****

**MINI PROJECT REPORT**

**ON**

“Chat Room And Forum Client”

Submitted in

Partial Fulfillment of requirements for the Award of Degree

Of

Bachelor of Technology

In

Computer Science Engineering

By

(Group Number: 20)

Siddharth Singh (1816410279)

Shreyam kanaujiya (1816410265)

Sidharth Singh (1816410279)

Under the supervision of

**RAJAT VERMA**

**(Assistant Professor, CSE Department)**

* **Objective**

The main objective of the Chat Room project is to create a chat application which helps different users to communicate with each other through a server connected. This is a simple chat program with or without a server and can have many clients. The server needs to be started first and clients can be connected later. Simple Chat Room provides a bidirectional communication between client and server. It enables user to seamlessly communicate with each other. The user has an option to login to the chat room. The user should be able to give the ip address of the server and the port at which he is connecting. The user can chat using this chat application. The server shall poll for other users that are active in the chat room and make those users visible. If the user at the other end is active then they can start a chat session. The user can save the chat transcript or clear it based on his requirement.

The project aims to fulfillthe following objectives –

1. Improve communication between individuals in a corporation .
2. Reduce communication time between users .
3. User friendly design .

* **Introduction**

Earlier there was no mode of online communication between users. In big or small organizations communication between users posed a challenge. There was a requirement to record these communications and store the data for further evaluation. The idea is to automate the existing Simple Chat Room system and make the users to utilize the software so that their valuable information is stored digitally and can be retrieve for further management purposes. There was no online method of communicating to different users. Method of using sockets to communicate between nodes would be fast and reliable. It can also be used in online forums such as XDA , cs.rin.ru ,etc.

* **Feasibility Study**

1. The project is easy to maintain .
2. It’s a cost effective way to communicate between people.
3. It’s legally feasible as it uses open source libraries and software.
4. It does not require a high end machine and can even run on low end machines such as a raspberry pi’s .

* **Literature Review**

The term **chat room**, or **chatroom**, is primarily used to describe any form of synchronous conferencing, occasionally even asynchronous conferencing .The first chat system was used by the U.S. Government in 1971. It was developed by Murray Turoff, a young PhD graduate from Berkeley, and its first use was during President Nixon's wage-price freeze under Project Delphi. The system was called EMISARI and would allow 10 regional offices to link together in a real-time online chat known as the party line. It was in use up until 1986. The first public online chat system was called Talkomatic, created by Doug Brown and David R. Woolley in 1973 on the PLATO System at the University of Illinois. It offered several channels, each of which could accommodate up to five people, with messages appearing on all users' screens character-by-character as they were typed. Talkomatic was very popular among PLATO users into the mid-1980s. In 2014 Brown and Woolley released a web-based version of Talkomatic. The first dedicated online chat service that was widely available to the public was the CompuServe CB Simulator in 1980, created by CompuServe executive Alexander "Sandy" Trevor in Columbus, Ohio. Chat rooms gained mainstream popularity with AOL .

* **Hardware/Software Required**

1. **Software required –**
2. Python
3. Any spreadsheet application
4. **Hardware Required –**
5. Standalone system
6. **Technologies –**
7. Tkinter for GUI
8. SOCKETS for establishing connection
9. Threading, random, math for other purposes
10. **System Requirements –**
11. Any standalone computer
12. Secondary memory for storage of data
13. Latest version of python libraries
14. Any type of OS ( Windows, Linux , Mac OS, etc) running python.

