Assignment 1 - Programming Assignment 1 A Secure and Robust Chat System in GoLang and NodeJS

CPS 475/575 - Secure Application Development
Instructor: Dr. Phu Phung

Student name: Karthik Nimma
Student ID: 101650589
Email: nimmak1@udayton.edu

Introduction

[karthiknimma / secad-nimmak1 / Labs / Assignment 1 — Bitbucket]

In this assignment we have created a Chat Server and a Chat client.Each client can communicate with other clients in the chat system. The server is developed using go lang whereas the client is developed using nodejs. The server has the functionality to accept multiple clients requests simultaneously. This is thanks to the goroutines. Goroutines are light weight threads which allows the server to multitask. Each client has to be authenticated with user credentials before it can enter the chat system. The server stores the user credentials which it uses to authenticate the clients. Once the user has been logged, the user can communicate with other clients either privately or send a message to the public chat. Besides that, a user can also request a list of all current logged in users. All communications between the server and client is encoded in json format. This allows easy retrieval of relevant information. In order to avoid any data race scenarios each variable is accessed only through channels this ensures only one go routine can access a variable and others have to wait. Robust and defensive programming principles were implemented throughout the development process.For every input scenario the program prints out a message to inform the user about valid/invalid inputs. Data is validated before it is assigned to a variable or stored in a buffer.

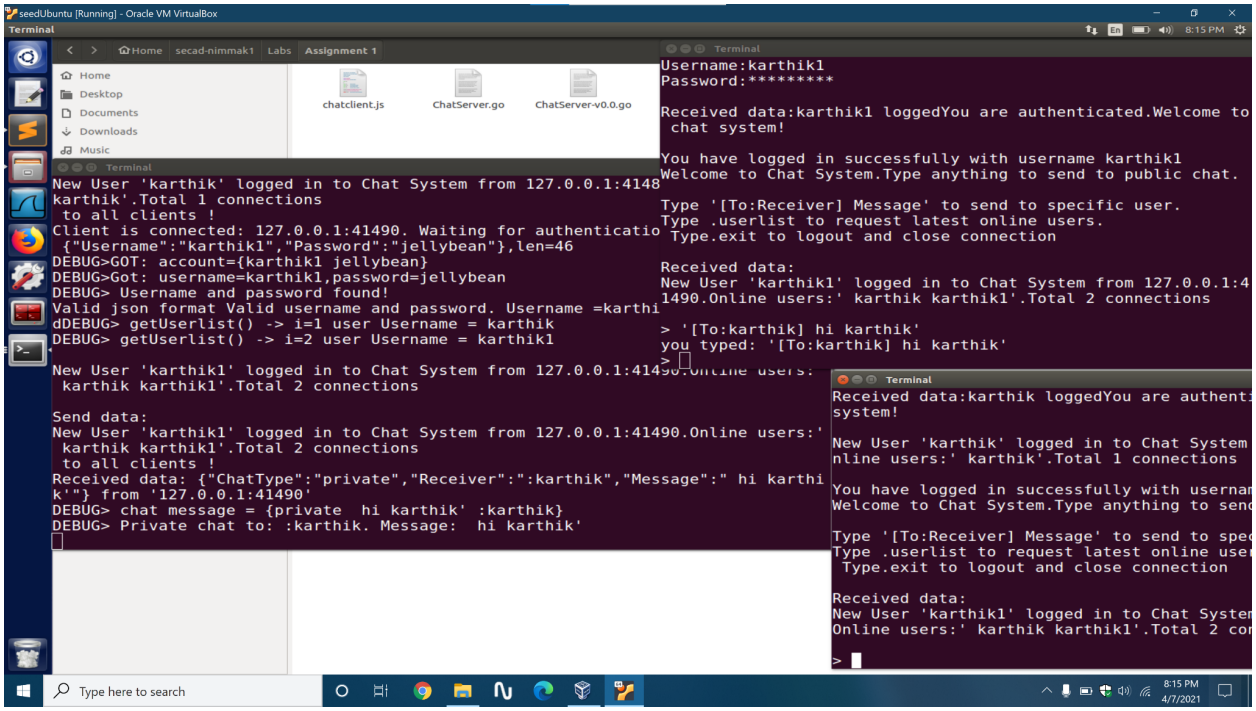
During the development, the main problem was lack of compile errors as the majority of errors showed up at runtime. Having print statements in almost every function helped to find the exact location of the bugs. Also while passing the values between functions it was important to typecast variables to respective types to ensure everything works correctly.

