

**Software Specification**

**Version 1.0**

By: Team4 ©

**Authors:**

Adhikari, Bibek Babu  
Banerjee, Avik   
Darel, Guy A.  
Fregoso Rodriguez, Jose  
Rodriguez, Eric A.  
Yu, Bert Boshun

**Affiliation**: University of California, Irvine

**Table of contents**

**Page No.**

**Glossary……………………………………………………………………………………... 3  
1 Client Software Architecture Overview ……………………………………………… 4**

1.1 Main data types and structures ……………………………………………… 4

1.2 Major software components …………………………………………………….. 5

• Diagram of module hierarchy……………………………………………… 5

1.3 Module interfaces ……………………………………………………………….. 5

• API of major module functions …………………………………………… 5

1.4 Overall program control flow ………………………………………………….... 6

**2 Server Software Architecture Overview …………………………………………… .... 7**

2.1 Main data types and structures…………..…………………………….……7

2.2 Major software components ……….…………………………………………..7

• Diagram of module hierarchy ………………………………………….....7

2.3 Module interfaces …………………………………………...……………………8

• API of major module functions …………………………………………...8

2.4 Overall program control flow …………………………………………………….8

**3 Installation …………....…………………………………………...…………………………… 11**

3.1 System requirements, compatibility…………………………………………………….. 11

3.2 Setup and configuration ………………………………………………………………….. 11

3.3 Building, compilation, installation ………………………………………………………. 11

**4 Documentation of packages, modules, interfaces ……………………………………...**

4.1 Detailed description of data structures ………………………………………….……...12

• Critical snippets of source code …………………………………………………….12

4.2 Detailed description of functions and parameters …………………………………….12

• Function prototypes and brief explanation ………………………………..………12

4.3 Detailed description of the communication protocol …………………………..…….. 14

• Explanation of communication related function calls ……………………...……14

• Explanation of communication protocol flags/settings……………………..…...14

**5 Development plan and timeline…………………………………………...……………..……15**

5.1 Partitioning of tasks…………………………………………...………………………………. 15

5.2 Team member responsibilities…………………………………………...……………….…...15

**6.- Copyright……………………………………………………………………………….………..18  
7.- Error messages……………………………………………………………………….………...19  
8.- Index………………………………………………………………………………………..……..20**

**Glossary**

**A**

**Active Status:** A user is currently connected to server

**Accept():** Accepts a connection. Typically clocks until a client connects with the server

**B**

**Bidirectional:** both sides can send and receive

**Bind():** Assigns the address specified to the socket referred by sockfd

**C**

**Chat history:** This is a feature that will let the user save the chatting log from previous chat sessions.

**Client:** initiates requests for service

**Char:** A single character

**Connect():** The socket is connected to the address of the server

**I**

**Idle Status:** A user will be show as idle if inactive for five minutes

**IP Address:** a unique string of numbers separated by periods that identifies each computer using the Internet Protocol to communicate over a network.

**Instant Messaging:** Feature that will allow the users to send and receive messages in fractions of seconds.

**Internet:** communication network to exchanges messages

**Int:** An 8 bit integer

**L**

**Listen():** Marks the socket referred to by sockfd as a passive socket, or a socket that will be used to accept incoming requests

**O**

**Offline Status:** Shown that a user is not connected to the server

**G**

**Group Messaging:** Conversation between more than two people.

**P**

**Port Number:** A port number is a 16-bit unsigned integer, thus ranging from 0 to 65535.

**Point-to-point communication:** Communication between two points (server and client)

**R**

**Read():** Read up to count bytes from fd into the buffer

**S**

**Server:** provides a service function to one or more clients

**Sockets:** a network connections between two processes

**Socket():** Creates a socket using a system call

**Struct:** A composite data type that groups many data types such as int or char together

**W**

**Write():** Writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd

## 

## 1.Client Software Architecture Overview

### **1.1 Main data types and structures**

**Table 1**

|  |  |
| --- | --- |
| **Element name** | **Data Structure Type** |
| Username | String (char[x]) |
| Password | String (char[x]) |
| Security | String (char[x]) |
| Message | String (char[x]) |

**Table 2:** Sockets

|  |  |
| --- | --- |
| ServSocketFD | Int |
| DataSocketFD | Int |
| PortNo | Int |
| ServerAddress | Int |
| ClientAddress | Int |
| RecvBuf[256] | String (char[x]) |
| SendBuf[256] | String (char[x]) |

ServSocketFD- socket file descriptor(handles input and output) for service

DataSocketFD- socket file descriptor(handles input and output) for data

PortNo- port number to connect with

ServerAddress- server address(this host)