* **Coding :**

import sys

if not sys.hexversion > 0x03000000:

version = 2

else:

version = 3

if len(sys.argv) > 1 and sys.argv[1] == "-cli":

print("Starting command line chat")

isCLI = True

else:

isCLI = False

if version == 2:

from Tkinter import \*

from tkFileDialog import asksaveasfilename

if version == 3:

from tkinter import \*

from tkinter.filedialog import asksaveasfilename

import threading

import socket

import random

import math

conn\_array = []

secret\_array = dict()

username\_array = dict()

contact\_array = dict()

username = "Self"

location = 0

port = 0

top = ""

main\_body\_text = 0

def binWord(word):

master = ""

for letter in word:

temp = bin(ord(letter))[2:]

while len(temp) < 7:

temp = '0' + temp

master = master + temp

return master

def xcrypt(message, key):

count = 0

master = ""

for letter in message:

if count == len(key):

count = 0

master += str(int(letter) ^ int(key[count]))

count += 1

return master

def x\_encode(string, number):

return xcrypt(binWord(string), bin(number)[2:])

def refract(binary):

master = ""

for x in range(0, int(len(binary) / 7)):

master += chr(int(binary[x \* 7: (x + 1) \* 7], 2) + 0)

return master

def formatNumber(number):

temp = str(number)

while len(temp) < 4:

temp = '0' + temp

return temp

def netThrow(conn, secret, message):

try:

conn.send(formatNumber(len(x\_encode(message, secret))).encode())

conn.send(x\_encode(message, secret).encode())

except socket.error:

if len(conn\_array) != 0:

writeToScreen(

"Connection issue. Sending message failed.", "System")

processFlag("-001")

def netCatch(conn, secret):

try:

data = conn.recv(4)

if data.decode()[0] == '-':

processFlag(data.decode(), conn)

return 1

data = conn.recv(int(data.decode()))

return refract(xcrypt(data.decode(), bin(secret)[2:]))

except socket.error:

if len(conn\_array) != 0:

writeToScreen(

"Connection issue. Receiving message failed.", "System")

processFlag("-001")

def isPrime(number):

x = 1

if number == 2 or number == 3:

return True

while x < math.sqrt(number):

x += 1

if number % x == 0:

return False

return True

def processFlag(number, conn=None):

global statusConnect

global conn\_array

global secret\_array

global username\_array

global contact\_array

global isCLI

t = int(number[1:])

if t == 1:

if len(conn\_array) == 1:

writeToScreen("Connection closed.", "System")

dump = secret\_array.pop(conn\_array[0])

dump = conn\_array.pop()

try:

dump.close()

except socket.error:

print("Issue with someone being bad about disconnecting")

if not isCLI:

statusConnect.set("Connect")

connecter.config(state=NORMAL)

return

if conn != None:

writeToScreen("Connect to " + conn.getsockname()

[0] + " closed.", "System")

dump = secret\_array.pop(conn)

conn\_array.remove(conn)

conn.close()

if t == 2:

name = netCatch(conn, secret\_array[conn])

if(isUsernameFree(name)):

writeToScreen(

"User " + username\_array[conn] + " has changed their username to " + name, "System")

username\_array[conn] = name

contact\_array[

conn.getpeername()[0]] = [conn.getpeername()[1], name]

if t == 4:

data = conn.recv(4)

data = conn.recv(int(data.decode()))

Client(data.decode(),

int(contact\_array[conn.getpeername()[0]][0])).start()

def processUserCommands(command, param):

global conn\_array

global secret\_array

global username

if command == "nick":

for letter in param[0]:

if letter == " " or letter == "\n":

if isCLI:

error\_window(0, "Invalid username. No spaces allowed.")

else:

error\_window(root, "Invalid username. No spaces allowed.")

return

if isUsernameFree(param[0]):

writeToScreen("Username is being changed to " + param[0], "System")

for conn in conn\_array:

conn.send("-002".encode())

netThrow(conn, secret\_array[conn], param[0])

username = param[0]

else:

writeToScreen(param[0] +

" is already taken as a username", "System")

if command == "disconnect":

for conn in conn\_array:

conn.send("-001".encode())

processFlag("-001")

if command == "connect":

if(options\_sanitation(param[1], param[0])):

Client(param[0], int(param[1])).start()

if command == "host":

if(options\_sanitation(param[0])):

