ZotChat

Developer Manual v1.0
By: ZotMeUp© (Team 7)

Development Team:
Anson Do
Arian Reyes
Xianzhang Li
Adrian Gomez
Kevin Huang

Affiliation:
University of California, Irvine
Department of Electrical Engineering and Computer Science

Table of Contents

Glossary

C

Chat - talking

Chess - a 2 player board game that is won by capturing the other player's king

Client - the person who is obtaining information from the server

clientAddress - the port number of the client

E

Encrypt - changing the password to be stored differently so it won't be easily obtained

F

Friend - theoretically, someone who you can message with

G

GUI - the graphic user interface that users interact with

I

Instant Messaging - sending and receiving messages to and from other users

M

Message - a string of text that is being sent to another user

P

Password - a code that is used to access your account

Port Number - a 16-bit unsigned number from 0 to 65535

R

Rbuffer - the message being received

S

Sbuffer - the message being sent

Server - provides the service to clients

serverAddress - port number of the server

Socket - a network that connects two or more clients

U

Username - a name that is used to access your account

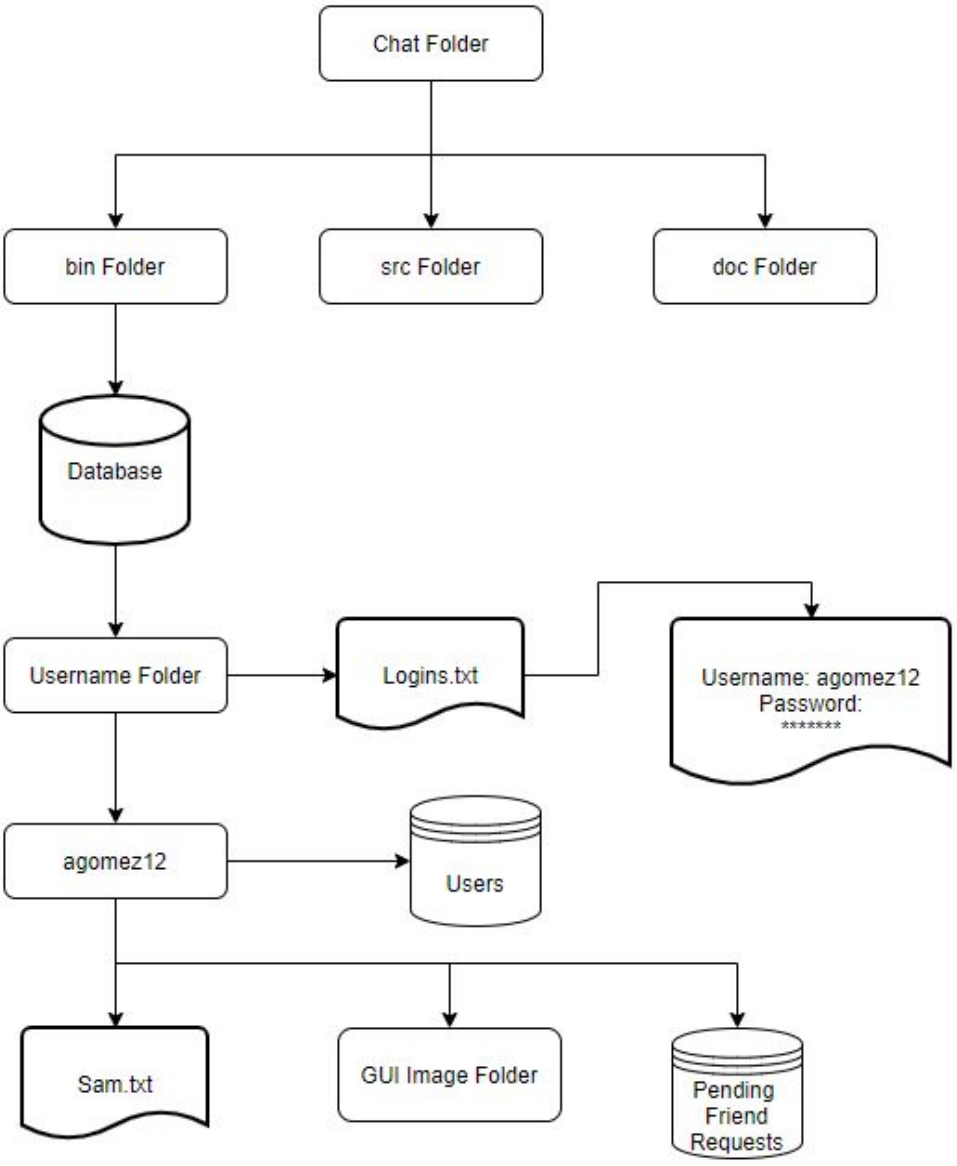
1. Client Software Architecture Overview

