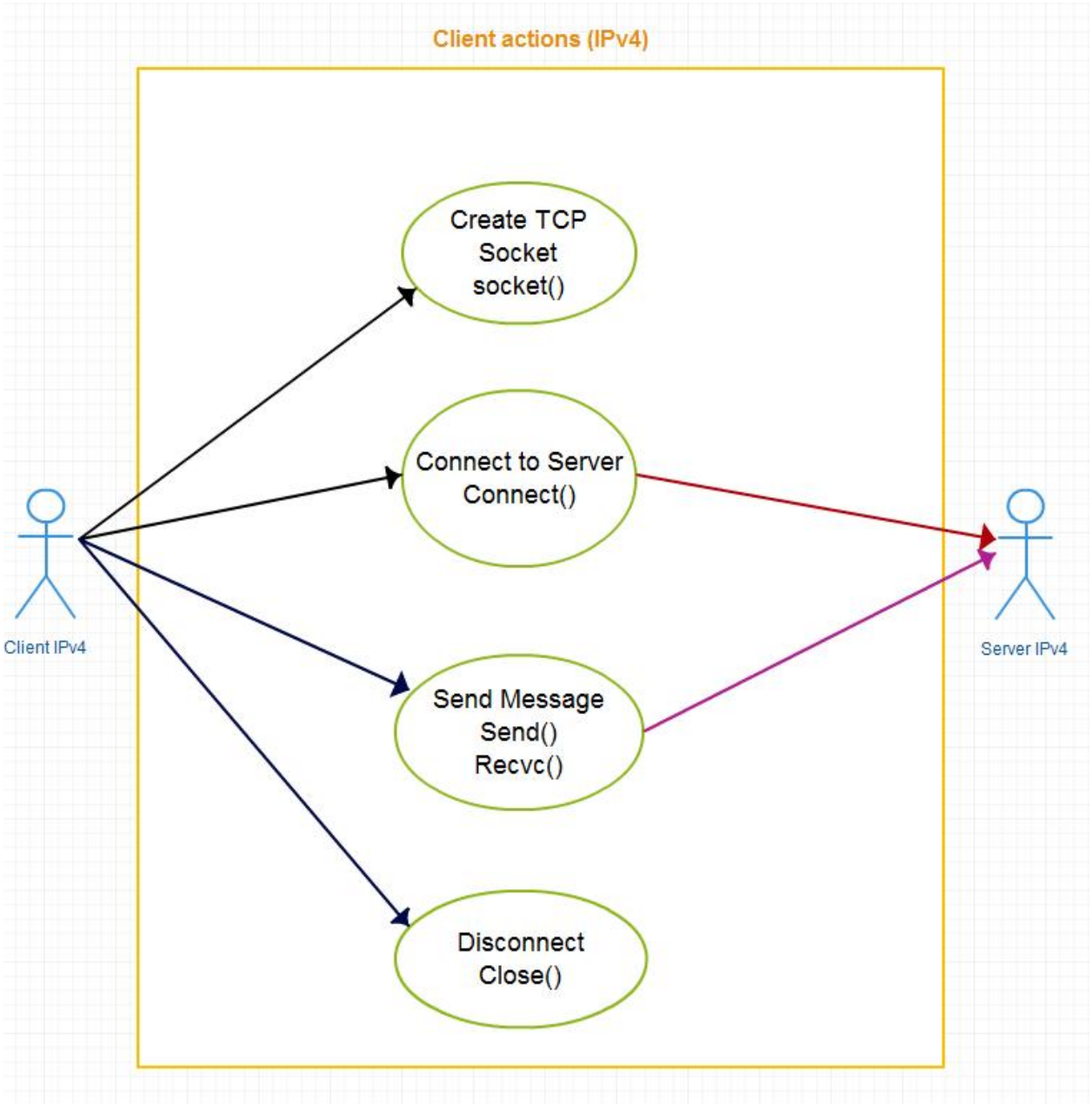


Fig 2.3 Showing client IPv4 use case. Source: Owner Created

Use case 2

Server IPv4 use case

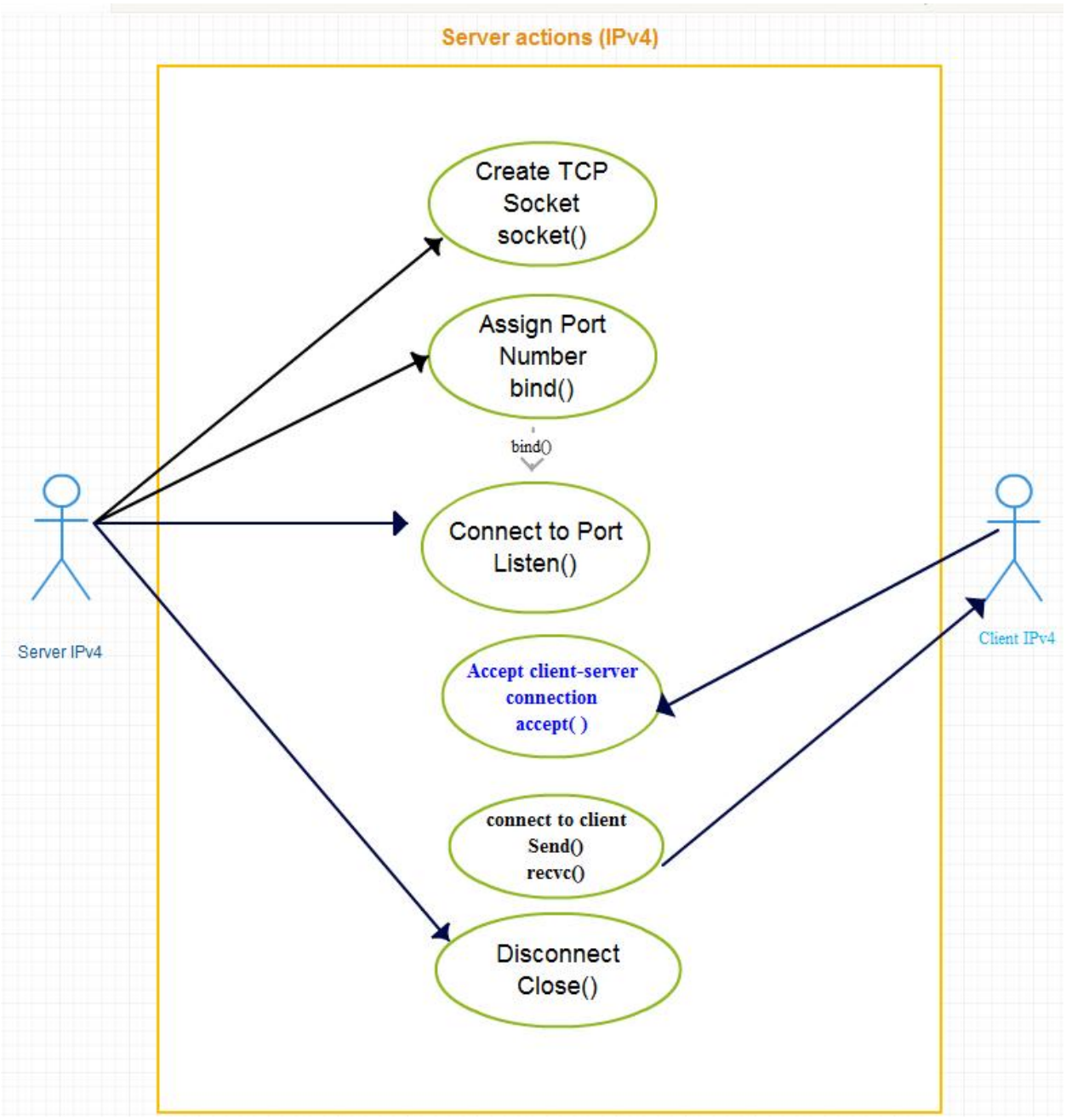


Fig 2.4 Showing server IPv4 use case. Source: Owner Created

Code design/Networking

Code design entails the inner abstracted procedures and algorithms that are queried by the external user friendly environment.

We will try and give a shadow of how the code will look like but much of it will be detailed in coding phase.

This is to be used later by the programmer to know what is needed though his choice of language and skill to apply entirely depends on him.

The illustrations try to explain how the internal enviroment will look like from the programmer point of view .