Server(int(param[0])).start()

def isUsernameFree(name):

global username\_array

global username

for conn in username\_array:

if name == username\_array[conn] or name == username:

return False

return True

def passFriends(conn):

global conn\_array

for connection in conn\_array:

if conn != connection:

conn.send("-004".encode())

conn.send(

formatNumber(len(connection.getpeername()[0])).encode())

conn.send(connection.getpeername()[0].encode())

def client\_options\_window(master):

top = Toplevel(master)

top.title("Connection options")

top.protocol("WM\_DELETE\_WINDOW", lambda: optionDelete(top))

top.grab\_set()

Label(top, text="Server IP:").grid(row=0)

location = Entry(top)

location.grid(row=0, column=1)

location.focus\_set()

Label(top, text="Port:").grid(row=1)

port = Entry(top)

port.grid(row=1, column=1)

go = Button(top, text="Connect", command=lambda:

client\_options\_go(location.get(), port.get(), top))

go.grid(row=2, column=1)

def client\_options\_go(dest, port, window):

"Processes the options entered by the user in the client options window."""

if options\_sanitation(port, dest):

if not isCLI:

window.destroy()

Client(dest, int(port)).start()

elif isCLI:

sys.exit(1)

def options\_sanitation(por, loc=""):

global root

if version == 2:

por = unicode(por)

if isCLI:

root = 0

if not por.isdigit():

error\_window(root, "Please input a port number.")

return False

if int(por) < 0 or 65555 < int(por):

error\_window(root, "Please input a port number between 0 and 65555")

return False

if loc != "":

if not ip\_process(loc.split(".")):

error\_window(root, "Please input a valid ip address.")

return False

return True

def ip\_process(ipArray):

if len(ipArray) != 4:

return False

for ip in ipArray:

if version == 2:

ip = unicode(ip)

if not ip.isdigit():

return False

t = int(ip)

if t < 0 or 255 < t:

return False

return True

def server\_options\_window(master):

top = Toplevel(master)

top.title("Connection options")

top.grab\_set()

top.protocol("WM\_DELETE\_WINDOW", lambda: optionDelete(top))

Label(top, text="Port:").grid(row=0)

port = Entry(top)

port.grid(row=0, column=1)

port.focus\_set()

go = Button(top, text="Launch", command=lambda:

server\_options\_go(port.get(), top))

go.grid(row=1, column=1)

def server\_options\_go(port, window):

if options\_sanitation(port):

if not isCLI:

window.destroy()

Server(int(port)).start()

elif isCLI:

sys.exit(1)

def username\_options\_window(master):

top = Toplevel(master)

top.title("Username options")

top.grab\_set()

Label(top, text="Username:").grid(row=0)

name = Entry(top)

name.focus\_set()

name.grid(row=0, column=1)

go = Button(top, text="Change", command=lambda:

username\_options\_go(name.get(), top))

go.grid(row=1, column=1)

def username\_options\_go(name, window):

processUserCommands("nick", [name])

window.destroy()

def error\_window(master, texty):

global isCLI

if isCLI:

writeToScreen(texty, "System")

else:

window = Toplevel(master)

window.title("ERROR")

window.grab\_set()

Label(window, text=texty).pack()

go = Button(window, text="OK", command=window.destroy)

go.pack()

go.focus\_set()

def optionDelete(window):

connecter.config(state=NORMAL)

window.destroy()

def contacts\_window(master):

global contact\_array

cWindow = Toplevel(master)

cWindow.title("Contacts")

cWindow.grab\_set()

scrollbar = Scrollbar(cWindow, orient=VERTICAL)

listbox = Listbox(cWindow, yscrollcommand=scrollbar.set)

scrollbar.config(command=listbox.yview)

scrollbar.pack(side=RIGHT, fill=Y)

buttons = Frame(cWindow)

cBut = Button(buttons, text="Connect",

command=lambda: contacts\_connect(

listbox.get(ACTIVE).split(" ")))

cBut.pack(side=LEFT)

dBut = Button(buttons, text="Remove",

command=lambda: contacts\_remove(

listbox.get(ACTIVE).split(" "), listbox))

dBut.pack(side=LEFT)

aBut = Button(buttons, text="Add",

command=lambda: contacts\_add(listbox, cWindow))

aBut.pack(side=LEFT)

buttons.pack(side=BOTTOM)

for person in contact\_array:

listbox.insert(END, contact\_array[person][1] + " " +

person + " " + contact\_array[person][0])

listbox.pack(side=LEFT, fill=BOTH, expand=1)

def contacts\_connect(item):