Design

The chat Server was developed in go lang as this language offers go routines which are lightweight threads capable of running simultaneously . This is an important feature required by any server. JSON format is used to encode and decode messages on both sides. JSON makes it easy to interpret data. The authentication module is broken down into three functions for simplicity, each function passes data to the other and the final result is then sent back to the client. [ ]byte is used to send and receive data between server and client. Some functions return multiple values and the values are used as needed. Once authentication is completed, the control is handed over to the client_goroutine which keeps on running as long as the connection is established.This function calls the handleMessages function which parses the client message and categorizes the nature of the request as either a command,public message or private message.Messages are implemented using a predefined struct which stores the chatType, message and the receiver's username. Data race issue is avoided by the use of channels which does not allow other goroutines to access a variable while one goroutine is writing to it. The client can request for a userlist anytime and the server will send the latest userList back to that client.

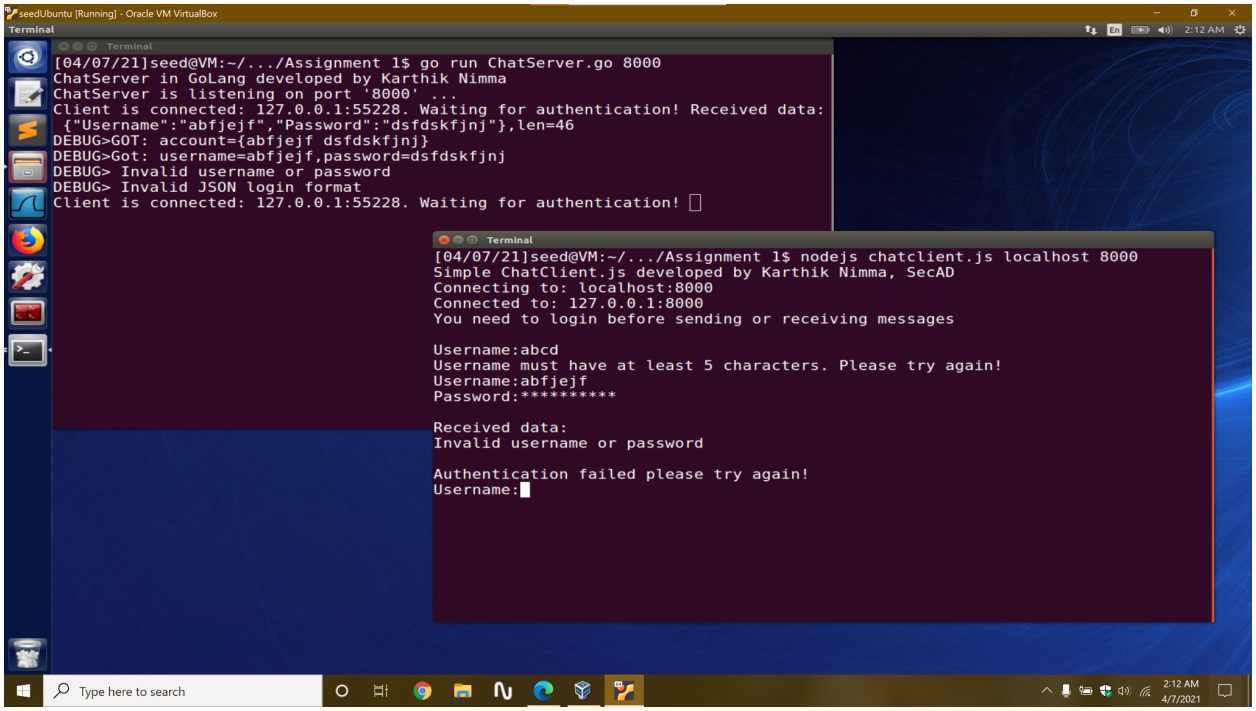
Test Cases

1.



