# 

[**1. INTRODUCTION** 2](#_Toc35030776)

[**1.1 Abstract** 2](#_Toc35030777)

[**1.2 Module Description** 2](#_Toc35030778)

[**2. SYSTEM ANALYSIS** 4](#_Toc35030779)

[**2.1 Existing System** 4](#_Toc35030780)

[**2.2 Proposed System** 4](#_Toc35030781)

[**2.3 Feasibility Study** 5](#_Toc35030782)

[**3. SYSTEM CONFIGURATION** 7](#_Toc35030783)

[**3.1 Hardware Requirement** 7](#_Toc35030784)

[**3.2 Software Requirement** 7](#_Toc35030785)

[**4. SOFTWARE DESCRIPTION** 7](#_Toc35030786)

[**4.1 Overview of Front-end** 7](#_Toc35030787)

[**4.2 Overview of Back-end** 8](#_Toc35030788)

[**5. SYSTEM DESIGN AND DEVELOPMENT** 10](#_Toc35030789)

[**5.1 Data Flow Diagram** 10](#_Toc35030790)

[**5.3 Input Design** 12](#_Toc35030791)

[**6. TESTING** 13](#_Toc35030792)

[**6.1 White Box Testing** 13](#_Toc35030793)

[**6.2 Black Box Testing** 20](#_Toc35030794)

[**6.3 System Testing** 24](#_Toc35030795)

[**6.4 Test Case Reports** 25](#_Toc35030796)

[**7. SYSTEM IMPLEMENTATION** 29](#_Toc35030797)

[**7.1 Implementation Procedure** 29](#_Toc35030798)

[**7.2 Coding** 31](#_Toc35030799)

[**7.3 Testing** 33](#_Toc35030800)

[**7.4 Installation** 33](#_Toc35030801)

[**7.5 Documentation** 33](#_Toc35030802)

[**8. CONCLUSION AND FUTURE ENHANCEMENT** 34](#_Toc35030803)

[**9. SCREENSHOTS** 34](#_Toc35030804)

[**10. SOURCE CODE** 36](#_Toc35030805)

[**11. BIBLIOGRAPHY** 40](#_Toc35030806)

# **1. INTRODUCTION**

## **1.1 Abstract**

Mentoring is a conventional method of transferring knowledge and ideas from a confirmed professional in an society to an inexperienced member in the sector. Education sector has found mentoring as quite effective tool since long back and with the advent of new technologies, comes an idea of online mentoring. Instead of face-to-face meetings, Mentoring System uses asynchronous, electronic communications to establish and support the relationship between mentor and the student using virtual mode. Mentoring uses computerized medium to transfer knowledge and skills from teacher to student. It basically focuses on student and faculty

## **1.2 Module Description**

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client. On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

Additionally, java.net includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients. This lesson shows you how to use the Socket and ServerSocket classes.

If you are trying to connect to the Web, the URL class and related classes (URLConnection, URLEncoder) are probably more appropriate than the socket classes. In fact, URLs are a relatively high-level connection to the Web and use sockets as part of the underlying implementation. See Working with URLs for information about connecting to the Web via URLs.

# **2. SYSTEM ANALYSIS**

## **2.1 Existing System**

To connect to other machine we need a socket connection. A socket connection means the two machines have information about each other’s network location (IP Address) and TCP port.The java.net.Socket class represents a Socket. To communicate over a socket connection, streams are used to both input and output the data. The socket connection is closed explicitly once the message to server is sent. In the program, Client keeps reading input from user and sends to the server until “Over” is typed. Server application makes a ServerSocket on a specific port which is 5000. This starts our Server listening for client requests coming in for port 5000. Then Server makes a new Socket to communicate with the client. This makes connection between only one client and one server.

## **2.2 Proposed System**

The server and clients can run on different computers in the same network, e.g. Local Area Network (LAN). There can be multiple clients connect to a server and they can chat to each other, just like in a chat room where everyone can see other users’ messages. There’s no private chat between two users, for simplicity. After getting connected to the server, a user must provide his or her name to enter the chat. The server sends a list of currently online users to the new user. Every user is notified when a new user arrives and when a user has gone. Each message is prefixed with the username to keep track who sent the message. The application consists of two parts: server and client. Each part can run independently on separate computers.

## **2.3 Feasibility Study**

TCP/IP, or the Transmission Control Protocol/Internet Protocol, is a suite of communication protocols used to interconnect network devices on the internet. TCP/IP can also be used as a communications protocol in a private computer network (an intranet or an extranet).

The entire Internet Protocol suite -- a set of rules and procedures -- is commonly referred to as TCP/IP, though others are included in the suite. The TCP/IP protocol suite functions as an abstraction layer between internet applications and the routing/switching fabric.

TCP/IP specifies how data is exchanged over the internet by providing end-to-end communications that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination. TCP/IP requires little central management, and it is designed to make networks reliable, with the ability to recover automatically from the failure of any device on the network.

The two main protocols in the Internet Protocol suite serve specific functions. TCP defines how applications can create channels of communication across a network. It also manages how a message is assembled into smaller packets before they are then transmitted over the internet and reassembled in the right order at the destination address.

IP defines how to address and route each packet to make sure it reaches the right destination. Each gateway computer on the network checks this IP address to determine where to forward the message.

A subnet mask is what tells a computer, or other network device, what portion of the IP address is used to represent the network and what part is used to represent hosts (other computers) on the network.

**TCP/IP model layers**

*TCP/IP functionality is divided into four layers, each of which include specific protocols:*

* The application layer provides applications with standardized data exchange. Its protocols include the HTTP, FTP, Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP) and Simple Network Management Protocol (SNMP). At the application layer, the payload is the actual application data.
* The transport layer is responsible for maintaining end-to-end communications across the network. TCP handles communications between hosts and provides flow control, multiplexing and reliability. The transport protocols include TCP and User Datagram Protocol (UDP), which is sometimes used instead of TCP for special purposes.
* The network layer, also called the internet layer, deals with packets and connects independent networks to transport the packets across network boundaries. The network layer protocols are the IP and the Internet Control Message Protocol (ICMP), which is used for error reporting.
* The physical layer, also known as the network interface layer or data link layer, consists of protocols that operate only on a link -- the network component that interconnects nodes or hosts in the network. The protocols in this lowest layer include Ethernet for local area networks (LANs) and the Address Resolution Protocol (ARP).