TCP and UDP socket overview

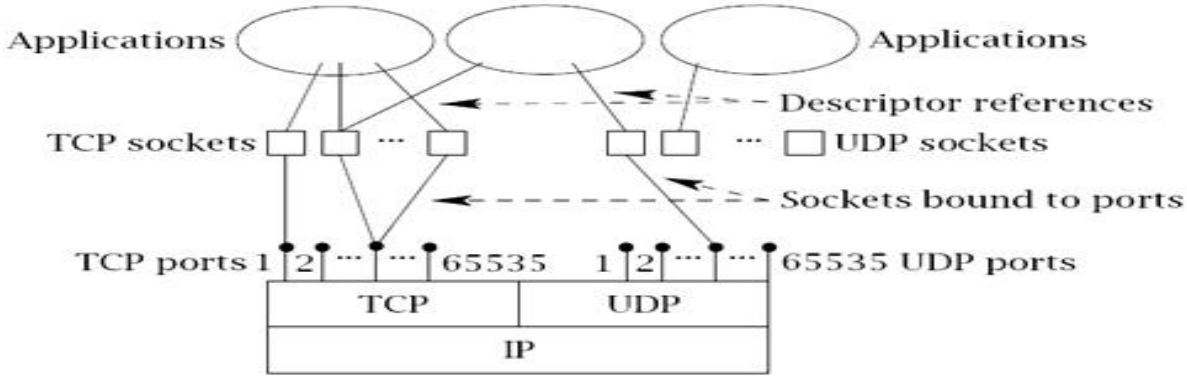


Fig 2.3 Showing TCP UDP code design. Source: Book by Calvert 2001

TCP/IP socket overview

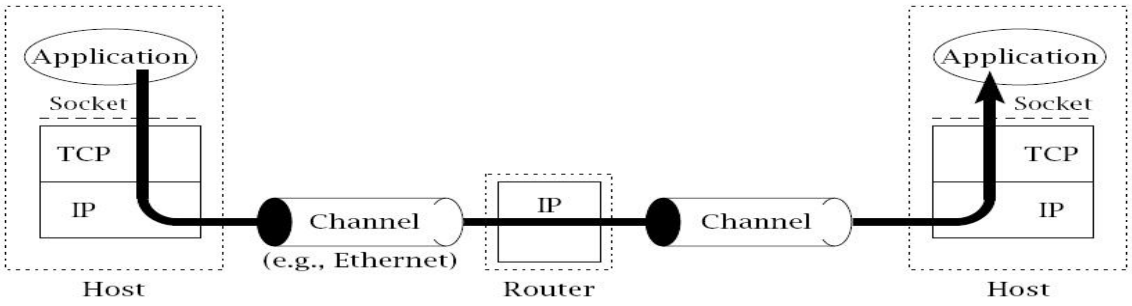


Fig 2.4 Showing TCP /IP code design. Source: Book by Calvert 2001

Process Design

Input Design/Output Design

The chat server app and chat client app ought to be turned on, and to send and receive messages, the following process occurs as illustrated in the following flow charts.

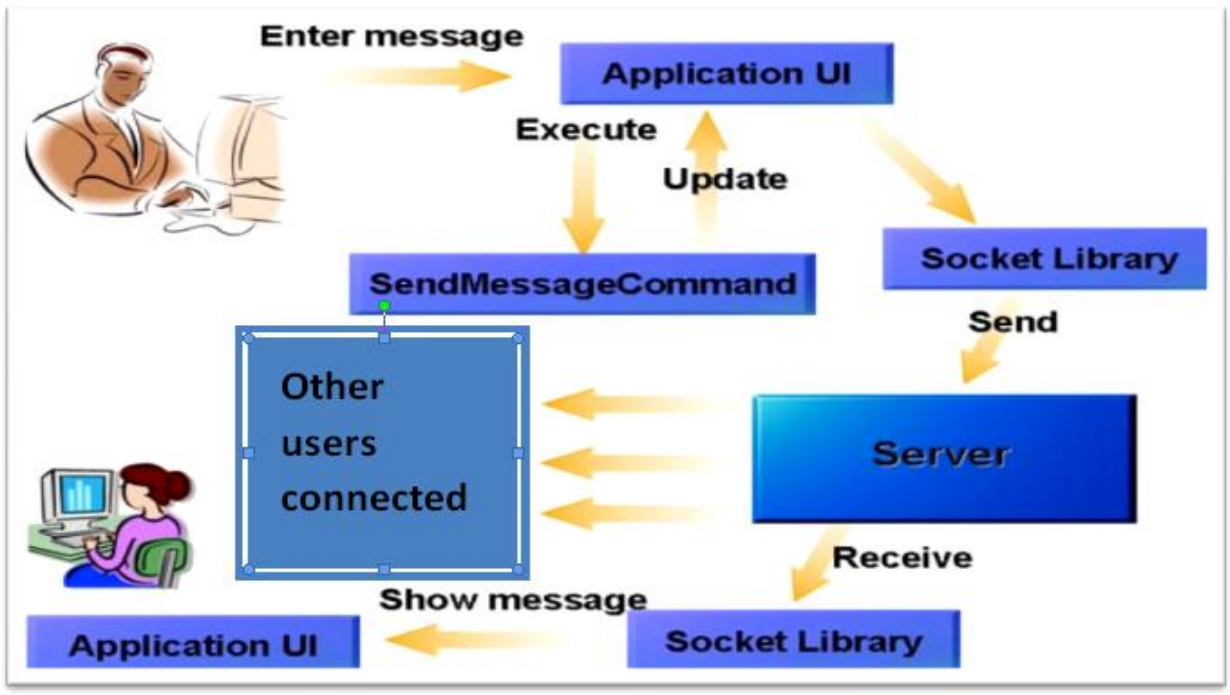


Fig 2.0 Client- Server Process Design, Source: Google Images (Edited)

Detailed Input/output design

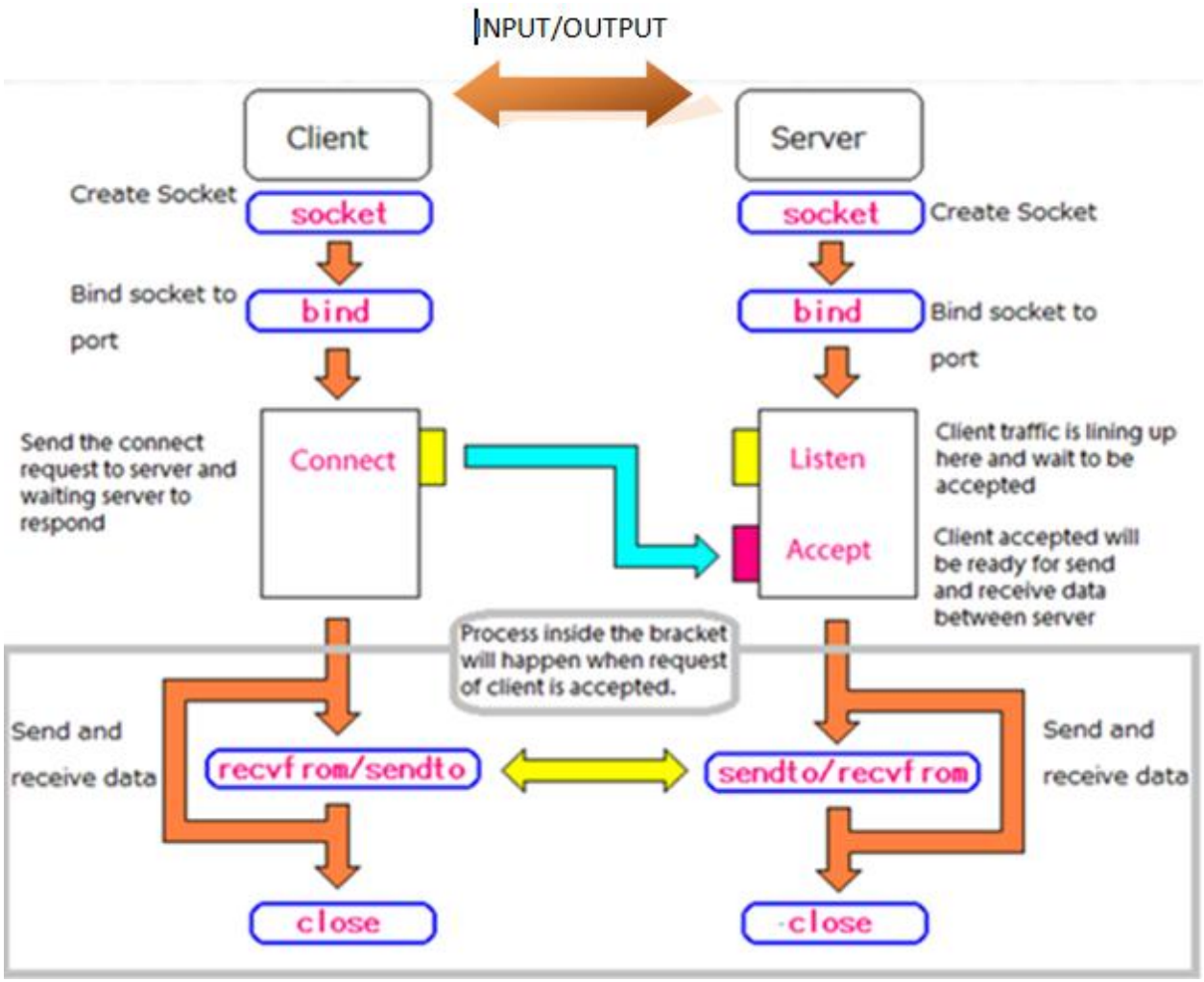


Fig 2.1 Client- Server Process Design, Source: Google Images (Edited)

SYTSTEM IMPLIMENTATION

According to Austin 2014 given the architecture document from the design phase and the requirement document from the analysis phase, the team should build exactly what has been requested, though there is still room for innovation and flexibility.