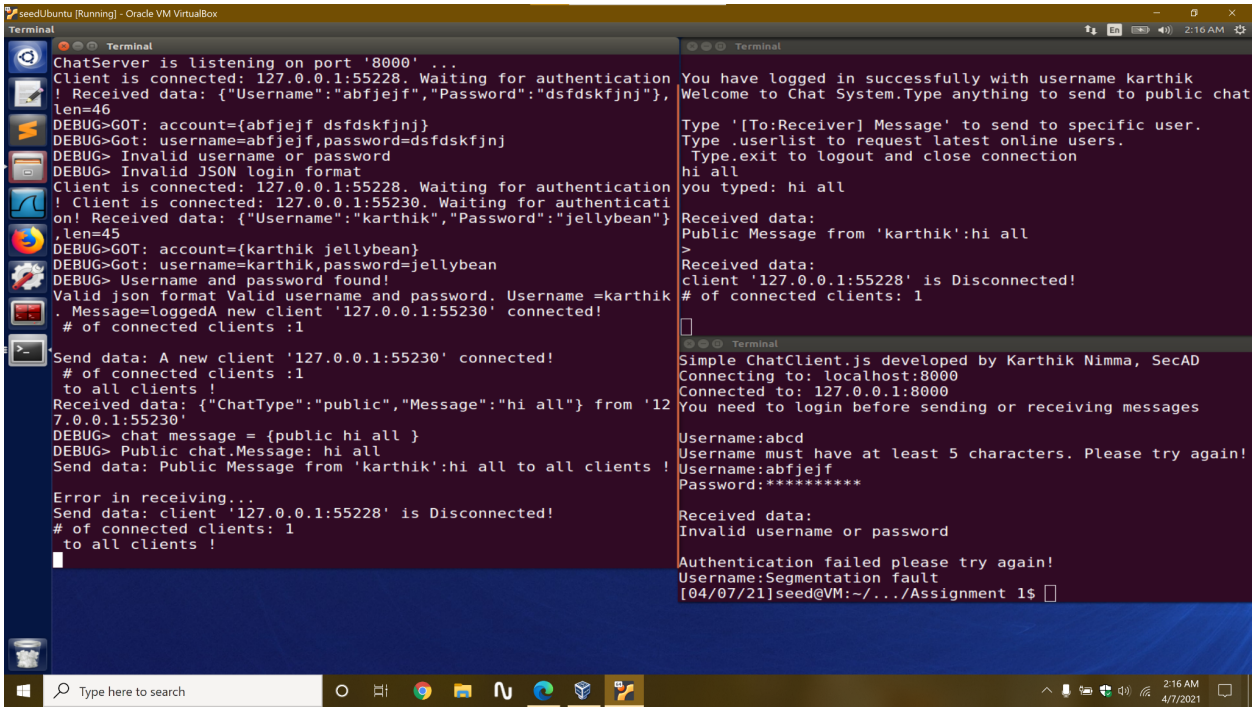
This is a valid login request. The server receives the request in JSON format and parses the login data. It checks the username password using the check account function which returns true if the credentials exist.

2.