# **3. SYSTEM CONFIGURATION**

## **3.1 Hardware Requirement**

* Processors: **Intel** Atom® **processor** or **Intel**® Core™ i3 **processor**.
* Disk space: 1 GB.

## **3.2 Software Requirement**

* Operating systems: Windows\* 7 or later, macOS, and Linux.
* > Java- jdk 8

# **4. SOFTWARE DESCRIPTION**

## **4.1 Overview of Front-end**

**Terminal**

Under Linux there are GUIs (graphical user interfaces), where you can point and click and drag, and hopefully get work done without first reading lots of documentation.

**Java**

The Java is usually installed as C:\Program Files\Java\jdk1.7.0\_67 on those machines where it is available; putting C:\Program Files\Java\jdk1.7.0\_67 in your Unix shell’s search path makes it possible to start it by typing the command: java

On Windows machines where you have installed Java from the oracle corporation, the Java command will be available. If you have the Java .exe launcher installed. Setting environment variables for other ways to launch java.

You must have administrator privilege to install the JDK on Microsoft Windows.

To run the JDK installer:

* + Start the JDK 11 installer by double-clicking the installer's icon or filename in the download location.
  + Follow the instructions provided by the Installation wizard.
  + After the installation is complete, delete the downloaded file to recover the disk space.

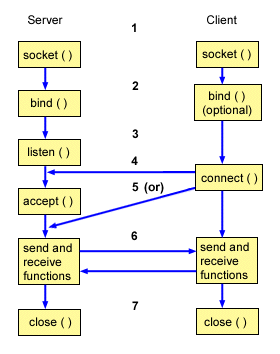
## **4.2 Overview of Back-end**

Sockets are commonly used for client and server interaction. Typical system configuration places the server on one machine, with the clients on other machines. The clients connect to the server, exchange information, and then disconnect.

A socket has a typical flow of events. In a connection-oriented client-to-server model, the socket on the server process waits for requests from a client. To do this, the server first establishes (binds) an address that clients can use to find the server. When the address is established, the server waits for clients to request a service. The client-to-server data exchange takes place when a client connects to the server through a socket. The server performs the client's request and sends the reply back to the client.

**This is a typical flow of events for a connection-oriented socket:**

* The socket() API creates an endpoint for communications and returns a socket descriptor that represents the endpoint.
* When an application has a socket descriptor, it can bind a unique name to the socket. Servers must bind a name to be accessible from the network.
* The listen() API indicates a willingness to accept client connection requests. When a listen() API is issued for a socket, that socket cannot actively initiate connection requests. The listen() API is issued after a socket is allocated with a socket() API and the bind() API binds a name to the socket. A listen() API must be issued before an accept() API is issued.
* The client application uses a connect() API on a stream socket to establish a connection to the server.
* The server application uses the accept() API to accept a client connection request. The server must issue the bind() and listen() APIs successfully before it can issue an accept() API.
* When a connection is established between stream sockets (between client and server), you can use any of the socket API data transfer APIs. Clients and servers have many data transfer APIs from which to choose, such as send(), recv(), read(), write(), and others.
* When a server or client wants to stop operations, it issues a close() API to release any system resources acquired by the socket.



Typically, a network configuration does not allow connections between a secure internal network and a less secure external network. However, you can enable sockets to communicate with server programs that run on a system outside a firewall (a very secure host).

# **5. SYSTEM DESIGN AND DEVELOPMENT**

## **5.1 Data Flow Diagram**

IP

PORT

Socket

Co

**5.2 Use Case Diagram**

User

Server

## 

## 

## **5.3 Input Design**

Socket

Connection

Message

IP/Port

**5.4 Output Design**

Socket

Message

Display

Connection

# 

# **6. TESTING**

## **6.1 White Box Testing**

WHITE BOX TESTING is testing of a software solution's internal structure, design, and coding. In this type of testing, the code is visible to the tester. It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security. White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing. It is usually performed by developers.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing.

During this test process approximately 90% of all errors would be covered and fixed. In the Release v0.1beta test plan include Acceptance, Functional, Performance and User Acceptance testing. In the Release v0.9beta, as shown in Fig. 4, test plan will consist of Integration, Acceptance, Bash & Multi-User testing, Technical, User Acceptance and

Regression testing. Regression testing of outstanding errors will be performed on an ongoing basis. When all errors would be fixed and tested agaIn that means that the system works in integrated manner and the Release v0.9beta is the final, proving of the system as a single application that implement and fulfill the policy and strategy for testing.

we have a look on general tools which we need to support the testing process and consider the testing framework we want to use. The framework consists of a test management tool, a test automation component which acts as integration point for the testing toolset and an integrated defect tracking tool.

We assume ​ MS Project is used for overall resource planning and work breakdown structure (WBS). This means we derive the primary planning information from a MS project plan which is maintained by the overall project manager. This is also the primary tool for maintaining planned and actual effort.

Open Office shall be the tool to create the documentation which cannot be derived automatically from any testing tool. This includes test policy, test strategy documents, test design, GitHub.

We assume the development team make use of a collaborative versioning tool like www.github.com​ . This could be shared with the testing team. This would facilitate not only access to the software components of the SUT but could be also used as document repository with version control and wikis.

The central tool for ​ planning​ , ​ control ​ and also ​ reporting ​ shall be the test management tool.

The tool is ideally also linked to the tools for analysis and design and also implementation and test execution. This is why we created a shortlist with possible open source solutions.

One option was ​ [www.testlink.org](http://www.testlink.org). Unfortunately we lost quickly the confidence into this product. We could not find valuable documentation and also the demo was not working.