1.1 Main Data Types and Structures

Structure Name	Data Type
username	String of chars (char[x])
password	String of chars (char[x])
message	String of chars (char[x])

Socket Name	Data Type	Explanation
serverSocketFD	int	I/O for Service
dataSocketFD	int	I/O for Data
portNum	int	Port to connect to
serverAddress	int	Server Address
clientAddress	int	Client we connect to
Rbuffer[256]	String of chars (char[x])	Buffer for receiving a message
Sbuffer[256]	String of chars (char[x])	Buffer for sending a message

1.2 Major Software Components



1.3 Module Interfaces

- *Graphical User Interface (GUI)* : Contains the images that are being used for the GUI. This includes the “Login” Screen, “Sign-Up” Screen, “Messaging” Screen and etc.
- *Client* : Contains the functions responsible for logging in, sending/ receiving messages, and sending/receiving data from the GUI and server. These processes will be encrypted for the safety of the user.
- *Server* : Contains the functions responsible for data management from the client function. This includes sending and receiving messages, storing messages, and processing messages.

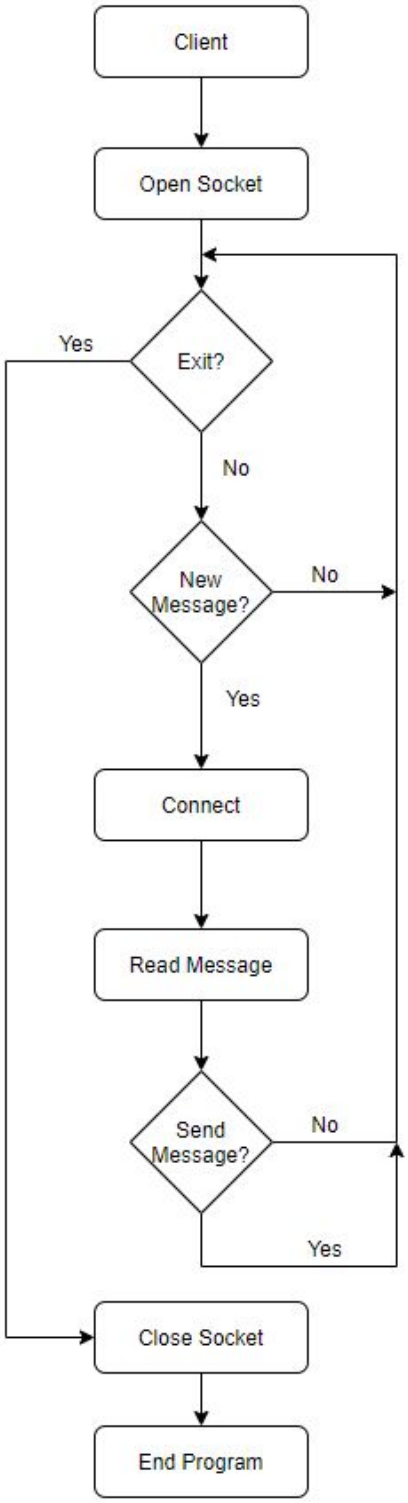
1.4 Overall Program Control Flow

Depending on whether the user is a new or a returning user, the program will adjust accordingly. If the user is new, then the user will be sent to another screen to create the credentials that would be sent and stored in the server.

If the user is already an existing user, then the information provided will be compared to the one stored on the server. This process will either return a “True” or “False” depending on the comparison of the information.

Once the user is able to log in, this will allow the user to be connected to the server. When the user has received a message, then they will be connected in order to view the message from the GUI. If the user wants to send a message, then it will read the conversation and push the message from the GUI to the server. The main user will be in a constant state of waiting for a message until it times out after a certain period of time.