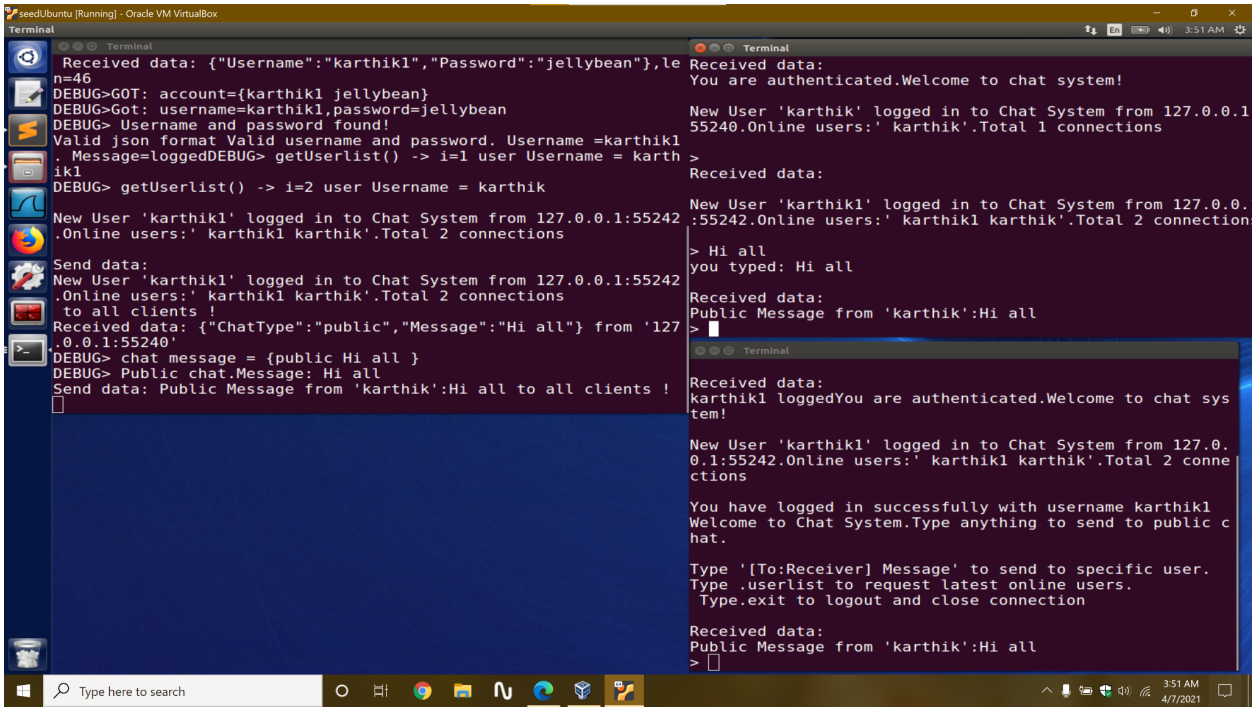
Input validation is done on both sides of the system and an error is printed if input does not follow certain requirements.If the server does not have the login credentials an error message is returned to the client describing the nature of the error.

3.



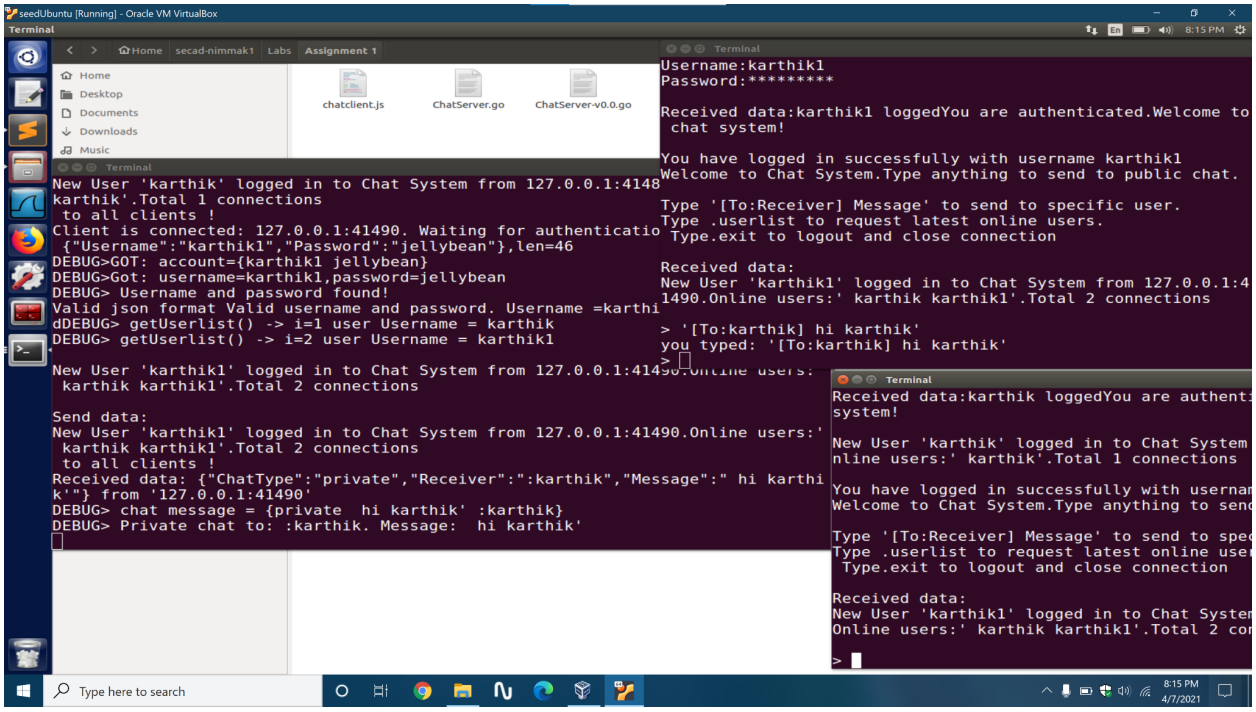
Public messages can be sent and received by all authenticated clients.On the bottom right corner we can see that the client failed authentication and hence hasn't received the public message.