The implementation team therefore relies on this blueprint documents to bring out the system to life by turning it into code that works for both user (UI) and programmers (API).

The implementation team therefore set to achieve the following in our chat application:

1. Defining how the chat application should be built (i.e., physical system design)
2. Ensuring that the chat application is operational and used
3. Ensuring that the chat application meets quality standard (i.e., quality assurance).

The languages preferred were C#, C, C++, JAVA, and WEB Programming Languages (PHP, JavaScript).
The language of choice depended on team ability a level of skill to be applied (User needs)

We decided on C# Object Oriented Programming because of the user choice of language and the capability in which it supports low level programming for sockets.
The application ought to be free and friendly in as far as much as error removal is concerned.
Critical errors were removed during coding phase through compilation and running of the programs.
The agile nature of the application gave an opportunity to create this application with a flexible modular approach that saw the end result.
The final functionalities and advanced user interface was left for future development but to test the working of this application using sockets, a simple approach was adopted.

The blocking sockets method (wait till a message is received) was used but later development will involve non-blocking (Multithread threads).

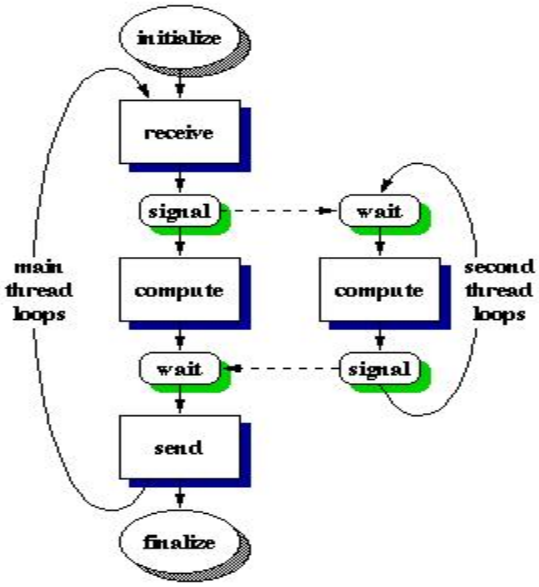


Fig 2.2 showing a multithreaded chat application. Source(Google images)

The implementation of this application was done with close scrutiny of user needs and future requirements.
The future implementation strategy of implementing this application therefore will fall into this sub phases of implementation:

- Planning for implementation
- deployment
- use of the app,
- maintaining the app
- transition to the newer version of app

Fig 2.3 showing Implementation phase . Source(Google images)

THE TESTING PHASE

In this phase we determine whether the system can undergo real-time execution and run all the required functionalities without breaking down.

Testing was crucial for all phases of this application development to see an application that is well coordinated and collaborated from the word go.

The system was forced to fail with extreme conditions or using non-required inputs.

Testing objectives:

- To determine whether the system can carry its required objectives without breakdown
- To determine errors, faults and malfunctioning of the system.
- To give a clear operation of the system towards attaining the user needs.
- To see if the intended new system meets both user and system requirements.

Scope of the test plan

The test will mainly be restricted to the following areas:

User validation

The server app needs to authenticate the client app by ensuring that the request to connect to it is authentic and valid.

The client ought to establish a connection to the server and if the server is down receive a connection error message reporting the server is down.

Data retrieval

The applications ought to retrieve previous data but for our case since no database approach was made the demonstrative application will only receive any user and no record of his/her chat session or history will be saved or retrieved.

Future expansions will make data retrieval of sessions saved as history of user possible.

Error reporting and recovery

The application (Server and Client) should respond with error reporting in case of failure or unnecessary occurrences and give recovery plan.

Testing types

This application was tested on various testing methods that is white box, black box, unit testing, validation testing and recovery testing.

Black-Box Testing

According to tutorials point is a technique of testing without having any knowledge of the interior workings of the application.

While performing a black-box test on this app, the tester interacted with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

White-Box Testing

According to tutorials point 2006 White-Box testing is the detailed investigation of internal logic and structure of the code.

The tester here did scrutinize the source code to find out which unit/chunk of the code was behaving inappropriately.

Levels of testing

According to tutorials point 2006 levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are:

- Functional Testing
- Non-functional Testing

Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested.

The application was tested by providing input and then the results were examined if they conformed to the functionality it was intended for.

To accomplish functionality testing the following 5 steps were applied

a. The determination of the functionality that the chat application is meant to perform.
b. The creation of test data based on the specifications of the chat application.
c. The output based on the test data and the specifications of the chat application.
d. The writing of test scenarios and the execution of test cases.
e. The comparison of actual and expected results based on the executed test cases

Functional Testing performed

Unit testing

Unit testing was performed on the individual units of source code assigned areas. This was evident during coding stage through running of code and debugging.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. (Tutorials point, 2006).

The various modules were combined and to make the whole application.

Acceptance Testing

The app was tested to gauge whether the application meets the intended specifications.

Test case

To document the testing phase a test case was our choice tool. Following is a test case generated from the testing of the system.

Test case 1

Case ID: 1

Product module: Connect to server

Purpose: Connecting to server from client

Pre-conditions: The server and client ought to be on

Steps:

1. Turn on server and client apps
2. Submit IP of server to the client (Use 127.0.0.1 for local host)
3. Read result
4. If connect send confirmation

Expected outcome:

The client receives connected message if yes and not connected message if connection error occurs.

Actual outcome: An error message was given after 2-3 attempts but changes to the code finally made the connection.

Post-conditions: The right IP of remote host has to be supplied.

Test case 2

Case ID: 2

Product module: send message

Purpose: Send message from client to server and reply from server to multiple clients

Pre-conditions: The server and client ought to be on

Steps:

1. Turn on server and client apps
2. Submit IP of server to the client (Use 127.0.0.1 for local host inbuilt catered for no need)
3. Read result
4. If connect send message from server

Expected outcome:

- The client receives connected message if yes and not connected message if connection error occurs.
- The client message needs to be seen at the server and the same message be seen in various clients connected.
- The message from the server goes to all clients connected at once.

Actual outcome:

Connection established.

Client message was seen in server and message from server was seen in client connected.

Post-conditions: The right IP of remote host has to be supplied (This was embedded into code).

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes like security, usability, performance etc.

Functional Testing performed

Performance Testing

It was done to identify any bottlenecks or performance issues rather than finding bugs in our application. Network delay and server side processing was found to be a major bottleneck to the performance of this application.

Load Testing

Tutorials point 2006 defines load testing as a process of testing the behavior of software by applying maximum load in terms of software accessing and manipulating large input data.

Our application server loads slowly when several clients are connected to it. Future enhancements like multithreading will improve this and lead to performance improvement.

Usability Testing

According to Tutorials point 2006 usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

Molich in 2000 stated that a user-friendly system should fulfill the following five goals, i.e., easy to Learn, easy to remember, efficient to use, satisfactory to use, and easy to understand.

Security Testing

Security testing involves testing software in order to identify any flaws and gaps from security and vulnerability point of view.

The security testing aimed to ensure the following were achieved

- Confidentiality
- Integrity
- Authentication
- Availability
- Authorization
- Non-repudiation

EVALUATION

To evaluate this software application the key areas to look are:

- Performance
- Security
- Usability
- Expected requirements met and not met
- Quality of app

Performance

Performance of the application made was rather quick if a few clients were connected. Performance degraded with the following.

- Addition of more clients
- Speed of processing of the server
- Internet connection speed (In cases when the server and client were hosted in different machines)

Security

The security of this application was not yet fully implemented. Socket security was enhanced through an inbuilt mechanism, that is establishing a connection and connect if not issue error message.

User security was left for future enhancements since this was application was meant only to demonstrate use of sockets in a simple chat.

Usability

The app is easy to install and use and requires no technical skills to operate.

However, use of advanced graphical user interface (GUI) features like forms, menus and navigation elements was left for future enhancements.

Have a look at the manual at the appendix for more.

Expected requirements met and not met

The application did meet the basic requirements though lofty requirements remained unattended for future improvisation.

Some requirements met are:

- Basic socket implementation
- Ability to connect to server by client and vice versa
- Ability to send message from client to server and vise versa
- Ability to error report

Some requirements not met are:

Due to the complexity of coding requirements and expected delivery time some requirements were not met.