ClientAddress- client address we connect with

RecvBuf - message buffer for receiving a message

SendBuf - message buffer for sending a response

### 

|  |
| --- |
| **Figure 0.** This figure shows the directory diagram of the server mainly, with the database directory tree. |

### 

### **1.2 Major software components**

* Login
  + Sends username and encrypted password to server to determine if it can connect to the server or not.
* Send Message
  + Sends message to server to be received by another user.
* ReceiveMessage
  + Receives message from other users.
* SendtoGUI
  + Sends received messages and sent messages to the GUI to be displayed.
* Register
  + Allows users to register into the server. Asks for usernames, password.
* Encrypt
  + Encrypts the password to send to the server
* Sockets
  + Will pass in arguments through the command line
  + For the server:
    - Create a socket
    - Bind the socket to a port number
    - Place the socket in listen mode for connections
    - Accept incoming connections to the socket
  + For the client:
    - Create a socket
    - Connect the socket using connect()
    - Send and receive data, likely using read() and write() system calls

### 1.3 Module interfaces

* **Graphics:** Contains all of the graphics we are using for the GUI.(Ex. Login Window,Signup Window Message Box).
* **Client:** Contains necessary functions to login (with encrypted passwords), send and receive messages, and also send and receive data to and from the GUI and Server.
* **Server:** Contains necessary functions to store, process, send and receive messages with the client module.

### 1.4 Overall program control flow

Client receives login from GUI and send the username and encrypted password to the server, which returns either true or false depending on if the password is correct. If it’s true, then it connects to the server. While messaging, it will receive messages from the GUI, then send it to the server. Afterwards, it will wait until there is a message for it to receive and then it will run ReceiveMessage. Afterwards, it will send the received message to GUI. Then it will begin the loop of receiving and sending again until the user quits.

During registration, it will ask for username, password. It will send it to the server to be stored.

|  |
| --- |
| **The following shows the main processes on the client side.**  **As the figure shows, the client will be in an infinite loop that will wait for either an exit command to close the application, or a new input message to be sent. In the case that it receives a new message, it will then connect to the server, read the input message from user, and then send such message to the server.**  **Figure 1: Overall Control Flow for Client.** |

## **2. Server Software Architecture Overview**

### 2.1 Main data types and structures

* String: MessageReceived (Holds last Message)
* String: MessageStore (Holds latest Message)
* String: OnlineUsers (List of online users)
* Text file: Friends (Friends List)
* Text file: Login.txt (List of all registered users and passwords)

### 2.2 Major software components

* Diagram of module hierarchy

|  |
| --- |
| **Figure 2: Module Hierarchy**  This diagram shows the compilation order. GUI and Client are linked together, while Server and Client are linked together. You can notice that Server and GUI do not interact directly, any data has to first pass through Client. |

### 2.3 Module interfaces

**Please go to section 4.2 for further explanation and details of the following API.**

Client

* Void Encrypt(char\*);
* Void ClientCmp(char\* ServerOut);

Server

* char\* ServerCmp(char\* username, char\* password);
* Void FatalError(const char \*Program, const char \*ErrorMsg)

Server Database

* char\* ServerLogin(char\* username, char\* password);
* char\* ServerSignUp(char\* username, char\* password);
* char\* ServerAddFriend(char\* username);
* char\* ServerDeleteFriend(char\* username);

GUI

* Char \*GetLogin();
* Char\* GetPwd();
* Void GUIPrintError();
* Int GUISuccessLogin();
* char\* GUILoadConversation(char \*sender, char\* friend);
* Int GUIDrawConversation(char\* conv);

### 2.4 Overall program control flow

|  |
| --- |
| **The following diagram shows how the login process works.**  **Figure 3: Control Flow** |

|  |
| --- |
| **The following diagram shows the process of sending and receiving a message.**  **Once the GUI gets a message, it sends it to the client which will send the message to the server with a destination annexed to it. Then the server proceeds to send the message to the corresponding destination client, which will fetch the message and finally display on its GUI.**  **The blue path ( receiving process) is exactly the same, but backwards.**  **Figure 4: Messaging Process** |

|  |
| --- |
| **The following diagram shows the general process of the server threads.**  **The server will start by opening a new socket, then proceed by binding to corresponding clients. Next it will enter an infinite loop that will wait for either a new incoming connection, or an exit command, which will exit the server application.**  **In the case that there is a new connection, it will accept it, read the message from user, send it to its destination and lastly, close the connection to the current client so that it leaves space to new clients trying to connect to the server.**  **Figure 5: Client and Server General Control Flow** |

## 3. Installation

### 3.1 System Requirements

* **Operating System:** Linux CentOS 6.9
* **Disk Space:** 20 MB free.
* **RAM:** 512 MB or more highly recommended.
* **CPU:** Single core Intel processor or better
* **Internet:**

### **3.2 Setup and configuration**

To install this software, copy the Chat\_V1.0.tar.gz and Chat\_V1.0\_src.tar.gz file from the host server to your personal Linux server using the *~cp* command.