4 & 5.



Everytime a new client connects an updated userlist is sent through public chat to all connected clients.

6.



Private messages can be exchanged between two clients but each client has to follow a protocol to indicate to the server that this is a private chat. The accepted format is shown in the right side terminal of the above screenshot. The server(on the left) then parses the request and retrieves information such as sender , receiver and message. For some reason my server does not forward that message to the client.

Appendix

ChatServer.go

```
/* Simple ChatServer in GoLang by Karthik Nimma for SecAD*/
package main

import (
	"fmt"
	"net"
	"os"
	"encoding/json"

)

const BUFFERSIZE int = 1024

type User struct {
	Username string
	Login bool
	Key string
}

type Command struct{
	Command string //userlist,quit,etc
}

type ChatMessage struct {
	ChatType string //private or public
	Message string
	Receiver string //only available in private chat
}

var authenticated_clients = make(map[net.Conn]string)
var lostclient = make(chan net.Conn)
var newclient = make(chan net.Conn)

// To implement yet.....................
// var currentLoggedUser User
```

```
var allLoggedIn_conns = make(map[net.Conn]interface{})
var usersList = make(map[string]bool)
var message = make(chan string)
var activeUser string
func main() {
	if len(os.Args) != 2 {
		fmt.Printf("Usage: %s <port>\n", os.Args[0])
		os.Exit(0)
	}
	port := os.Args[1]
	if len(port) > 5 {
		fmt.Println("Invalid port value. Try again!")
		os.Exit(1)
	}
	server, err := net.Listen("tcp", ":"+port)
	if err != nil {
		fmt.Printf("Cannot listen on port '" + port + "'!\n")
		os.Exit(2)
	}
	fmt.Println("ChatServer in GoLang developed by Karthik Nimma")
	fmt.Printf("ChatServer is listening on port '%s' ...\n", port)

	//newclient := make(chan net.Conn)
	go func(){
		for{
			client_conn, _ := server.Accept()
			go login(client_conn)
		}
	}()
	for{
		select{
		case client_conn := <- newclient:
			go authenticating(client_conn)
		case client_conn := <- lostclient:
			delete(authenticated_clients,client_conn)
			delete(allLoggedIn_conns,client_conn)
			byemessage := fmt.Sprintf("client '%s' is Disconnected!\n# of
connected clients: %d\n",client_conn.RemoteAddr().String(),len(authenticated_clients))
			go sendtoAll([]byte(byemessage))
		}
```

```
	}
}
func is_authenticated(client_conn net.Conn) bool {
	if allLoggedIn_conns[client_conn] != nil {
		return true
	}
	return false
}

func authenticating(client_conn net.Conn){
	authenticated_clients[client_conn] = client_conn.RemoteAddr().String()
	var newuser User
	newuser.Key = client_conn.RemoteAddr().String()
	newuser.Username = activeUser
	newuser.Login = true
	allLoggedIn_conns[client_conn] = newuser
	sendto(client_conn,[]byte("You are authenticated.Welcome to chat system!\n"))

	// welcomemessage := //combined the two current messages with Sprintf
	//	fmt.Sprintf("A new client '%s' connected! \n # of connected clients
:%d\n",client_conn.RemoteAddr().String(),len(authenticated_clients))
	msg := getUserlist()
	myString := string(msg[:])
	welcomemessage := //combined the two current messages with Sprintf
	fmt.Sprintf("\nNew User '%s' logged in to Chat System from %s.Online
users:'%s'.Total %d
connections\n",newuser.Username,client_conn.RemoteAddr().String(),myString,len(auth
enticated_clients))

	fmt.Println(welcomemessage)
	go sendtoAll([]byte (welcomemessage))
	go client_goroutine(client_conn)
}
func client_goroutine(client_conn net.Conn){

	var buffer [BUFFERSIZE]byte
	go func(){
		for {
			byte_received, read_err := client_conn.Read(buffer[0:])
			if read_err != nil {
```

```
                        fmt.Println("Error in receiving...")