- Ability to create multithreaded server application (Non-Blocking)
- Ability to create a nice GUI interface
- Ability to create a secure application
- Ability to create an application that handles simultaneous communication(ours can only send and receive once)

Quality of app

The application quality measured from the algorithms, style of coding language preference, expected user requirements met, can be termed as average.

Future recommendations and enhancements will make this work to be of high quality.

CONCLUSIONS

The phases taken to build this simple application implementing sockets were a stepping to learn so much about sockets.

Networking principles were covered that is TCP /IPv4 sockets on which this project relied so much.

An introduction to blocking sockets and non-blocking sockets was a step also to learn more about sockets and future advancements of this application will require prior knowledge of these terms.

Socket programming in C# seemed hard at first but later advancements will involve use of other languages to bring out the GUI and other elements which I could not achieve in C# due to little knowledge on that language.

Data structures and functions encountered were**, socket (), bind (), connect (), listen (), accept** and they formed a bigger role in making this application.

Socket programming is mainly implemented in distributed systems to allow programs to communicate with each other.

RECOMMENDATIONS AND FUTURE WORK

The application ought to have achieved better reward in level of coding skills, level of skill applied to develop the graphical user interface, level of security and quality of functionalities provided.

Since time and skills became major factors to consider ,most of the requirements were not met although the basic requirements were met.

The recommendation I give for future development of this application is this:

- A highly effective, responsive Graphical user interface (GUI)
- Multithreaded server application (Non-Blocking server)
- Close working with user to know all user needs (User oriented programming)
- Better research in other languages like python, php, C++ and UNIX sockets to see which makes it better
- Team development of this application from scratch.
- Development of Advanced Sockets application
- TCP IPv6 socket implementation
- Implement a dedicated server where many clients can connect to.

References

- Oracle Tutorial, J. (2015, January 2). What Is a Socket? Retrieved April 9, 2015, from https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html
- Mitchell, B. (2014, January 6). Retrieved April 20, 2015, from http://compnetworking.about.com/od/networkprotocols/f/port-numbers.htm By Bradley Mitchell
- Grinter, R., & Palen, L. (2002, March 7). Instant Messaging in Teen Life. Retrieved April 21, 2015, from https://www.cs.colorado.edu/~palen/Papers/grinter-palen-IM.pdf
- Farah, W. (2000, February 2). DEVELOPING A CHAT SERVER. Retrieved April 24, 2015, from http://www.ndu.edu.lb/academics/senior_projects/chatserver.htm
- Hardikar, N. (2007). *A case study of the use of instant messaging technology in the workplace*.
- Calvert, K.,& Donahoo, M. (2002). *TCP/IP sockets in Java practical guide for programmers*. San Francisco: Morgan Kaufmann.
- Rhodes, J. (2001). *Videoconferencing for the real world: Implementing effective visual communications systems*. Boston: Focal Press.
- Austin ,O., *The Project Report Guidelines .Moi University.*
- Rosen ,B *(2014) Systems Analysis and Design, (10th ed.).* Retrieved April 23, 2015 ,from *http://books.google.co.ke/books/about/Systems_Analysis_and_Design.html?id=h_PUASRnDDYC&redir_esc=yLink*
- Neustaedter, C. (n.d.). Video Chat and Human-Computer Interaction | SciTech Connect. Retrieved April 23, 2015, from http://scitechconnect.elsevier.com/video-chat-hci/

- Ponniah, P. (2007). *Data modeling fundamentals a practical guide for IT professionals*. Hoboken, N.J.: Wiley-Interscience.
- Simsion, G., & Witt, G. (2005). *Data modeling essentials* (3rd ed.). Amsterdam: Morgan Kaufmann.
- Donahoo, Michael J., Kenneth L. Calvert, and Michael J. Donahoo (2001). *The pocket guide to TCP/IP sockets* (C version. ed.). San Francisco: Morgan Kaufmann.
- Shelly, G., & Cashman, T. (2001). *Systems analysis and design* (4th ed.). Boston: Course Technology
- Agile Methodology. (2014, April 4). Retrieved April 25, 2015, from http://agilemethodology.org/
- Mitchell, M., Verma, G., & Mohammed, O. (2006). *Software Testing - Overview*. Retrieved April 25, 2015, from http://www.tutorialspoint.com/software_testing/software_testing_overview.htm

- "How to C# Socket Programming." *How to C# Socket Programming*. N.p., n.d. Web. 07 May 2015.
- "Communication through Sockets." *Windows Sample in C# for Visual Studio 2012*. N.p., n.d. Web. 07 May 2015.
- "Communication through Sockets." Windows Sample in C# for Visual Studio 2012. N.p., n.d. Web.07 May 2015.

Sources with no references

- *Google search Engine*
- *Reports by other students*
- *Google books*
- *Google images*
- *Digital images*
- *Blogs*

APPENDIX1

Installation guide

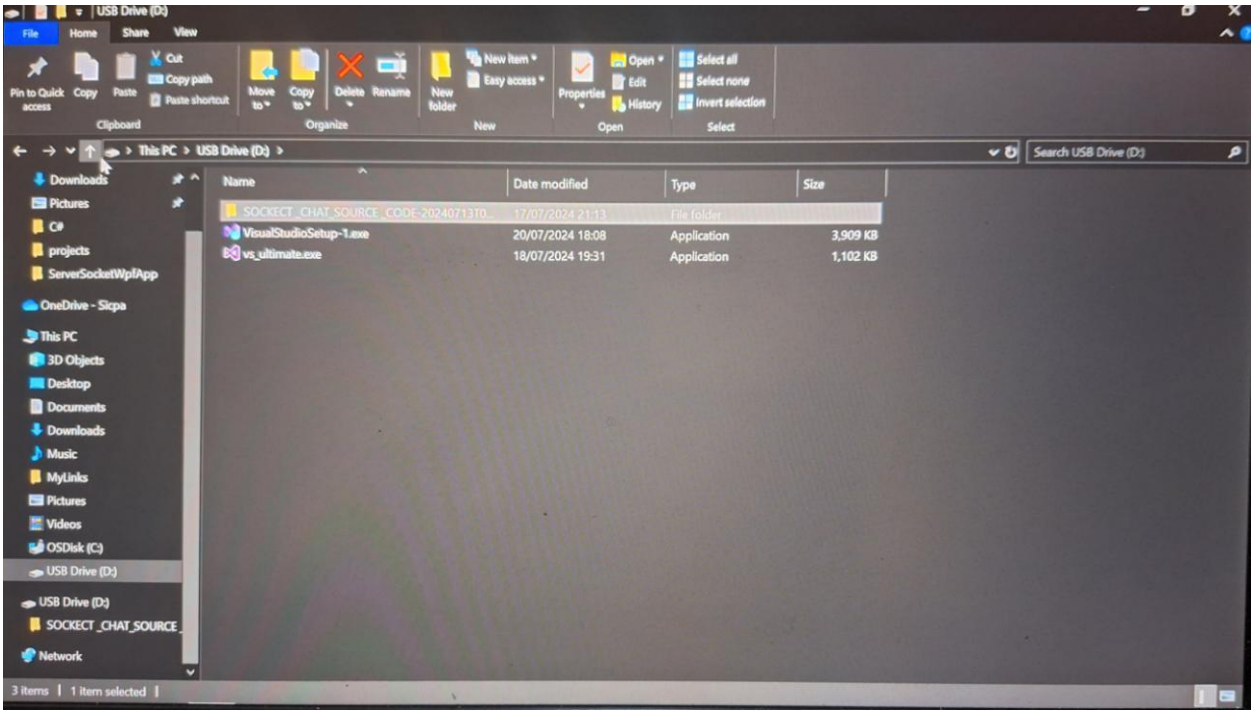
PREREQUISITE

NB: UPON PRE-REQUISITE EXPLAINED IN THE REQUIREMENT ANALYSIS WE STILL URGE YOU TO HAVE MICROSOFT VISUAL 2013 OR ABOVE INSTALLED ON YOUR SYSTEM TO RUN AND COMPILE THIS APPLICATION.