Therefore it was easy to decide for w ww.squasthtest.org. A product which offered diverse documentation and we had only minor technical problems to install the components (see also appendix). Squash [5] is an open source project aiming to structure and industrialize functional testing activities. These deliverables cover all functional qualification processes:

● manual testing (test repository management)

● functional test automation

● management of test datasets

The Squash TM (Test Management) features are:

- Multi-project repositories with milestones

- Requirement management

- Test case management

- Run management

- Bug Tracking

- Monitoring and reporting

- Collaborative work

Requirements workspace: the module which supports,

- documentation of determined scope and risks including objectives of testing

- requirements can be linked to test cases and test steps can be linked to requirements

- requirements workflow, versioning, logging, printing

- requirements tree (bulk editing, export, ...)

Campaign workspace: the module which supports[5]

- monitoring and documentation of progress, test coverage and exit criteria

- Execution of test suites and individual test cases following the test procedures

- Re-execute the tests that prefiously failed in order to confirm a fix

- log the outcome of the test execution and record the identities and versions of the SUT

- compare actual results with expected results and report discrepancies as incidents

In addition to Squash TM a toolbox is offered, called Squash TA (Test Automation).

Dedicated to the industrialization of automated tests execution. Squash TA natively interacts with various open source robots like Selenium. Squash TA is a management controller of automated tests of Web applications, web services and batches. Squash TA's features are based on developments which were started in 2009 by Henix team. Squash TA has been made to answer to automation needs of Third Party software testing lead by Henix. These developments have been released to the Squash Community and constitute the base of Squash TA. [5]

Features [5]:

- The realization of robust automated tests

Squash TA can build automated tests less sensitive to the SUT (System Under Test) changes.

- The industrialization of your test executions

Your tests are reproducible at will through the possibilities to control the context and the execution environment.

- Universal : TA manages all your automated tests

Squash TA offers you to manage numerous types of automated tests : web applications, web services, batches. Moreover Squash TA may be deployed in agile environment and for ATDD methodology.

- Linkable with open source robots

Squash TA supports the main open source tools : Selenium I and II, Sahi, SoapUI.

**Component Testing (unit test)**

Unit testing is the practice of testing of the functions (methods) and analysis of the developer's code. Therefore developer tests some parts of his or her code after finishing it. This help developers to identify failures or errors in the code and improve the quality of the code. Also unit testing gives the developer a chance to see the features of the software in action.

IntelliJ IDEA features robust, fast, and flexible static code analysis. It detects compiler and runtime errors, suggests corrections and improvements before compile. This cross-platform IDE with own set of several hundred code inspections available for analyzing code on-the-fly in the editor. Below, in the there is a short example showing a careless mistake.

Here the first if-condition may lead to NullPointer exception being thrown in the second if, as not all situations are covered. At this point adding an assertion in order to avoid a NullPointer being thrown during the application runtime. Figure 13 shows us exactly what we get from the intention action in IntelliJ IDEA build-in inspection code.

Unit is a small and powerful open source Java framework for writing and running automated unit tests. The libraries for JUnit are shipped with IntelliJ IDEA and they can be easily added to the project with the necessary library to the classpath automatically. With JUnit it is easier to write code faster. It increases the quality of code and it is less complex. All tests could be ordered in test suites and test cases and it saves much time. Since the tests are programmed directly in Java, testing with JUnit is as easy as compiling. The test cases are self-monitoring and therefore repeatable. In the Figure 14 below show the example of test fixture in which the purpose is to verify that a test is running in a fixed environment.

Also for testing the units of the database layer, could be used DbUnit that is a JUnit extension.

This extension is used for controlling a database dependency within applications by allowing developers to manage the state of a database throughout a test. The developers write code in a SQL language to extract the data. DbUnit has the ability to export and import database data to and from XML datasets. The main benefit of DbUnit is, that the framework drops the database and creates a new one with test data before every execution of a testcase. The framework comes with own declared methods to compare whole datasets with each other.

When developer run a test case he gets only the result of the test Pass/Fail, but he doesn't know if he tested all written code. For this reason, there is GroboUtils that include code coverage tool, automated documentation and multi-threaded tests. For using code coverage first need to add \*.jar file to the directory and change Ant build.xml file in the project with new reference and location of GroboCoverage. Code coverage tool works by inserting "probes" into the post-compiled class files. These probes call out to a logger, telling it which class, method index and probe index was invoked. During the post-compilation of the class files, a set of data files get generated, which record what a method index and probe index translate into with regards to the source file. In the below, there is a short example to show how to run the test with code coverage enabled using JUnit task.

Units of the source code is simple a classes of a programming language and describing the objects and behavior of the objects that are created from the class. In other words unit could be depends on another units. So if the developer want to test any unit he needs to know that this unit could have dependencies to other units and that mean that he needs to initialize the other affinity units too. Mock objects help a developer to test the interactions between the objects in the programs. For this purpose it easier to use an open source library JMock that give an opportunity to makes it quick and easy to define mock objects, precisely specify the interactions between objects and it is plugs into JUnit test framework, so it is easy to extend. In Figure 16 show how to import jMock classes, define test fixture class and create a "Mockery" that represents the context in which the Publisher exists. The context mocks out the objects that the Publisher collaborates with and checks that they are used correctly during the test.

**Security Testing**

It is one of the most important technical challenges and as the customer belongs to an governmental organisation it become very important to produce secure software as security is a key limiting factor in deploying information technology in authority organizations. Security testing can help to build a secure application.

As security testing encompasses a wide area it become hard to give a definition of this testing.

In general, it is a process of determining to what extent a system as a whole secures confidentiality, integrity and availability of information. Confidentiality is the prevention of intentional or unintentional unauthorized disclosure of content. The concept of availability ensures that information is accessible. Integrity concerns guarantee that information protected from being modified by unauthorized parties and that the information provided by the system is correct.

In our case we need to make tests that are conducted to prevent any unauthorized access to the software code and modification of the data. In first discovery stage the purpose is to identify systems within scope and the services in use. Discovery phase is vulnerability analysis, in which the applicable vulnerability classes that match the interface are identified. After the discovery stage, the goal of vulnerability scan is to find for known security issues by using automated tools and plugins like Black-box vulnerability scanners and match conditions with known vulnerabilities.

Also, for our project, we will use OWASP Zed Attack Proxy (ZAP), easy and open source free tool that provides automated scanners for OWASP top 10 vulnerabilities as well as a set of tools that allow to make professional penetration testing of security vulnerability in web applications. Security testing ends with the security review stage. This stage does not apply any of approaches like discovery, vulnerability scan, vulnerability assessment, security assessment, penetration test or security audit. This is a confirmation that external or internal security standards are applied to system or product.