                        lostclient <- client_conn
                        return
                }
                // go sendtoAll(buffer[0:byte_received])
                fmt.Printf("Received data: %s from '%s'\n",
                        buffer[0:byte_received], client_conn.RemoteAddr().String())
                        data := decrypt(client_conn, buffer[0:byte_received])
                        handleMessages(client_conn,data)
        }
    }()
}
func decrypt(client_conn net.Conn, data []byte) []byte {
    return data
}
func encrypt(client_conn net.Conn, data []byte) []byte {
    return data
}


func sendtoAll(data []byte) {
    for client_conn,_ := range authenticated_clients{
        _, write_err := client_conn.Write(data)
        if write_err != nil{
            fmt.Println("Error in sending...")
            continue;
        }
    }
    fmt.Printf("Send data: %s to all clients ! \n",data)
}
func sendto(client_conn net.Conn,data []byte){
    _, write_err := client_conn.Write(data)
    if write_err != nil{
        fmt.Println("Error in sending...")
        return
    }
}

// Assignment 1 authentication
func login(client_conn net.Conn){
```

```
	fmt.Printf("Client is connected: %s. Waiting for authentication!
",client_conn.RemoteAddr().String())
	var buffer [BUFFERSIZE]byte
	byte_received, read_err := client_conn.Read(buffer[0:])
	if read_err != nil {
		fmt.Println("Error in receiving...")
		lostclient <- client_conn
		return
	}
	clientdata := buffer[0:byte_received]
	fmt.Printf("Received data: %s,len=%d \n", clientdata, len(clientdata))

	// if checklogin fails dont add to client list
	checklogin, username, msg2 := checklogin(clientdata)



	if(!checklogin){
		// send error message back to client
		activeUser = ""
		fmt.Println("DEBUG> Invalid JSON login format")
		sendto(client_conn,[]byte(msg2))
		login(client_conn)
	}else{
		// add client to the connected client lists
		fmt.Printf("Valid json format Valid username and password. Username =" +
username + ". Message=" +msg2)
		activeUser = username
		authentication_msg := username+" "+msg2
		sendto(client_conn,[]byte(authentication_msg))
		newclient <- client_conn

		//TRIED TO IMPLEMENT FUNCTIONALITY TO ALLOW USERS TO
LOGIN FROM MULTIPLE DEVICES WITHOUT CAUSING REDUNDANT
USERNAMES
		// IN USERLIST.
		//if searchUser(client_conn,activeUser) {
		//	go client_goroutine(client_conn)
		//}else{
```

```
                //      newclient <- client_conn
                //}
        }
}
func searchUser(client_conn net.Conn,currentUser string) bool{
        for client_conn,_ := range allLoggedIn_conns{
                user := allLoggedIn_conns[client_conn].(User)
                if(user.Username == currentUser){
                        return true
                }
        }
        return false
}
func checklogin(data []byte) (bool,string,string){
        type Account struct{
                Username string
                Password string
        }
        var account Account
        err := json.Unmarshal(data, &account)
        if err!=nil || account.Username == "" || account.Password == ""{
                fmt.Printf("JSON parsing error: %s\n",err)
                return false,"",`[BAD LOGIN] Expected: {"Username":"..","Password":".."}`
        }
        fmt.Printf("DEBUG>GOT: account=%s\n", account)
        fmt.Printf("DEBUG>Got:
username=%s,password=%s\n",account.Username,account.Password)

        accountExists := checkaccount(account.Username,account.Password)
        if accountExists{
                fmt.Println("DEBUG> Username and password found!")
                return true,account.Username,"logged"
        }
        fmt.Println("DEBUG> Invalid username or password")