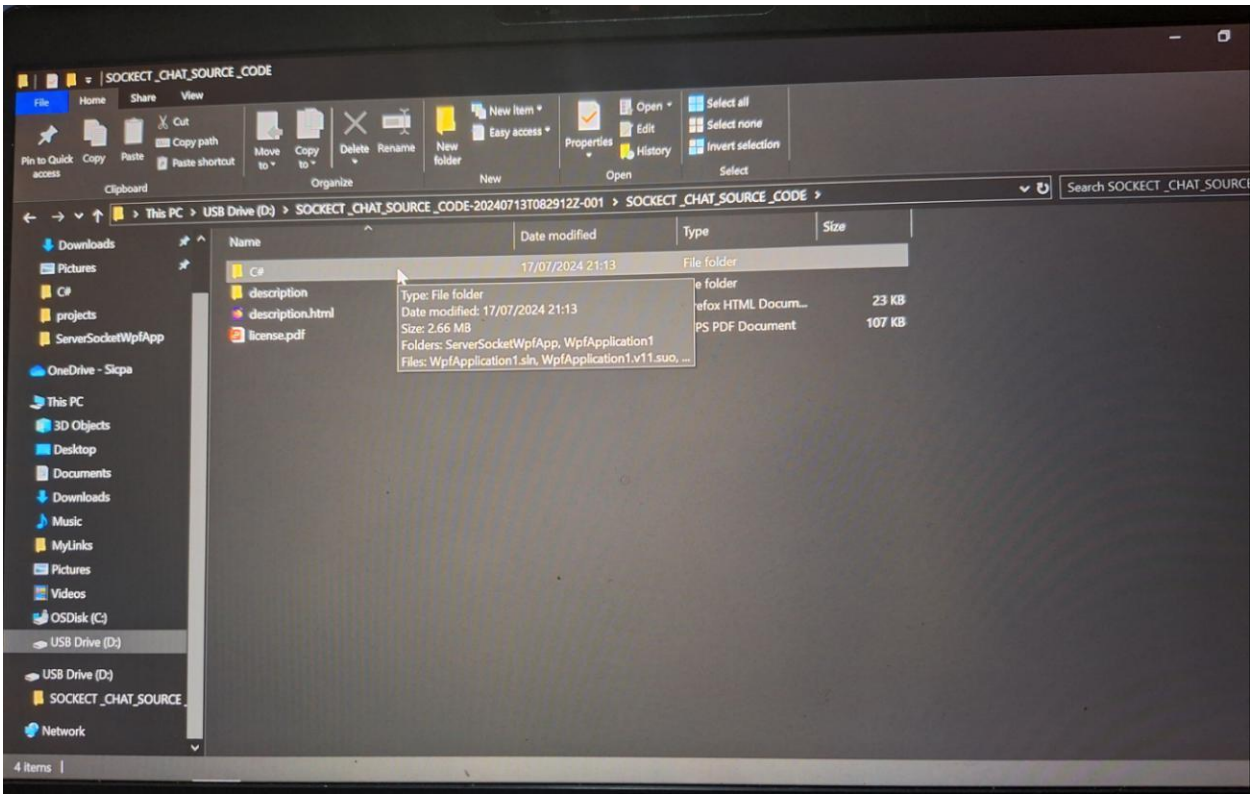
STEPS TO INSTALL AND RUN APP

To view source code and run the app ensure you have Microsoft visual 2013 installed or any program that can run such files.

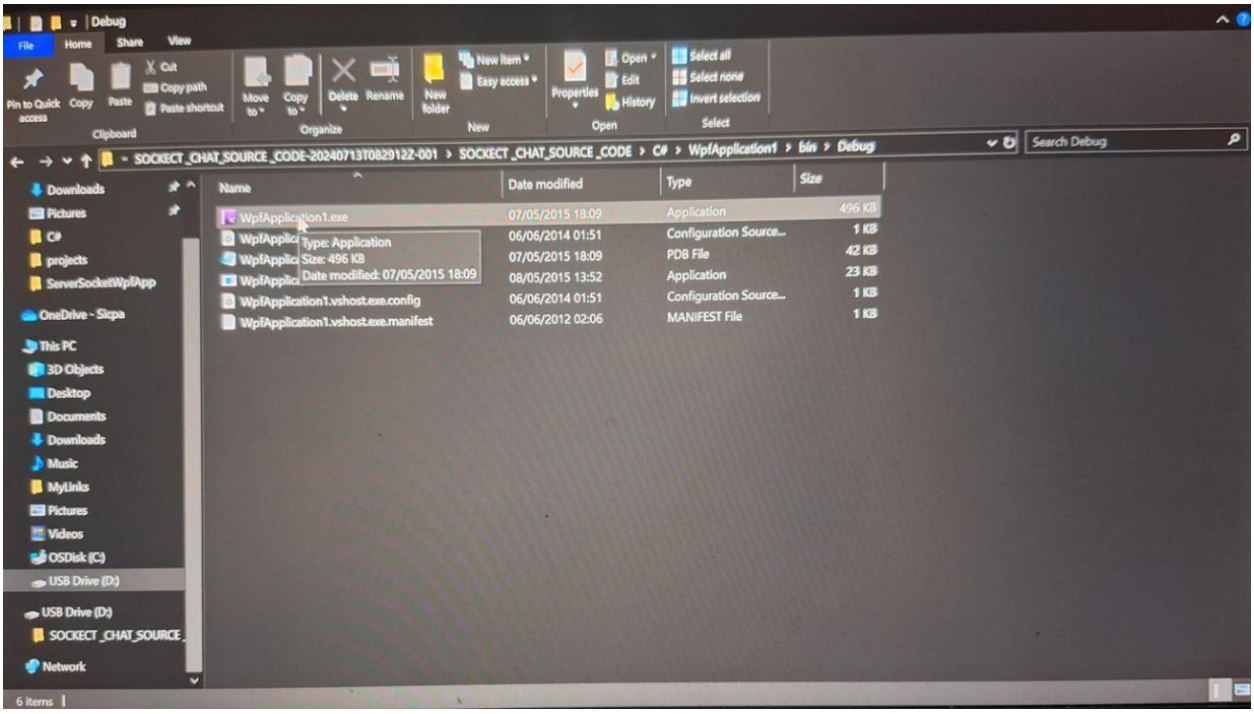
1. **Insert CD containing the chat application in to your machine OR**
2. **Open the ticked folder SOCKET_CHAT_SOURCE_CODE from CD**