Diagram of Previous paragraphs



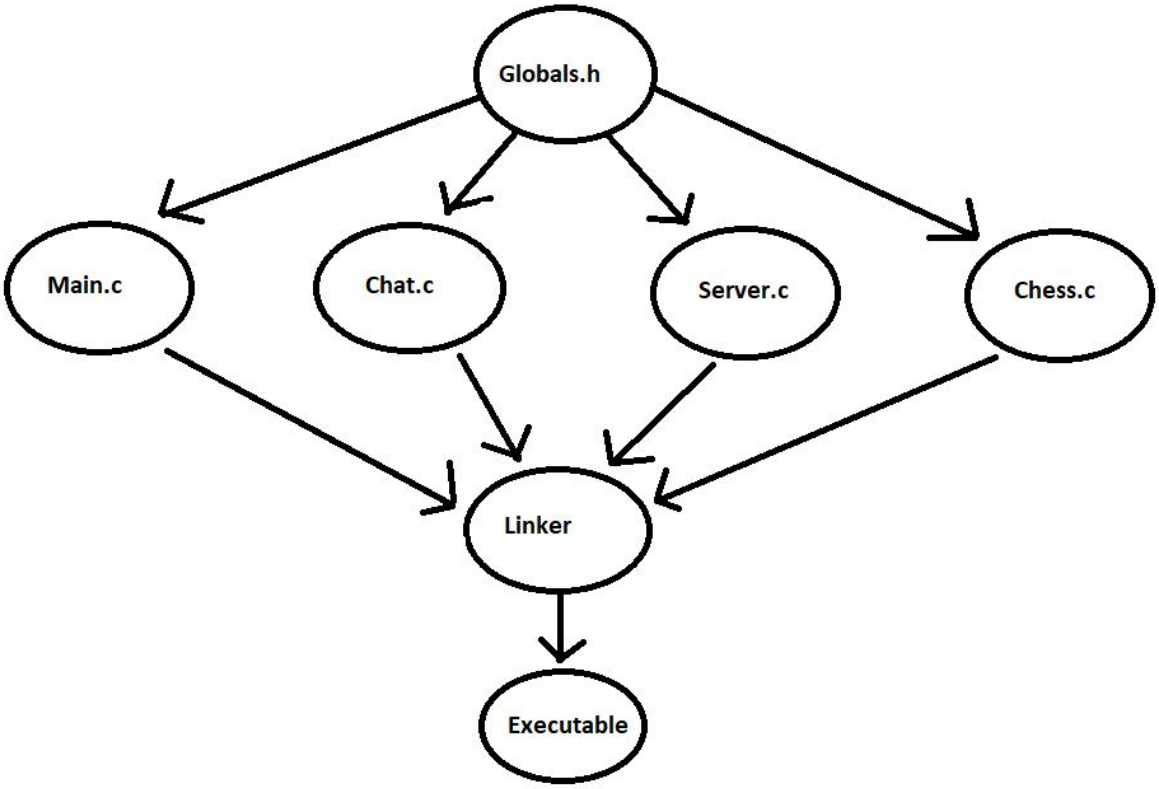
2. Server Software Architecture Overview