Remark: Security testing shall be performed also on component testing level

Narrative: A numerous numbers of security checks and vulnerabilities can be automated and found in the code review with using open source utility OWASP Dependency Check Gradle Plugin which could be easy integrated in IntelliJ IDEA project. Features of the OWASP Dependency Check can be run in a continuous integration to determine if there are new vulnerabilities discovered based on the addition of a new dependency, or the discovery of a new vulnerability in an existing dependency.

Of course, this plugin also provides numerous security inspections. These can inform a developer of potential security problems in the code. For example, in Figure 21 this code executes a dynamically generated SQL string, which might be susceptible to SQL Injection

## **6.2 Black Box Testing**

BLACK BOX TESTING, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

1. black box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.
2. black box test design technique: Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

Black Box Testing method is applicable to the following levels of software testing:

* Integration Testing
* System Testing
* Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black-box testing method comes into use.

**Acceptance Testing**

Before starting this phase of testing, according to acceptance criteria specified in project description in first two releases 9 of 10 test cases should be completed successfully and the pass rate 90% must be achieved before the software will be accepted for starting acceptance testing. Acceptance testing is usually a final phase of functional testing and it is done by main stakeholders of the project like users or customers. The aim of the acceptance testing is to check if all project requirements have been satisfied.

There are different types of acceptance testing. Most common is ​ user acceptance testing. ​ The general focus is on the usability and functionality of the system, hereby validating the fitness-for-use of the system by the user or customer. This type is foreseen in the current planning, maybe we add also ​ operational acceptance testing.

This is performed with the aim to assure that the new system will be integrated in the existing production environment of the customer. The focus is on the operational issues like backup and restore procedure, restart capabilities, regular operational housekeeping tasks like database reorganization and so on. We ignore ​ contract and regulation acceptance testing​ , a system is performed against acceptance criteria as documented in a contract (legal and safety standards, norms and regulation), before the system is accepted.

**Integration testing**

Integration testing is a kind of testing which the software modules are combined and will be stated as a group, this type of testing sometimes called grey-box tasting which means the tester partially knows about internal structure and algorithm of the software. Usually the integration testing starts after unit testing, after all components of the project have already been tested. The purpose of integration testing is to verify functional, performance, and reliability of the software.

Testing of the integrated modules is designed to find latent defects as well as interface and database defects.The integration test depends on integration approach and in our project we use a Top-down approach which gives us a possibility first to test top modules in the hierarchy and then all extensions that are simulated with Mock objects. This type of integration approach could be time-consuming, but the benefit is that faults of the program could be been found easier and this testing could be performed with implementation. TestNG ​ is a Java testing framework inspired by JUnit and it is compatible with Eclipse, IDEA, Selenium, Maven. The test process could be simplified with using many advantages of TestNG. It uses annotation to provide a great way to test code in different ways: unit, regression, functional, integration and many others types of testing. It also allows to divide test methods in complex groupings, it means that not only methods can belong to the groups but also a group or the set of groups contain to other groups(MetaGroups) as it can be seen below in This gives a maximum suppleness in a meaning of how to partition the tests. This powerful framework generates its own reports in XML or HTML formats.

Continuous integration

Continuous Integration is a development practice that integrate the code to the entire team regularly in a shared repository to verify that it works all together. [18] When a developer commits his work, the whole system will be compiled and transferred to the test environment, where unit-tests, integration tests, system tests run automatically and this helps to detect integration error as soon as it possible. GitLab CI is a open-source continuous integration service included with GitLab and after performing the tests in any programming language, the developers get immediate feedback if there is any problem. GitLab CI improves the quality of the product, documents all activities like build test and deploy. Errors in the code can be found more easily and be excluded in time. [17]

## **6.3 System Testing**

Major focus at this level is the complete system. After component testing and integration all test we will focus on a consistent and integrated system.

**Functional Testing**

Functional testing is a kind of black-box testing, it means software or application will be tested under the condition that knowledge of the application's code/internal structure and programming knowledge in general is not required. The goal of functional testing is to verify that the application is behaving the way it was designed to. One of the best tools to test the functionality of web applications is Selenium. We assume, that all functions can be triggered using the GUI. See test cases in [1]

“Selenium is a web testing tool which uses simple scripts to run tests directly within a browser. In simple terms, “it automates browsers”. It is a portable software testing framework for web applications that provides a record/playback tool for authoring tests without learning a test scripting language by using Selenium IDE”

**Performance Testing**

Performance testing is a non-functional testing which determine the system parameters in terms of scalability, stability and some others quality attributes. For normal behavior of the program in our project, we will use load and stress testing.

Load testing is one of the important and in the same time the simplest form of performance testing is a load testing. The load testing is performed to determine a system’s behavior under normal and at peak conditions. It helps to identify the highest operating capacity of an application as well as any slow downs in the application and on the hardware side, determine

which element is causing reduction. Stress testing involves testing beyond normal operational capacity. Normally this kind of test is done to determine the system's robustness in terms of extreme load for example multi-user testing. It is performed to identify the defects in an application when multiple users login to the application and it also demonstrates the system’s reliability by establishing that the various processes do not have a negative influence on each other. This test also helps in finding, analysing and measuring the complication in system parameters such as response time, throughput, locks/dead locks or any other issues associated with concurrency.

In our case, performance testing must be ensured that the system provides acceptable response times not more than 6 seconds on the server side. For this purpose, we will use Apache JMeter to test performance both on static and dynamic resources. By building a test plan is composed of a sequence of test components that determine how the load or stress test will be simulated. The output performance data could be done in several ways, including CSV and XML files, and graph you can see how Apache JMeter perform performance testing.