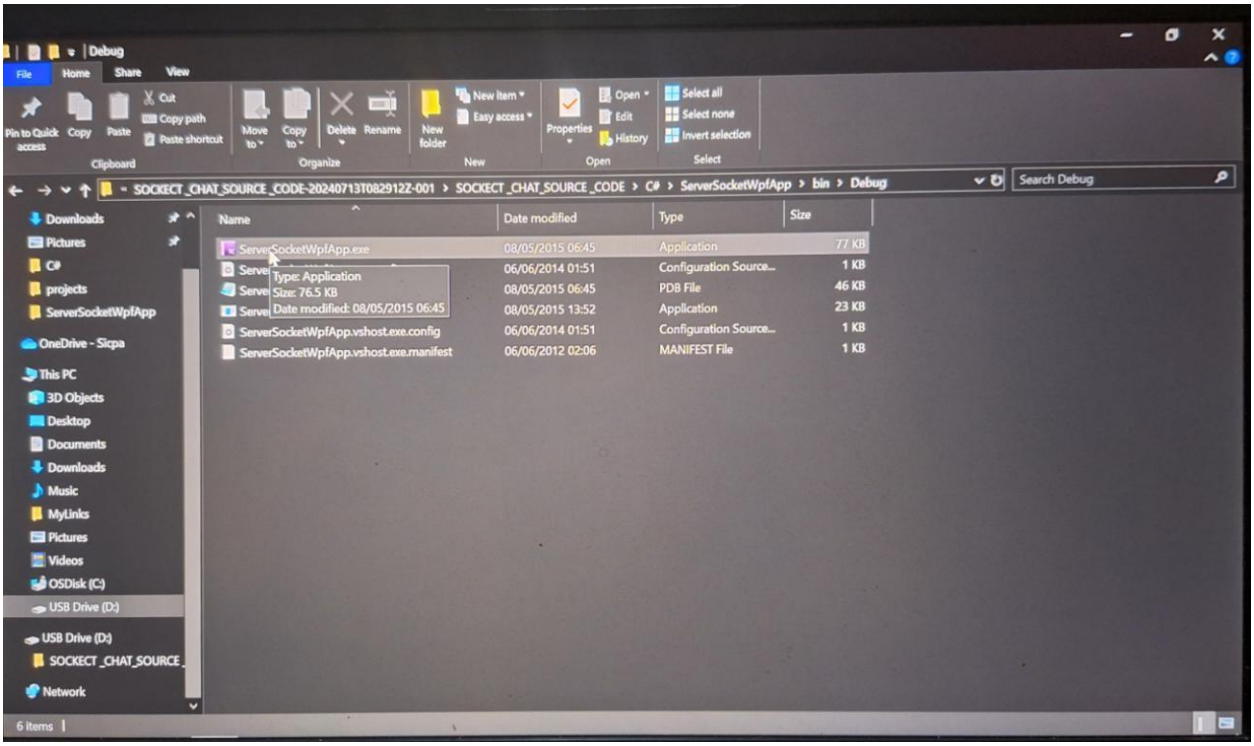
3. **Open folder ticked C#**



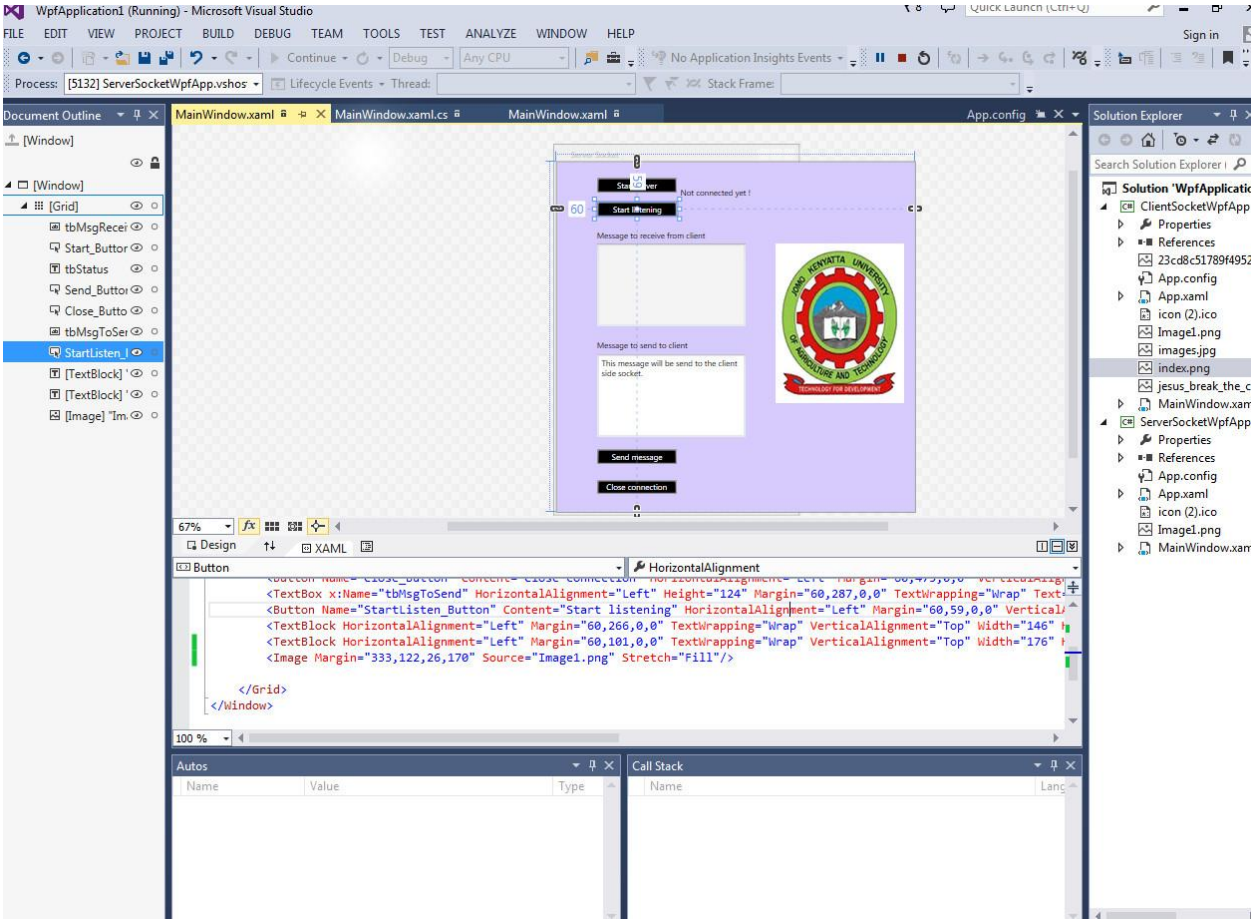
4. **Open the file ticked WpfApplication1 to open client application**



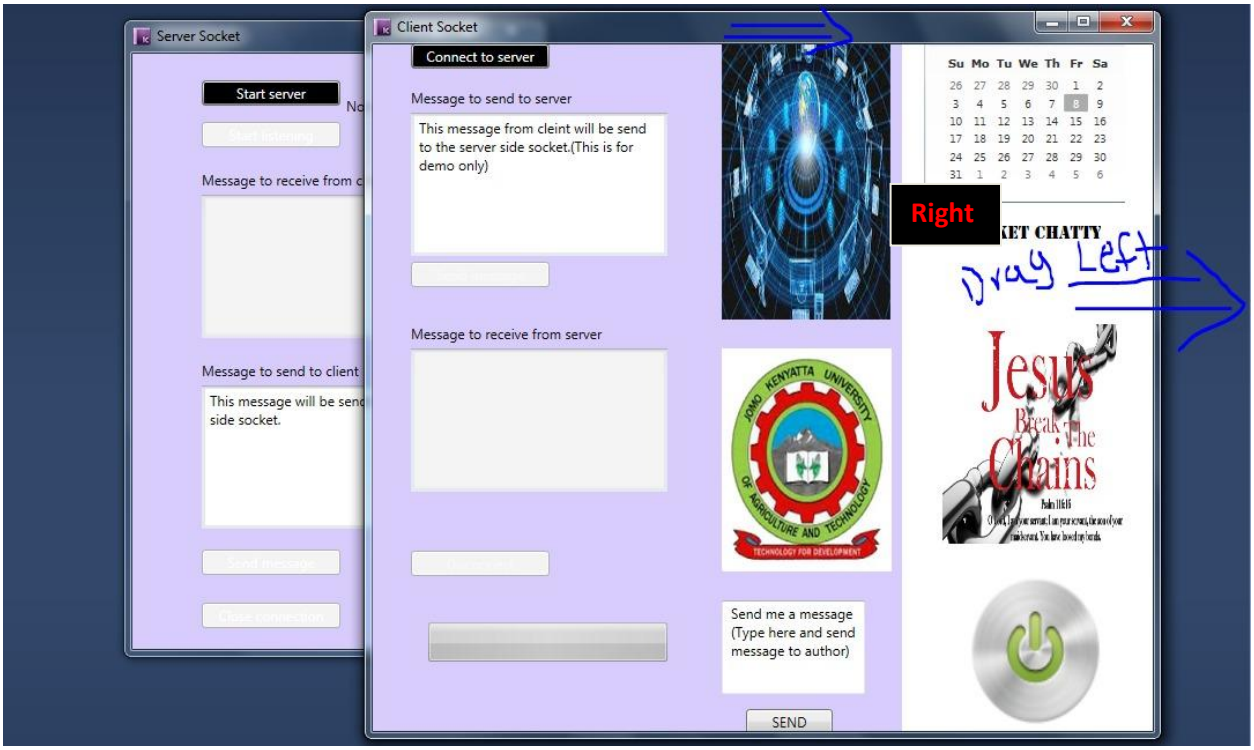
5. **Click on the server to open the server application.**



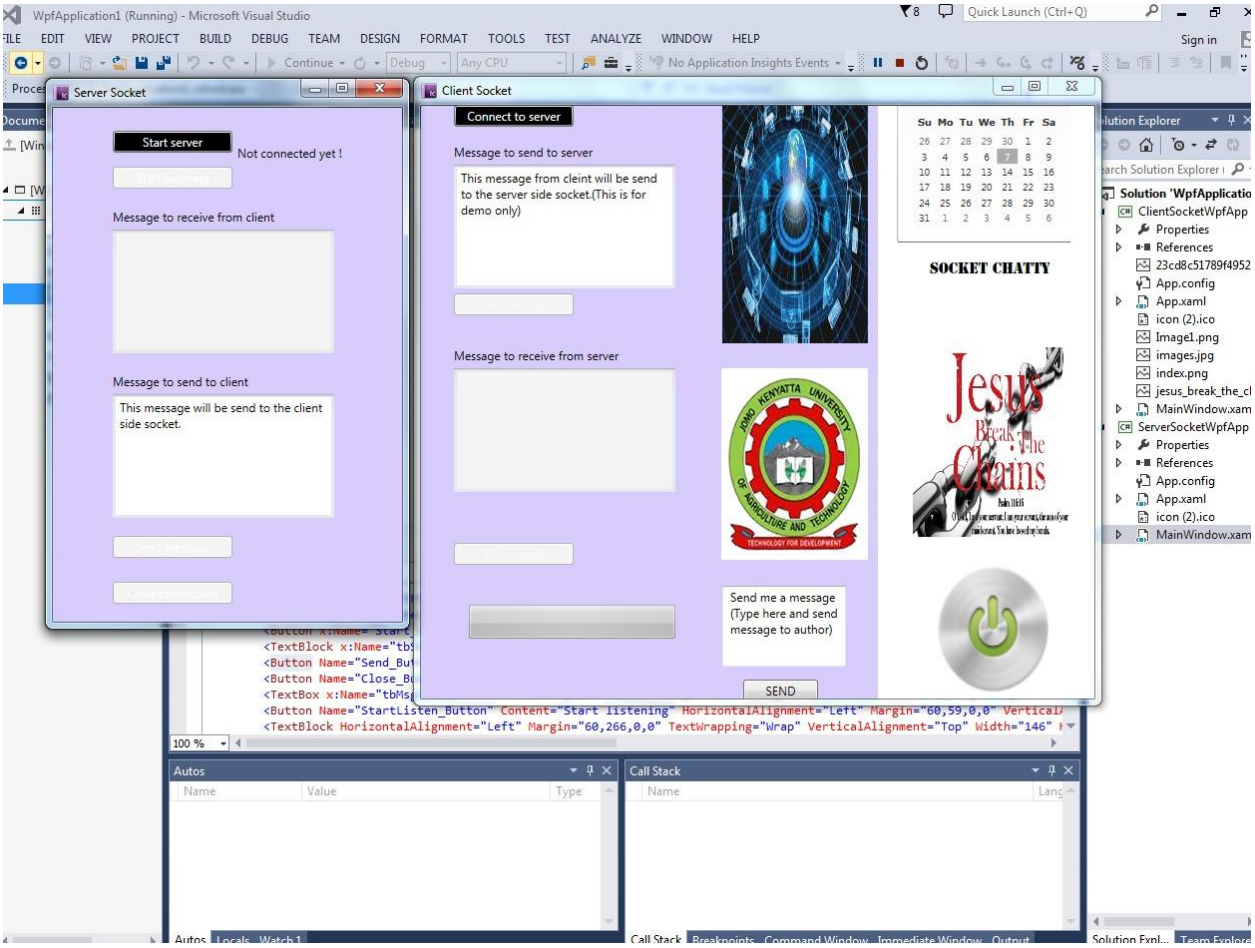
6. **A server and client window will appear as below (Server window is hidden under client window)**