        return false,"","Invalid username or password\n"
}
func checkaccount(name string,passkey string) (bool){
        if(string(name) == "karthik" && string(passkey) == "jellybean"){
                return true
        }
```

```
	if(string(name) == "karthik1" && string(passkey) == "jellybean"){
		return true
	}
	if(string(name) == "karthik2" && string(passkey) == "jellybean"){
		return true
	}
	if(string(name) == "testuser" && string(passkey) == "testpassword"){
		return true
	}
	return false

}
func handleMessages(client_conn net.Conn, data []byte){
	if !is_authenticated(client_conn){
		sendto(client_conn,[]byte("You havent been authenticated.Message
ignored!"))
		return
	}
	var command Command
	command_err := json.Unmarshal(data, &command)
	if command_err != nil ||  command.Command == ""{
		var chatMessage ChatMessage
		chat_err := json.Unmarshal(data, &chatMessage)
		if chat_err != nil || chatMessage.ChatType == "" {
			fmt.Printf("Unknown data type = %s\n", data)
			errorMessage(client_conn)
			return
		}
		// Chat message
		fmt.Printf("DEBUG> chat message = %s\n", chatMessage)
		if chatMessage.ChatType == "private" {
			privateChat(client_conn,chatMessage)
		} else if chatMessage.ChatType=="public"{
			publicChat(client_conn,chatMessage)
		}
	}else{
		commandResponse(client_conn, command)
	}
}
```

```
func commandResponse(client_conn net.Conn, command Command){
	fmt.Printf("DEBUG> Command = %s\n", command.Command)
	switch command.Command{
		case "UserList":
			fmt.Printf("DEBUG>get Userlist and return\n")
			userList := getUserlist()
			fmt.Printf("DEBUG> userListJSON: %s\n", userList)
			sendto(client_conn,userList)
		case "Quit":
			fmt.Printf("DEBUG> Client quited!\n")
			lostclient <- client_conn
	}
}
func errorMessage(client_conn net.Conn){
	sendto(client_conn, []byte("[BAD DATA: Unknown command or Data format "))
	expected_format := fmt.Sprintf(`Expected format: {"Command":"Quit"} |
{"Command":"Userlist"} | {"ChatType":"private","Receiver":"..","Message":".."}`)
	sendto(client_conn,[]byte(expected_format))
}
func privateChat(client_conn net.Conn, chatMessage ChatMessage){
	fmt.Printf("DEBUG> Private chat to: %s. Message: %s\n",
chatMessage.Receiver, chatMessage.Message)
	if allLoggedIn_conns[client_conn] != nil{
		user := allLoggedIn_conns[client_conn].(User)
		message := fmt.Sprintf("Private message from
'"+user.Username+"':%s",chatMessage.Message)
		for receiver_client_conn , _ := range allLoggedIn_conns {
			if allLoggedIn_conns[receiver_client_conn] != nil {
				receiverUser :=
allLoggedIn_conns[receiver_client_conn].(User)
				// one user maybe logged in from different devices
				if receiverUser.Username == chatMessage.Receiver{
					sendto(receiver_client_conn, []byte(message))
				}
			}
		}
	}
}

func publicChat(client_conn net.Conn, chatMessage ChatMessage){
```

```
		fmt.Printf("DEBUG> Public chat.Message: %s\n", chatMessage.Message)
		if allLoggedIn_conns[client_conn] != nil{
			user := allLoggedIn_conns[client_conn].(User)
			message := fmt.Sprintf("Public Message from '"+user.Username+"':%s",
chatMessage.Message)
			sendtoAll([]byte(message))
		}
}
func getUserlist() []byte{
		//userList := []string{}
		var userList []string
		debug_len :=0
		for _, usereach := range allLoggedIn_conns {
			debug_len = debug_len + 1
			var usereach1 User = usereach.(User)
			fmt.Printf("DEBUG> getUserlist() -> i=%d user Username =
%s\n",debug_len,usereach1.Username)
			userList = append(userList," "+usereach1.Username)
		}
  // store to byte array
		// var str = []string{"str1","str2"}
		var x = []byte{}

		for i:=0; i<len(userList); i++{
		    b := []byte(userList[i])
		    for j:=0; j<len(b); j++{
		        x = append(x,b[j])
		    }
		}
		return x
}
```