2.1 Main Data Types and Structures

Text file: Friends.txt
Text file: Login.txt
String Array: Save all the message

2.2 Major Software Components

/* Diagram of module hierarchy*/

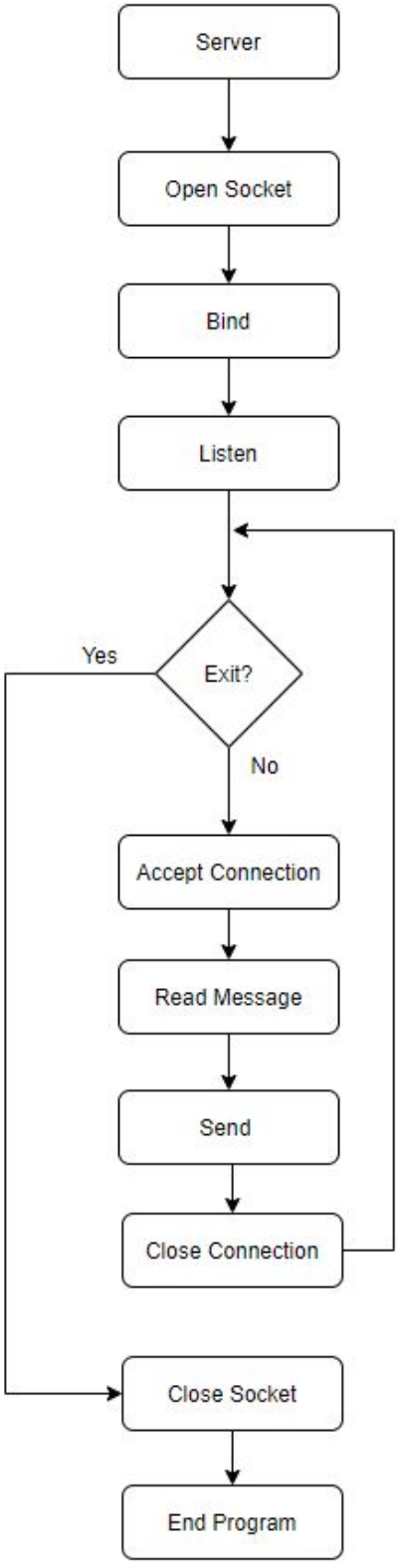


2.3 Module Interfaces

API of major module functions

- Server: This will attempt to connect the program to the server it needs to be connected to be able communicate through the internet
- Chess: This will be the chess implementation in the chat window
- Chat: This contains the program to make the chat function work within our program
- Main: This will be where all the other modules will be compiled into and run through the main function this will also include all the necessary global variables and will hold the GUI creation.

2.4 Overall Program Control Flow



3. Installation

3.1 System Requirements, Compatibility

System:	Linux
Disk Space:	128MB free
Ram:	512 MB or more
CPU:	1.2GHz or higher
Internet:	essential for the chat application

3.2 Setup and configuration

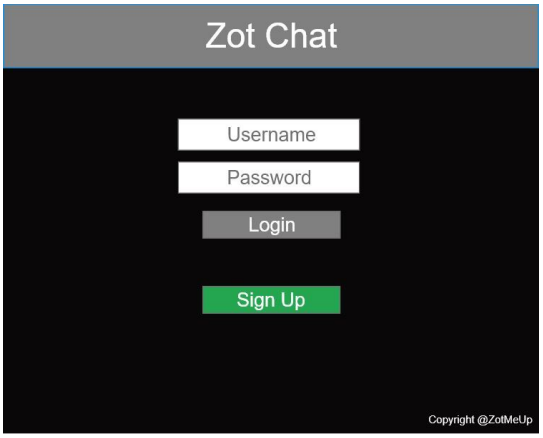
To install the chat app, copy (using ~cp) the chat_V1.0_src.tar.gz and chat_V1.0_src.tar.gz file from the host to your own Linx.
Type make chat and make test command which can run the chat client.

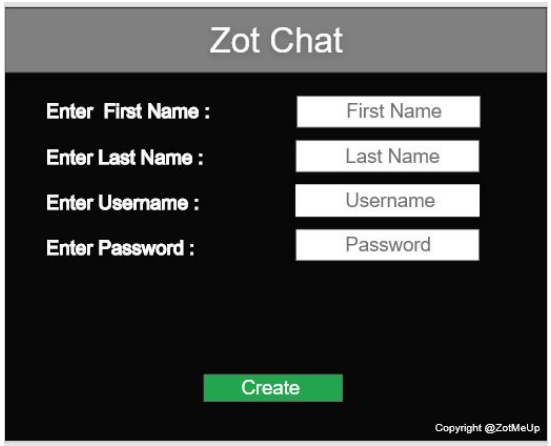
3.3 Building, Compilation, Installation

Type make all to compile every portion of the program, then run the executable that was made from the make all.
As stated in 3.2 as well you can also type make test to auto compile everything and run the executable with just that single command.

4. Documentation of Packages, Modules, Interfaces

4.1 Detailed description of data structures





Chat Login - Client

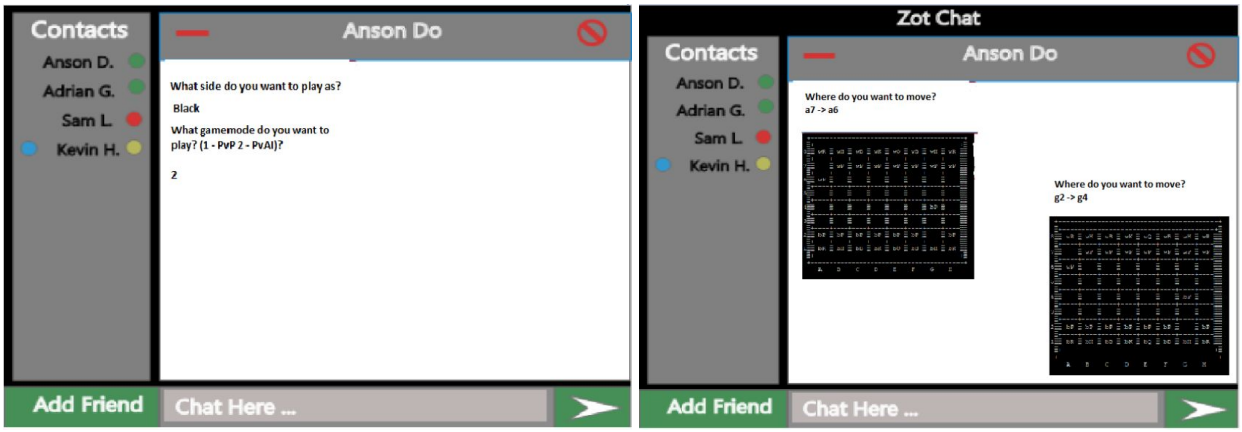
Client Logging In or Create Log in:

1) username
    a) Data Type: string
    b) Functionality: Stores username inserted by the user
2) password
    a) Data Type: string
    b) Functionality: Stores password inserted by the user
3) firstName
    a) Data Type: string
    b) Functionality: Stores first name inserted by the user

4) lastName

    a) Data type: string

    b) Functionality: Stores last name inserted by the user



Client/Server Chatroom:

Client:

1) message

    a) Data Type: string

    b) Functionality: Stores the message the client wants to send.

2) serviceSocketFD

    a) Data Type: int

    b) Functionality: Socket file description for service. Handles the input and output from client and server.

3) dataSocketFD

    a) Data Type: int

    b) Functionality: Socket file description for data. Handles the input and output from client and server.

4) portNum
    a) Data Type: int
    b) Functionality: Holds the port number the client and server will use in order to connect.
5) serverAddress
    a) Data Type: int
    b) Functionality: Location of the server. Since, the server is the host will send the location to the client.
6) clientAddress
    a) Data Type: int
    b) Functionality: Has a copy of the address of the server the client will connect to.
7) Rbuffer[256]
    a) Data Type: string
    b) Functionality: Message buffer for receiving a message from the server.
8) Sbuffer[256]
    a) Data Type: string
    b) Functionality: Message buffer for sending a response to the server.

Server:

1) messageReceive
    a) Data Type: string
    b) Functionality: Holds a message that was last sent from the client.
2) messageStore
    a) Data Type: string
    b) Functionality: Holds a message from the server before it sends it to the other client.
3) usersConnected
    a) Data Type: string
    b) Functionality: Has the list of all the users who have an existing socket. If users exit the socket or open a socket then the list will be updated.
4) login
    a) Implemented in Text File
    b) Functionality: Has a list of all the existing logins.
    c) Additional Information: If a user creates an account then the list will be updated.
5) friendList
    a) Implemented in Text File
    b) Functionality: Has a list of all the friends the user has.

c) Additional Information: List will be updated when the user adds or removes a friend from the contact list.

4.2 Detailed description of functions and parameters .

1) Client Functions
    a) void encrypt(char* password);
        i) Takes in the clients password and encrypts it.
    b) void client();
        i) Takes
2) Server Functions for Login:
    a) char* serverLogin(char* username, char* password);
        i) Takes in username and password of the user and compares it to their existing list to see if it finds a match.
            (1) If incorrect information is given, a wrong username and/or password is given.
    b) char* serverCreateAccount(char* username, char* password, char* firstName, char* lastName);
        i) Stores information and is updated onto the login list.
3) Server Functions for Chat:
    a) char messageTransport(char* message, char *user);

i) Used to send the message to the corresponding client.

b) void errorMessage(char* message, char *user);

i) Error in sending the messages to the client.

ii) A message error will be printed out.

c) void errorConnecting(char* connection, char* errorMessage);

i) Prints an error message when there has been an error connecting to the client.

4) GUI Functions:

a) char* addFriend(char *username);

i) When adding a friend they will be implemented in the list of friends for that user.

b) char* removeFriend(char *username);

i) When removing a friend they will be eliminated from the list of friends for that user.

c) char* getLogin();

i) It takes the username and password of the user.

ii) If valid, then switches the screen to the chat room.

iii) If invalid, executes loginError()

d) char* getUsername();

i) Gets username to pull up a specific users platform.

ii) Each user has their own friends, conversations, and games.

iii) Username is obtained through ASCII characters.

e) char* getPassword();

i) Password is obtained through ASCII characters.

f) void loginError();

i) Prints login error on GUI.

g) char* loadConversation(char* sender, char* friend);

i) Retrieve conversation from the server and display it onto the GUI.

ii) Will retrieve using username of the user.