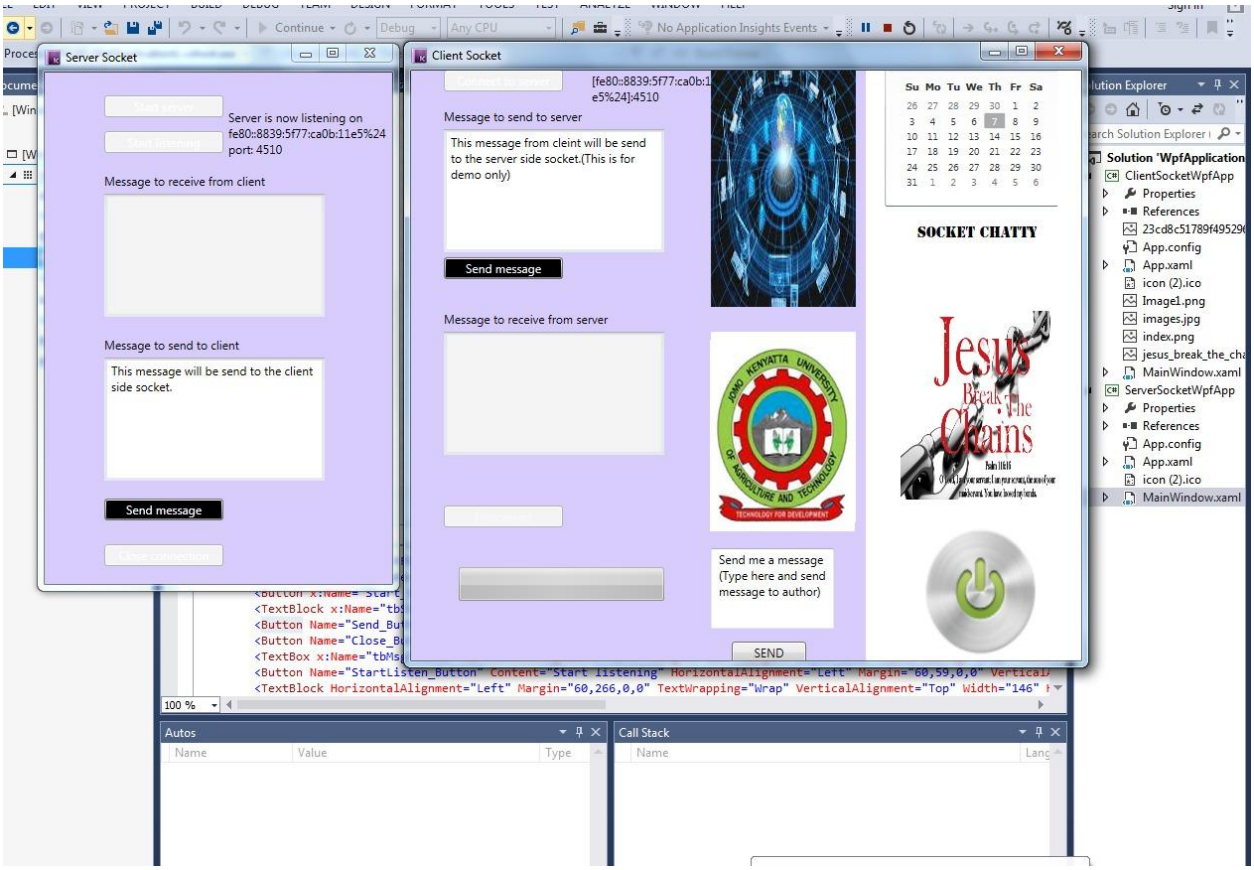
7. **Drag right the client window to view the server window if the client window overshadows the server window as mine, if not don't worry.**



8. **Now turn server online first and then connect client then send message from client to server and from server to client or simply apply as per steps given 1 to 7 to run and close the app.**


WpfApplication1 (Running) - Microsoft Visual Studio
Server Socket
Start server
Not connected yet !
Message to receive from client
Message to send to client
This message will be send to the client side socket.
Client Socket
Connect to server
Message to send to server
This message from cleint will be send to the server side socket.(This is for demo only)
Message to receive from server
Send me a message (Type here and send message to author)
SEND
SOCKET CHATTY
Jesus Break The Chains
JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY
TECHNOLOGY FOR DEVELOPMENT
Autos
Call Stack

9. **Output from the previous step**
10. **Done**



APPENDIX 2

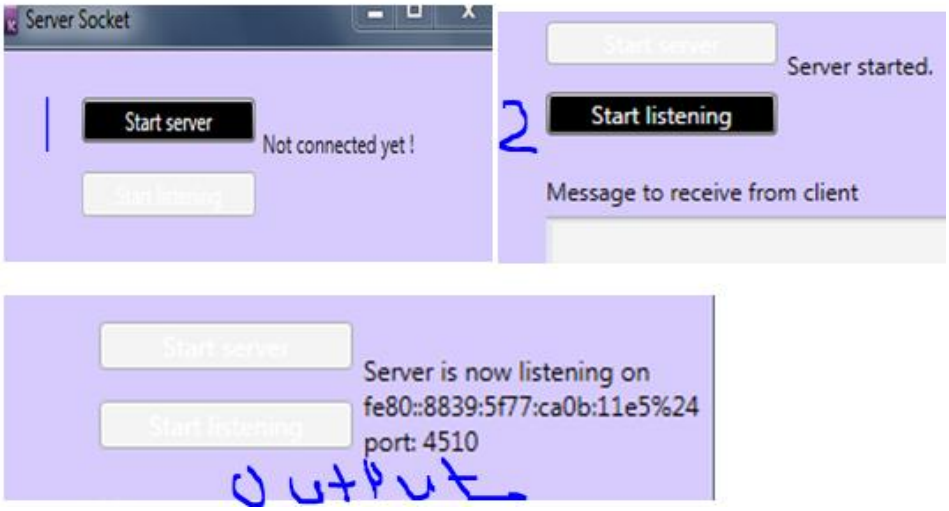
User manual

Starting application

View the installation and run guide

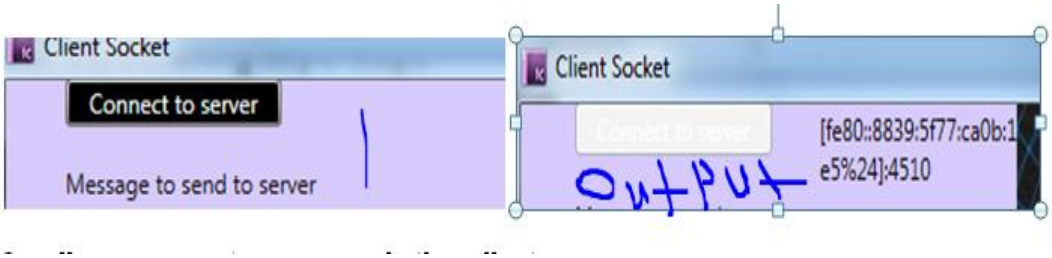
Turning on server application

Click on start server then click on start listening buttons as per steps



Turning on client application

Click on connect to server button



Sending message to server and other clients

Type message from the client chat window and click send message button

Sending message to all Clients

Type message from the server chat window and click send message button

Turning off Application

Just click on close connection and disconnect or close chat windows.