ChatClient.js

```
var net = require('net');

if(process.argv.length != 4){
	console.log("Usage: node %s <host> <port>", process.argv[1]);
	process.exit(1);
}

var host=process.argv[2];
var port=process.argv[3];
// var host = 'localhost'
// var port = '8000'
authenticated = false;


if(host.length >253 || port.length >5 ){
	console.log("Invalid host or port. Try again!\nUsage: node %s <port>",
process.argv[1]);
	process.exit(1);
}

var client = new net.Socket();
console.log("Simple ChatClient.js developed by Karthik Nimma, SecAD");
console.log("Connecting to: %s:%s", host, port);

client.connect(port,host, connected);

function connected(){
	console.log("Connected to: %s:%s", client.remoteAddress, client.remotePort);
	console.log("You need to login before sending or receiving messages\n")
	loginsync();

}
var readlineSync = require('readline-sync');
var username;
var password;
function loginsync(){
	username = readlineSync.question('Username:');
```

```
	if(!inputValidated(username)){
		console.log("Username must have at least 5 characters. Please try
again!");
		loginsync();
		return;
	}
	// handle the secret password by masking
	password = readlineSync.question('Password:',{
		hideEchoBack: true
	});
	if(!inputValidated(password)){
		console.log("Password must have atleast 5 characters.Try again!")
		loginsync();
		return
	}

	var login = '{"Username":"' + username + '","Password":"'+password+'"}';
	client.write(login)
}
function inputValidated(logindata){
	if(logindata.length >= 5){
		return true;
	}
	return false;
}

client.on("data",data => {
	console.log("\nReceived data:" + data);
	if(!authenticated){
		if(username && data.toString().includes("logged")){
		// if(username && data.includes(username+" logged")){
			console.log("You have logged in successfully with username " +
username);
			authenticated = true;
			startchat();
		}else{
			console.log("Authentication failed please try again!");
			loginsync();
		}
	}
```

```
});

client.on("error",function(data) {
	console.log("Error");
	process.exit(2);
});
client.on("close",function(data){
	console.log("Connection has been disconnected");
	process.exit(3);
});

function startchat(){
	var rl = require('readline')
	var keyboard = rl.createInterface({
		input: process.stdin,
		output: process.stdout
	});
	console.log("Welcome to Chat System.Type anything to send to public chat.\n");
	console.log("Type '[To:Receiver] Message' to send to specific user.");
	console.log("Type .userlist to request latest online users.\n Type.exit to logout
and close connection");

	keyboard.on('line',(input) => {
		console.log(`you typed: ${input}`);
		if(input === ".exit"){
			client.write('{"Command":"Quit"}');
			setTimeout(()=>{
				client.destroy();
				console.log("disconnected!");
				process.exit();},1);
		}else if(input === ".userlist"){
			client.write('{"Command":"UserList"}');
		}else if(input.includes("[To:")){
			endname = input.search("]");
			receiver = input.substring(4,endname);
			if((endname<0) || (receiver == "" || undefined)){
				console.log("unknown receiver.Try again!\n");
				return;
			}else{
```

```
client.write('{"ChatType":"private","Receiver":"'+receiver+'","Message":"'+input.substring(
endname+1,input.length)+'"}');
                        }
                }else
                        client.write('{"ChatType":"public","Message":"'+input+'"}');
        });
}
```