## **6.4 Test Case Reports**

|  |  |
| --- | --- |
| **TEST CASE SPECIFICATION** | **DESCRIPTION** |
| Test Case ID(TC\_ID) | Unique ID to identify/report the bug if present in the functionality of software |
| Test Case Objective | The purpose of the test. The lists can be generated to perform intended task, for which software is developed. Results should always follow the test case objective. |
| Prerequisite | This can include environment setup, supporting software environment setup. for the project, or any fields in which user will give the input. So that test cases can be planned accordingly. |
| Steps | This includes steps to be performed to give the input to the system, so that system can perform its specified task and display the result accordingly. If automated testing is used, then, these steps are translated to the scripting language of the tool. |
| Input Data | The choice of input data will be depended on the test case itself and the technique followed in the test case.  For e.g. equivalence partitioning, boundary value analysis etc. |
| Expected Result | It can be the user required output to be shown |
| Actual Result | This step should do a comparison of the expected and actual results to highlight any differences |
| Status | Whether expected results and actual result match, if it matches then PASS or else FAIL |

In order to efficiently cover the functional by tests, test cases need to be divided into types. If you start doing it, then their number will increase at least in three times. Various sources describe types in different ways, but the essence of the division does not change. We offer the following types of test cases that should divide your test plan.

**Positive**

There are test cases aimed at checking the correct operation of the claimed functionality using the correct input format specified in the software documentation.

**Negative**

There are test cases that check your anticipated every possible situation that should lead to an error message. Also, this type of test cases includes a verification that can lead to unexpected situations, ie those that are not described in the documentation.

For example, you can test the field email, introducing the characters that are not included in the list mentioned above. You can also try to interrupt the fields, check whether the data is stored in the system reboot or exposure to other external factors.

**Boundary value**

To check the values on either side constraints. One of these relates to tests positive, the other to negative. It is better to isolate them not to miss. These tests are an indication that you own test design, which you can see below.

For example, you found the information in the documentation that the password must contain at least 6 and no more than 60 characters. So you have to ascertain what happens if you type 5, 6, 60 and 61 characters. Do not forget about a case when the field is empty.

If documentation does not describe such restrictions, you can offer them themselves, discussing with the team!

**Integration**

Check connections between different parts of the program. This is not exactly the type of test cases, but rather the level of testing. But such tests are required. You have to describe them, especially if your system consists of at least two modules.

You can write test cases to check the appearance of the data entered in another part of the software.

For example, if you have a payment for a certain type of functionality. Then you definitely need to ascertain whether that functionality becomes available after payment. After all, developers are likely to have implemented these parts separately, and problems could arise when they integrate those parts.

**Testing localisation**

Check all UI elements in different languages and their locations (if there is a support for languages with different rules of writing and reading).

For example, if your software supports one of the location where the UI is placed from the right to left, you should pay attention to the work of Drop-Down List, check boxes, switching elements On / Off, etc.

Written tests to check GUI. You can describe the appearance of tips in the program hotkeys, errors, etc.

If you have enough time, you can write test cases that will help you with cross-platform testing, especially if the program depends on platforms.

If you have a great software that supports multiple languages, make a separate chapter for localisation test case.

If you are not using any test case management tool, you can use any open source tool or Excel Sheet to manage and execute your test cases.

Test case templates and examples are very useful because using them you can save time and resources for the cover product by a large number of test cases.

***All the above conditions are checked and tested and the Passed as positive.***

# **7. SYSTEM IMPLEMENTATION**

## **7.1 Implementation Procedure**

Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using spiral model.

The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

Below diagram shows the different phases of the Spiral Model:

Each phase of Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1. Objectives determination and identify alternative solutions: Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.



1. Identify and resolve Risks: During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.
2. Develop next version of the Product: During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
3. Review and plan for the next Phase: In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

**Risk Handling in Spiral Model**

A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype. The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development. Prototyping Model also support risk handling, but the risks must be identified completely before the start of the development work of the project. But in real life project risk may occur after the development work starts, in that case, we cannot use Prototyping Model. In each phase of the Spiral Model, the features of the product dated and analyzed and the risks at that point of time are identified and are resolved through prototyping. Thus, this model is much more flexible compared to other SDLC models.

## **7.2 Coding**

The most of the major decisions about the system have been made. The goal of the coding phase is to translate the design of the system into code in a given programming language. For a given design, the aim of this phase is to implement the design in the best possible manner. The coding phase affects both testing and maintenance profoundly. A well written code reduces the testing and maintenance effort. Since the testing and maintenance cost of software are much higher than the coding cost, the goal of coding should be to reduce the testing and maintenance effort. Hence, during coding the focus should be on developing programs that are easy to write. Simplicity and clarity should be strived for, during the coding phase.

An important concept that helps the understandability of programs is structured programming. The goal of structured programming is to arrange the control flow in the program. That is, program text should be organized as a sequence of statements, and during execution, the statements are executed in the sequence in the program.

For structured programming, a few single-entry-single-exit constructs should be used. These constructs includes selection (if-then-else), and iteration (while - do, repeat - until etc). With these constructs it is possible to construct a program as sequence of single - entry - single - exit constructs. There are many methods available for verifying the code. Some methods are static in nature that is, that is they do not involve execution of the code. Examples of such methods are data flow analysis, code reading, code reviews, testing (a method that involves executing the code, which is used very heavily). In the coding phase, the entire system is not tested together. Rather, the different modules are tested separately. This testing of modules is called "unit testing". Consequently, this phase is often referred to as "coding and unit testing". The output of this phase is the verified and unit tested code of the different modules.

## **7.3 Testing**

The testing phase of the software development lifecycle (SDLC) is where you focus on investigation and discovery. During the testing phase, developers find out whether their code and programming work according to customer requirements. And while it's not possible to solve all the failures you might find during the testing phase, it is possible to use the results from this phase to reduce the number of errors within the software program.

Before testing can begin, the project team develops a test plan. The test plan includes the types of testing you'll be using, resources for testing, how the software will be tested, who should be the testers during each phase, and test scripts, which are instructions each tester uses to test the software. Test scripts ensure consistency while testing.

There are several types of testing during the test phase, including quality assurance testing (QA), system integration testing (SIT), and user acceptance testing (UAT).