### 3.3 Building, Compiling, and installation

* Use Make all command to compile entire program. Then use the executable generated to run the program
* Use Make test command to compile a mock test program. Then use the executable generated to run the program

## 

## **4. Documentation of packages, modules, interfaces**

### **4.1 Detailed description of data structures**

#### CLIENT:

* + String: Username- Stores username received from user
  + String: Password- Stores password received from user
  + String Security- Stores encrypted password
  + String: Message- Stores message that client intends to send, along with an indicator for destination
  + int: ServSocketFD- socket file descriptor(handles input and output) for service
  + int: DataSocketFD- socket file descriptor(handles input and output) for data
  + int: PortNo- Holds port number to connect with
  + int: ServerAddress- server address(this host)
  + int: ClientAddress- client address we connect with
  + String: RecvBuf - message buffer for receiving a message from server
  + String: SendBuf - message buffer for sending a response to server

#### SERVER

* + String: MessegeRecieve- Holds the message that was last sent from the client
  + String: MessageStore- Holds a message from server before it sends it to the other
  + String: OnlineUsers- A string that contains the name of all users who have existing sockets separated by \n characters and updated whenever they open a socket or exit their socket
  + Text file: Friends- A file inside the server containing all friend usernames in with a \n between each name
  + Text file: Login.txt- A file inside the server containing a string with all registered usernames separated by a \n the respective user’s encrypted password and another \n character.

### 4.2 Detailed description of functions and parameters

* Function prototypes and brief explanation

#### **Client**

##### **Void Encrypt(char\*);**

* + //Takes an input of password then modifies all characters by adding 1 to its ascii value

##### **Void ClientCmp(char\* ServerOut);**

* + //ServerOut is the result of ServerLogin or ServerSignUp
  + //strcmp ServerOut with “Wrong Password\n”, “Username already exists\n” and “Username does not exist\n”
    - //If either is true output ServerOut as message (call GUIPrintError(ServerOut);)
    - //if both are false display contacts saved in ServerOut (call GUISuccessLogin(ServerOut);)

#### **Server**

##### **char\* ServerLogin(char\* username, char\* password);**

* + //strcmp every other line of Login.txt to contents of username and if it finds a match it does strcmp to the line below with password.
    - //If password doesn't match it outputs “Wrong Password\n”.
    - //If no username matches output “Username does not exist\n”.
    - //if both username and password comparisons result to true output username then every other line.
      * //The client username is output followed by every username all output in a single string.

##### **char\* ServerSignUp(char\* username, char\* password);**

* + //strcmp every other line of Login.txt to contents of username
    - //If a username matches, output “Username already exists\n”.
    - //if both username and password comparisons result to true. output username.

##### **Void FatalError(const char \*Program, const char \*ErrorMsg)**

* + Prints an error message if there is a problem connecting to the client,
  + Parameters are arguments in the command line, and an error message to output.

##### **char\* ServerMessageTransfer(char\* message, char\* recipientUsername)**

* + Transfers message from the sender to the recipient
    - Reads in the message from the sender client and writes out the message to the recipient client

#### **GUI**

##### **Char \*GetLogin();**

* + //This will get the ascii characters from the GUI input for username

##### **Char\* GetPwd();**

* + //This will get the ascii characters from the GUI input for password

##### **Void GUIPrintError();**

* + //This will print an error prompt onto the GUI

##### **Int GUISuccessLogin();**

* + //This will load the first screen which is the contacts screen with a default conversation (conversation of the first contact in the contacts list)

##### **char\* GUILoadConversation(char \*sender, char\* friend);**

* + //This will get the conversation from the server by requesting the username

##### **Int GUIDrawConversation(char\* conv);**

* + //This will translate the ASCII string of conversation on text and dialogs on the GUI.

### **4.3** D**etailed description of the communication protocol**

#### **Explanation of communication related function calls**

##### **Char \* ReceiveMessage();**

* + Receives messages from its complement (server receives messages from client and vice versa)
  + If there is message pending from server to client, this will return the message to the corresponding client in a string data type.