Error Reporting and Troubleshooting

- Ensure all the applications are turned on when sending message.
- Ensure network connection is available when the server and client run on different machines.
- Incase all this fails send error report email to kevnal456@gmail.com

CODES SNIPPETS

CLEINT CLASS CODE

```
// ORIGINALLY AUTHORED BY Houssem Dellai
//EDITED BY kEVIN NALIANYA


using System;
using System.Text;
using System.Windows;
using System.Net.Sockets;
using System.Net;

namespace WpfApplication1
{
   public partial class MainWindow : Window
   {

      // Receiving byte array
      byte[] bytes = new byte[1024];
      Socket senderSock;

      public MainWindow()
      {
         InitializeComponent();

         Send_Button.IsEnabled = false;
         Disconnect_Button.IsEnabled = false;
      }

      private void Connect_Click(object sender, RoutedEventArgs e)
      {
         try
         {
            // Create one SocketPermission for socket access restrictions
            SocketPermission permission = new SocketPermission(
               NetworkAccess.Connect,    // Connection permission
```

```
                TransportType.Tcp,        // Defines transport types
                "",                       // Gets the IP addresses
                SocketPermission.AllPorts // All ports
                );

            // Ensures the code to have permission to access a Socket
            permission.Demand();

            // Resolves a host name to an IPHostEntry instance
            IPHostEntry ipHost = Dns.GetHostEntry("");

            // Gets first IP address associated with a localhost
            IPAddress ipAddr = ipHost.AddressList[0];

            // Creates a network endpoint
            IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 4510);

            // Create one Socket object to setup Tcp connection
            senderSock = new Socket(
                ipAddr.AddressFamily,// Specifies the addressing scheme
                SocketType.Stream,   // The type of socket
                ProtocolType.Tcp     // Specifies the protocols
                );

            senderSock.NoDelay = false;   // Using the Nagle algorithm

            // Establishes a connection to a remote host
            senderSock.Connect(ipEndPoint);
            tbStatus.Text = "Socket connected to " +
senderSock.RemoteEndPoint.ToString();

            Connect_Button.IsEnabled = false;
            Send_Button.IsEnabled = true;
        }
        catch (Exception exc) { MessageBox.Show(exc.ToString()); }

    }

    private void Send_Click(object sender, RoutedEventArgs e)
```

```
        {
            try
            {
                // Sending message
                //<Client Quit> is the sign for end of data
                string theMessageToSend = tbMsg.Text;
                byte[] msg = Encoding.Unicode.GetBytes(theMessageToSend +
"<Client Quit>");

                // Sends data to a connected Socket.
                int bytesSend = senderSock.Send(msg);

                ReceiveDataFromServer();

                Send_Button.IsEnabled = false;
                Disconnect_Button.IsEnabled = true;
            }
            catch (Exception exc) { MessageBox.Show(exc.ToString()); }
        }

        private void ReceiveDataFromServer()
        {
            try
            {
                // Receives data from a bound Socket.
                int bytesRec = senderSock.Receive(bytes);

                // Converts byte array to string
                String theMessageToReceive = Encoding.Unicode.GetString(bytes, 0,
bytesRec);

                // Continues to read the data till data isn't available
                while (senderSock.Available > 0)
                {
                    bytesRec = senderSock.Receive(bytes);
                    theMessageToReceive += Encoding.Unicode.GetString(bytes, 0,
bytesRec);
                }
```

```
            tbReceivedMsg.Text = "The server reply: " + theMessageToReceive;
        }
        catch (Exception exc) { MessageBox.Show(exc.ToString()); }
    }

    private void Disconnect_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            // Disables sends and receives on a Socket.
            senderSock.Shutdown(SocketShutdown.Both);

            //Closes the Socket connection and releases all resources
            senderSock.Close();

            Disconnect_Button.IsEnabled = false;
        }
        catch (Exception exc) { MessageBox.Show(exc.ToString()); }
    }

    private void tbReceivedMsg_TextChanged(object sender,
System.Windows.Controls.TextChangedEventArgs e)
    {

    }

    private void ProgressBar_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
    {

    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {

    }

    private void RichTextBox_TextChanged(object sender,
System.Windows.Controls.TextChangedEventArgs e)
```

```
        {

        }

        private void ListBox_SelectionChanged(object sender,
System.Windows.Controls.SelectionChangedEventArgs e)
        {

        }
    }
}
```

SERVER CLASS CODE

```
// ORIGINALLY AUTHORED BY Houssem Dellai
//EDITED BY KEVIN NALIANYA


using System;
using System.Text;
using System.Windows;
using System.Net.Sockets;
using System.Net;

namespace WpfApplication1
{
    public partial class MainWindow : Window
    {

        // Receiving byte array
        byte[] bytes = new byte[1024];
        Socket senderSock;

        public MainWindow()
        {
            InitializeComponent();

            Send_Button.IsEnabled = false;
```

```
    Disconnect_Button.IsEnabled = false;
}

private void Connect_Click(object sender, RoutedEventArgs e)
{
    try
    {
        // Create one SocketPermission for socket access restrictions
        SocketPermission permission = new SocketPermission(
            NetworkAccess.Connect,    // Connection permission
            TransportType.Tcp,        // Defines transport types
            "",                       // Gets the IP addresses
            SocketPermission.AllPorts // All ports
            );

        // Ensures the code to have permission to access a Socket
        permission.Demand();

        // Resolves a host name to an IPHostEntry instance
        IPHostEntry ipHost = Dns.GetHostEntry("");

        // Gets first IP address associated with a localhost
        IPAddress ipAddr = ipHost.AddressList[0];

        // Creates a network endpoint
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 4510);

        // Create one Socket object to setup Tcp connection
        senderSock = new Socket(
            ipAddr.AddressFamily,// Specifies the addressing scheme
            SocketType.Stream,   // The type of socket
            ProtocolType.Tcp     // Specifies the protocols
            );

        senderSock.NoDelay = false;   // Using the Nagle algorithm

        // Establishes a connection to a remote host
        senderSock.Connect(ipEndPoint);
```

```
            tbStatus.Text = "Socket connected to " +
senderSock.RemoteEndPoint.ToString();

            Connect_Button.IsEnabled = false;
            Send_Button.IsEnabled = true;
        }
        catch (Exception exc) { MessageBox.Show(exc.ToString()); }

    }

    private void Send_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            // Sending message
            //<Client Quit> is the sign for end of data
            string theMessageToSend = tbMsg.Text;
            byte[] msg = Encoding.Unicode.GetBytes(theMessageToSend +
"<Client Quit>");

            // Sends data to a connected Socket.
            int bytesSend = senderSock.Send(msg);

            ReceiveDataFromServer();

            Send_Button.IsEnabled = false;
            Disconnect_Button.IsEnabled = true;
        }
        catch (Exception exc) { MessageBox.Show(exc.ToString()); }
    }

    private void ReceiveDataFromServer()
    {
        try
        {
            // Receives data from a bound Socket.
            int bytesRec = senderSock.Receive(bytes);

            // Converts byte array to string
```

```
            String theMessageToReceive = Encoding.Unicode.GetString(bytes, 0,
bytesRec);

            // Continues to read the data till data isn't available
            while (senderSock.Available > 0)
            {
               bytesRec = senderSock.Receive(bytes);
               theMessageToReceive += Encoding.Unicode.GetString(bytes, 0,
bytesRec);
            }

            tbReceivedMsg.Text = "The server reply: " + theMessageToReceive;
         }
         catch (Exception exc) { MessageBox.Show(exc.ToString()); }
      }

      private void Disconnect_Click(object sender, RoutedEventArgs e)
      {
         try
         {
            // Disables sends and receives on a Socket.
            senderSock.Shutdown(SocketShutdown.Both);

            //Closes the Socket connection and releases all resources
            senderSock.Close();

            Disconnect_Button.IsEnabled = false;
         }
         catch (Exception exc) { MessageBox.Show(exc.ToString()); }
      }

      private void tbReceivedMsg_TextChanged(object sender,
System.Windows.Controls.TextChangedEventArgs e)
      {

      }

      private void ProgressBar_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
```

```
        {

        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {

        }

        private void RichTextBox_TextChanged(object sender,
System.Windows.Controls.TextChangedEventArgs e)
        {

        }

        private void ListBox_SelectionChanged(object sender,
System.Windows.Controls.SelectionChangedEventArgs e)
        {

        }
    }
}
```

Index: N.A