## **7.4 Installation**

Program can be done using the following command,

Javac Chatserver.java

Java Chatserver

Javac Chatclient.java

Java Chatclient

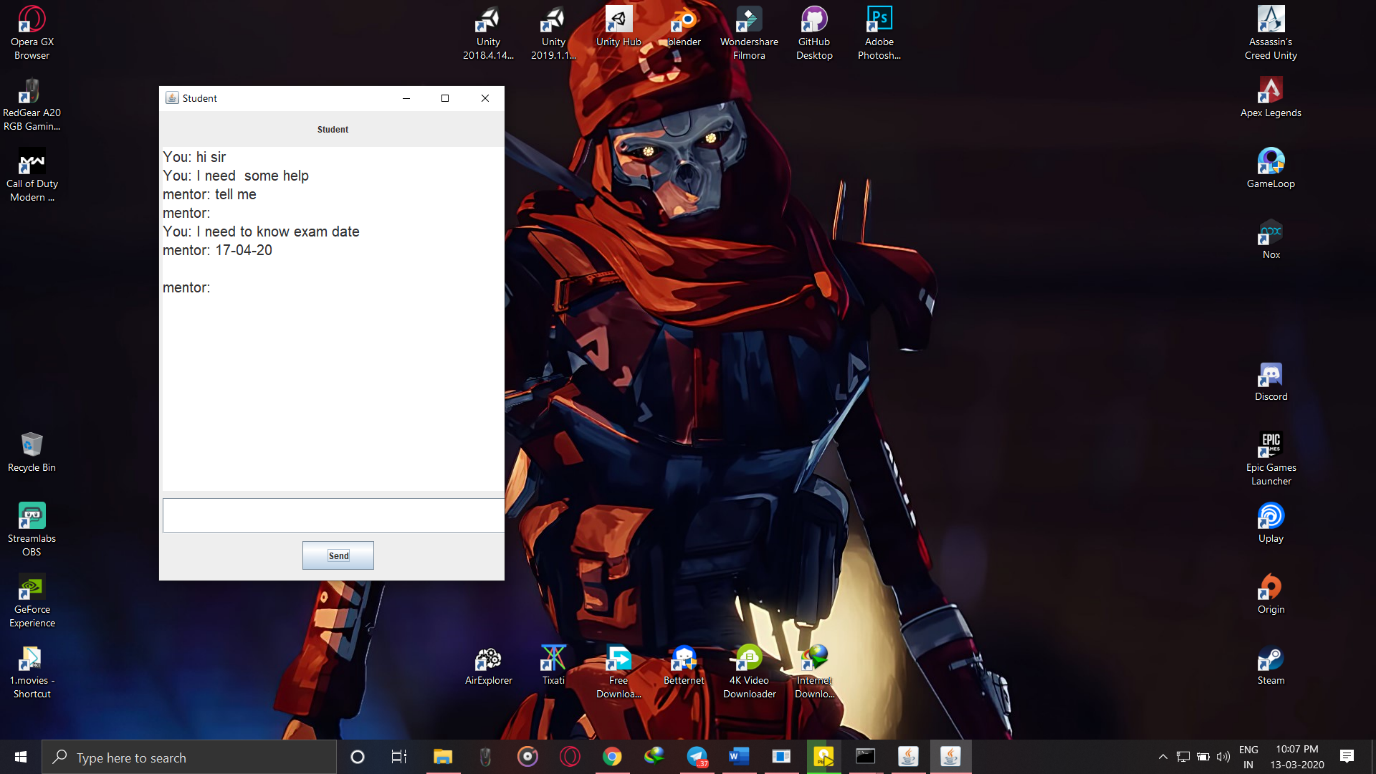
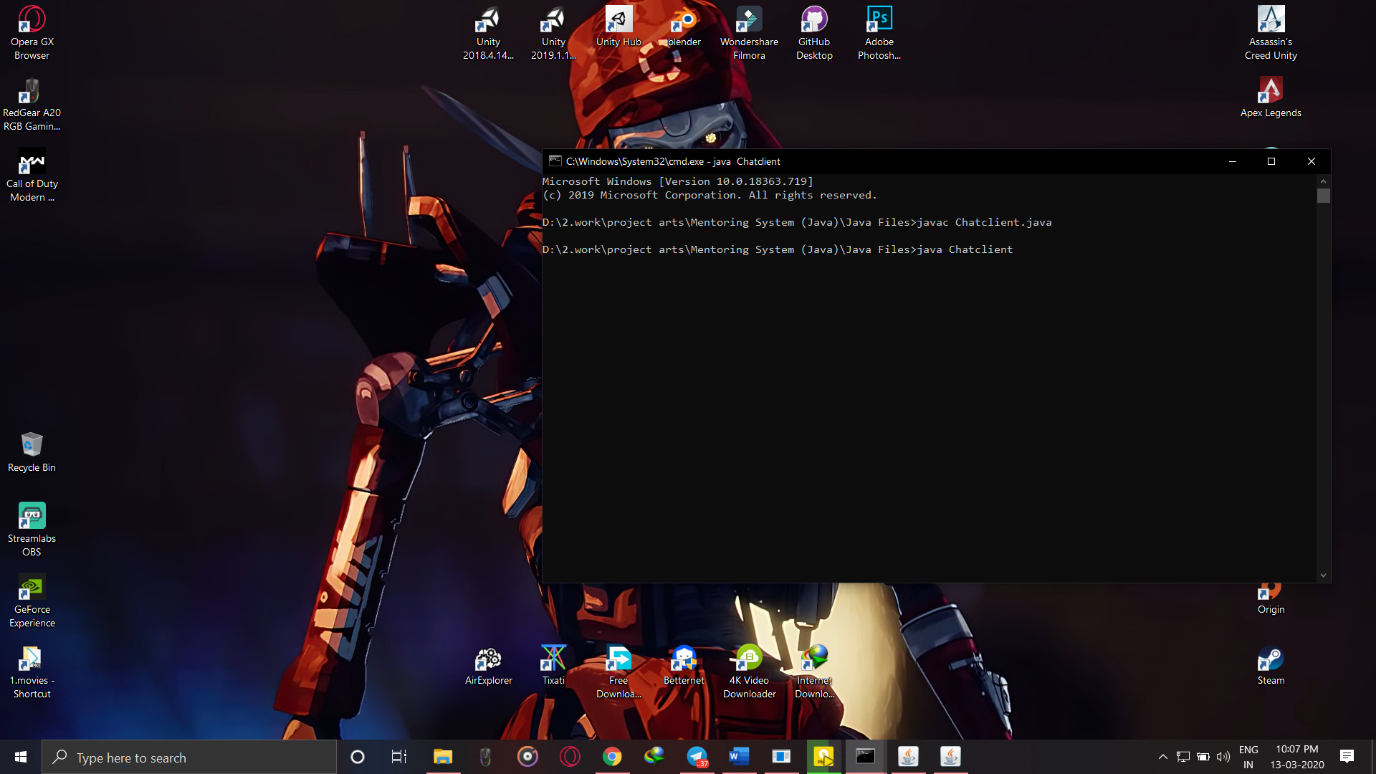
## **7.5 Documentation**