##### **Void \*SendMessages(char\* Message);**

* + Send messages to its complement (client sends messages to serve and vice versa)
  + Takes as parameter the message to be sent.

#### **Explanation of communication protocol flags/settings**

The following flags are used for strings sent between Server and Client, they will be useful to know what each of the strings being sent represent and what they should do.

* Messaging Flags:
* “**F:** /USERNAME/” → tells server/client the following message is from USERNAME
* “**T:** /USERNAME/” → tells server/client the following message is to USERNAME
* “**M:** /MESSAGE/” → Tells server/client that the message being sent is the string MESSAGE.
* “**R:** /” -> Receive message
* Client→ Server Flags:
* “**LUser:** /LOGINUSER/ **Pwd:** /PASSWORD/” → tells server that it is receiving a logging in package, where LOGINUSER is the username and PASSWORD is its corresponding encrypted password.
* “**SUser:** /SIGNUPUSER/ **Pwd:** /PASSWORD/” → tells server that it is receiving a Signing up package, where SIGNUPUSER is the username and PASSWORD is its corresponding encrypted password.
* “**Del:** /USERNAME/” → tells server that the client “sender” wants to remove USERNAME from its contacts list.
* “**Add:** /USERNAME/” → Tells server that client “sender” wants to add USERNAME to its contacts list.
* Server→ Client Flags:
* “**Err:** /Message/” → tells client to skip an action and instead report message to user
* “**A:** /ACTIVECONTACTS/” → tells client who is the “sender”’s online contacts
* “**C:** /USER/ /CONTACTS/” → tells client who is “sender” and who are the “sender”’s contacts.
* Server Settings:
* “**Cport:** /PORTNUMBER/” → If the server gets this string, it will now that we want to change its current port to the port given as PORTNUMBER.

## 5. Development plan and timeline

### 5.1 Partitioning of tasks

We decided to partitioning the project as follows, please look below Figure 5 for information regarding each task:

**Figure 6 :Partitioning of tasks**

|  |  |
| --- | --- |
| **Partition Name** | **N. of People** |
| Server Module (Main) | 2 |
| GUI development | 2 |
| Client Module (Main) | 2 |

#### **GUI development:**

GUI.c and GUI.h: This GUI.c contains all the necessary function needed to build the GUI for Instant Messaging.

#### **Server Module:**

Server.c: Contains necessary functions to store, process and send and receive messages with the client module.

#### **Client Module:**

Client.c: Contains necessary functions to login (with encrypted passwords), send and receive messages, and also send and receive data to and from the GUI.

### 5.2 Team member responsibilities

Please refer to **Section** **5.1** for a detailed description of the following responsibilities listed on Figure 6..

**FIgure 6: Team member responsibilities**

|  |  |
| --- | --- |
| **Member Name** | **Responsibility** |
| Avik Banerjee | Server (Database) |
| Bert Yu | GUI |
| Bibek Adhikari | GUI |
| Eric Rodriguez | Client |
| Guy Darel | Client |
| Jose Fregoso | Server |

**Back matter**

# **Copyright**

* Message4U © 2018
* This piece of Software is the sole property of Message4U©. You, the end user, have been given permission to use the software.
* Some icons were obtained from <https://www.flaticon.com/> as a free for use license.

### Error messages

* Username not found.
* Password not recognized.
* During Registration:
  + Username already taken
* Error connecting to server.
  + No Wifi
  + Invalid port
* Note to the End user: This software has been written by students, therefore, we are not responsible for any glitches, or other errors that may occur.

### Index

Active Status…………………………………………………………………………………………..2

Bidirectional…………………………………………………………………………………………...5

Bind()…………………………………………………………………………………………………..6

Chat history feature…………………………………………………………………………………..3

Client…………………………………………………………………………………………………..7

Char……………………………………………………………………………………………………9

Connect()………………………………………………………………………………………………2

Idle Status……………………………………………………………………………………………..14

IP Address……………………………………………………………………………………………..17

Instant Messaging Feature…………………………………………………………………………..13

Internet…………………………………………………………………………………………………14

Int……………………………………………………………………………………………………….18

Listen()………………………………………………………………………………………………….6

Offline Status…………………………………………………………………………………………..9

Group Messaging Feature…………………………………………………………………………...13

Port Number…………………………………………………………………………………………...16

Point-to-point communication………………………………………………………………………...15

Read()…………………………………………………………………………………………………..4

Server……………………………………………………………………………………………………3

Sockets………………………………………………………………………………………………….7

Socket()………………………………………………………………………………………………....9

Struct…………………………………………………………………………………………………….11

Write()...............................................................................................................................5