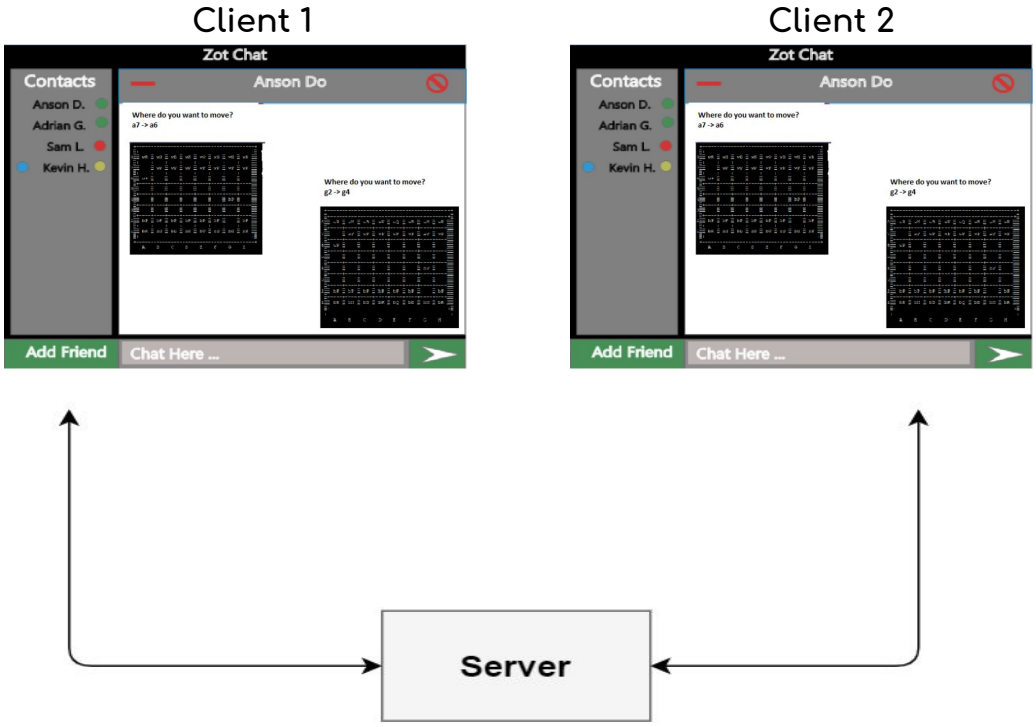
h) char* loadBoard(char* sender, char* friend);

i) Retrieve board moves from the server and display the board movements onto the GUI.

ii) Will retrieve using username of the user.

4.3 Detailed description of the communication protocol

Schematics of communication between multiple clients and server:



Main Idea: The server will be used to communicate with one client at a time.

Functions used for client and server communication:

1) Char *receiveMessage();
    a) Server is receiving the messages from the clients.
    b) Server will only receive a message if it's done receiving the message from the other client or if it's unoccupied.
    c) If a message is pending from server to client, then will return message to corresponding client.
    d) One message can be handled at a time.

2) void sendMessage(char* message);
    a) Takes in the message from one of the clients, sends it to the server and the server will send it to the corresponding client.

Can categorize the type of messages that would be sent from client to server.

5. Development Plan and Timeline

5.1 Partitioning of tasks

Features	Week 7	Week 8	Week 9	Week 10
Chat	X	WIP	WIP	FINISH
Server	X	WIP	WIP	FINISH
Chess	X	WIP	WIP	FINISH
Main	X	WIP	WIP	FINISH

5.2 Team member responsibilities

Anson Do: Chess Implementation, Main, Manuals, Debugging
Adrian Gomez: Chat GUI, Server/ Client Communication
Kevin Huang: Server/Client Communication, Password Encryption
Xianzhang Li: Chess client and server Communication
Arian Reyes: Chat GUI, Server/Client Communication

Copyright

- ZotMeUp© 2020
- Stock Messaging Icon from https://www.iconfinder.com/icons/171351/chat_messages_icon
-

ERROR MESSAGES

Error	Meaning	How to Resolve
100	Wrong username	Try to type a valid username
101	Wrong password	Try to type the correct password
102	Could not connect to server	Try to make sure you have the correct IP
103	Username taken	Try another username

INDEX