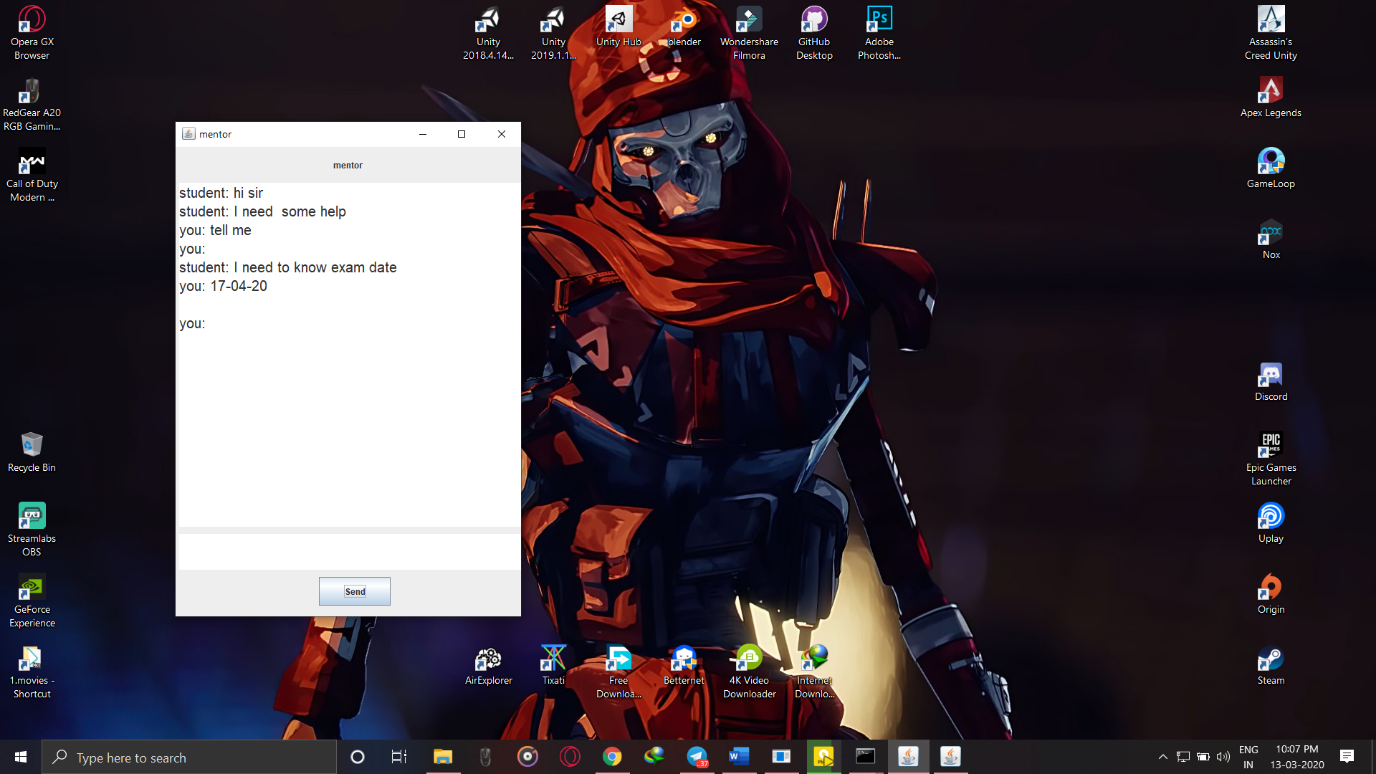
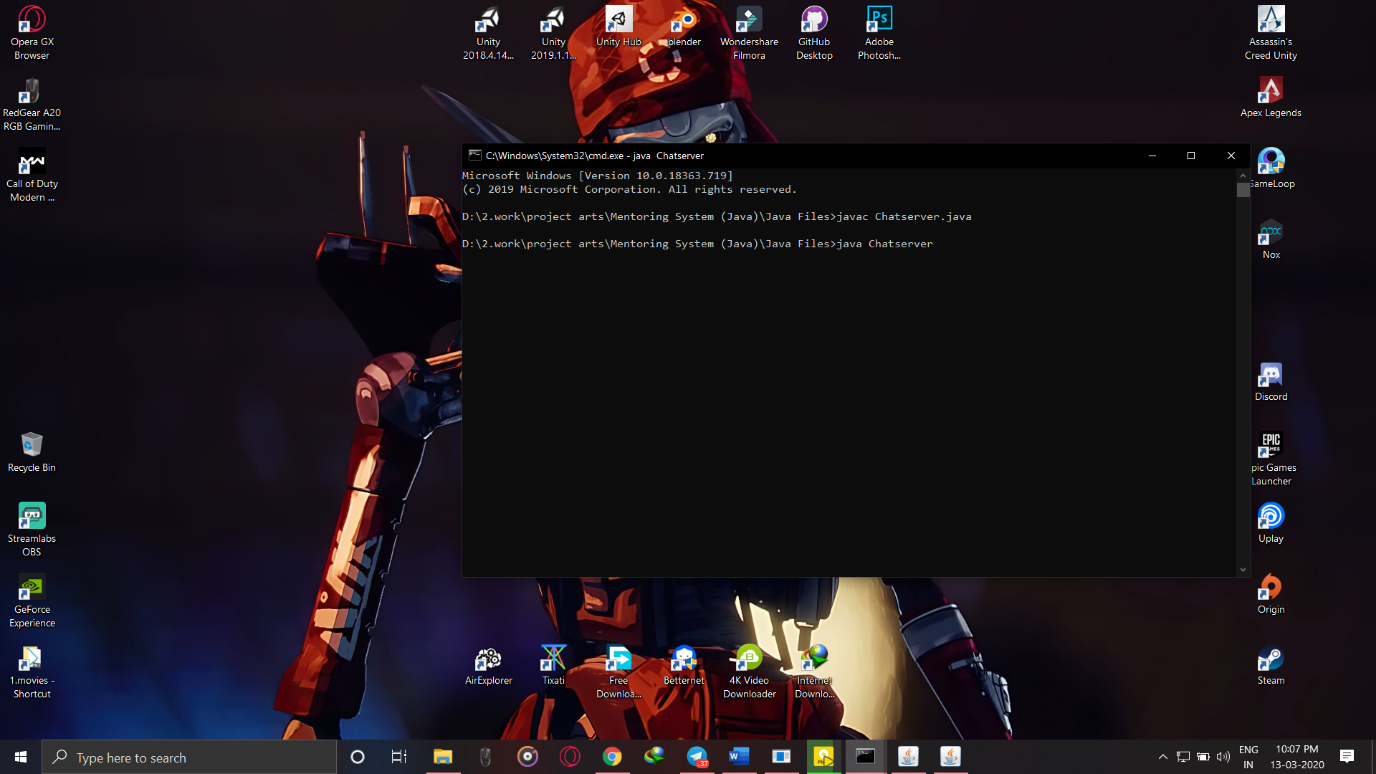
Document of this project can be found at Mentoring\_System\_Java.doc

# **8. CONCLUSION AND FUTURE ENHANCEMENT**

Thus the Project has been executed and tested correctly. In future, the Algorithm may be changed and enhanced by increasing the efficiency and time consumption.

# **9. SCREENSHOTS**





# 

# **10. SOURCE CODE**

**# chat\_server.java**

import javax.swing.\*;

import java.awt.event.\*;

import java.io.\*;

import java.net.\*;

class Chatserver {

public static void main(String[] args) {

try{

MyserverT myservert = new MyserverT();

myservert.start();

}

catch(Exception e){

}

}

}

class MyserverT extends Thread{

public JLabel l1,l2;

public JTextArea area,space;

public JButton b;

public DataInputStream din;

public DataOutputStream dout;

public ServerSocket ss;

public String msgin="",msgout="";

public Socket s;

MyserverT(){

JFrame f = new JFrame("mentor");

l1 = new JLabel();

l1.setBounds(220,10,280,30);

l1.setText("mentor");

f.add(l1);

area = new JTextArea();

area.setBounds(5,50,490,480);

area.setVisible(true);

area.setFont(area.getFont().deriveFont(20.0f));

area.setEditable(false);

f.add(area);

b = new JButton("Send");

b.setBounds(200,600,100,40);

f.add(b);

space = new JTextArea();

space.setBounds(5,540,490,50);

space.setFont(space.getFont().deriveFont(25.0f));

f.add(space);

f.setSize(500,700);

f.setLayout(null);

f.setVisible(true);

}

public void run(){

try{

ServerSocket ss = new ServerSocket(39521);

Socket s = ss.accept();

din = new DataInputStream(s.getInputStream());

dout = new DataOutputStream(s.getOutputStream());

while(!msgin.equals("end")){

msgin = din.readUTF();

area.append("student: "+msgin+"\n");

b.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent e){

msgout = space.getText().toString();

call(msgout,dout);

//space.setText("");

b = new JButton("Send");

}

});

}

}

catch(Exception e){

space.setText("Oops...!!! Problem in starting Server Socket");

space.setEditable(false);

b.setEnabled(false);

}

}

public void call(String a,DataOutputStream dout){

try{

area.append("you: "+a+"\n");

dout.writeUTF(a);

dout.flush();

space.setText("");

}

catch(Exception e){

space.setText("Error in sending message");

space.setEditable(false);

b.setEnabled(false);

}

}

}

**# chat\_client.java**

import javax.swing.\*;

import java.awt.event.\*;

import java.io.\*;

import java.net.\*;

class Chatclient {

public static void main(String[] args) {

try{

MyclientT myclientt = new MyclientT();

myclientt.start();

}

catch(Exception e){

}

}

}

class MyclientT extends Thread{

public JLabel l1,l2;

public JTextArea area;

public JTextField space;

public JButton b;

public DataInputStream din;

public DataOutputStream dout;

public String msgin="",msgout="";

public Socket s;

MyclientT(){

JFrame f = new JFrame("Student");

l1 = new JLabel();

l1.setBounds(220,10,280,30);

l1.setText("Student");

f.add(l1);

area = new JTextArea();

area.setBounds(5,50,490,480);

area.setVisible(true);

area.setFont(area.getFont().deriveFont(20.0f));

area.setEditable(false);

f.add(area);

b = new JButton("Send");

b.setBounds(200,600,100,40);

f.add(b);

space = new JTextField(20);

space.setBounds(5,540,490,50);

space.setFont(space.getFont().deriveFont(25.0f));

f.add(space);

f.setSize(500,700);

f.setLayout(null);

f.setVisible(true);

}

public void run(){

try{

Socket s = new Socket("localhost",39521);

din = new DataInputStream(s.getInputStream());

dout = new DataOutputStream(s.getOutputStream());

while(!msgin.equals("end")){

b.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent e){

msgout = space.getText();

call(msgout,dout);

//space.setText("");

b = new JButton("Send");

}

});

msgin = din.readUTF();

area.append("mentor: "+msgin+"\n");

}

}

catch(Exception e){

space.setText("Oops...!!! Problem in starting Client Socket");

space.setEditable(false);

b.setEnabled(false);

}

}

public void call(String a,DataOutputStream dout){

try{

area.append("You: "+a+"\n");

dout.writeUTF(a);

space.setText("");

}

catch(Exception e){

space.setText("Error in sending message");

space.setEditable(false);

b.setEnabled(false);

}

}

}

# **11. BIBLIOGRAPHY**

There is no Bibliography used in this document as everything is explained in detail.