Client(item[1], int(item[2])).start()

def contacts\_remove(item, listbox):

if listbox.size() != 0:

listbox.delete(ACTIVE)

global contact\_array

h = contact\_array.pop(item[1])

def contacts\_add(listbox, master):

aWindow = Toplevel(master)

aWindow.title("Contact add")

Label(aWindow, text="Username:").grid(row=0)

name = Entry(aWindow)

name.focus\_set()

name.grid(row=0, column=1)

Label(aWindow, text="IP:").grid(row=1)

ip = Entry(aWindow)

ip.grid(row=1, column=1)

Label(aWindow, text="Port:").grid(row=2)

port = Entry(aWindow)

port.grid(row=2, column=1)

go = Button(aWindow, text="Add", command=lambda:

contacts\_add\_helper(name.get(), ip.get(), port.get(),

aWindow, listbox))

go.grid(row=3, column=1)

def contacts\_add\_helper(username, ip, port, window, listbox):

for letter in username:

if letter == " " or letter == "\n":

error\_window(root, "Invalid username. No spaces allowed.")

return

if options\_sanitation(port, ip):

listbox.insert(END, username + " " + ip + " " + port)

contact\_array[ip] = [port, username]

window.destroy()

return

def load\_contacts():

global contact\_array

try:

filehandle = open("data\\contacts.dat", "r")

except IOError:

return

line = filehandle.readline()

while len(line) != 0:

temp = (line.rstrip('\n')).split(" ")

contact\_array[temp[0]] = temp[1:]

line = filehandle.readline()

filehandle.close()

def dump\_contacts():

global contact\_array

try:

filehandle = open("data\\contacts.dat", "w")

except IOError:

print("Can't dump contacts.")

return

for contact in contact\_array:

filehandle.write(

contact + " " + str(contact\_array[contact][0]) + " " +

contact\_array[contact][1] + "\n")

filehandle.close()

def placeText(text):

global conn\_array

global secret\_array

global username

writeToScreen(text, username)

for person in conn\_array:

netThrow(person, secret\_array[person], text)

def writeToScreen(text, username=""):

global main\_body\_text

global isCLI

if isCLI:

if username:

print(username + ": " + text)

else:

print(text)

else:

main\_body\_text.config(state=NORMAL)

main\_body\_text.insert(END, '\n')

if username:

main\_body\_text.insert(END, username + ": ")

main\_body\_text.insert(END, text)

main\_body\_text.yview(END)

main\_body\_text.config(state=DISABLED)

def processUserText(event):

data = text\_input.get()

if data[0] != "/":

placeText(data)

else:

if data.find(" ") == -1:

command = data[1:]

else:

command = data[1:data.find(" ")]

params = data[data.find(" ") + 1:].split(" ")

processUserCommands(command, params)

text\_input.delete(0, END)

def processUserInput(text):

if text[0] != "/":

placeText(text)

else:

if text.find(" ") == -1:

command = text[1:]

else:

command = text[1:text.find(" ")]

params = text[text.find(" ") + 1:].split(" ")

processUserCommands(command, params)

class Server (threading.Thread):

def \_\_init\_\_(self, port):

threading.Thread.\_\_init\_\_(self)

self.port = port

def run(self):

global conn\_array

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind(('', self.port))

if len(conn\_array) == 0:

writeToScreen(

"Socket is good, waiting for connections on port: " +

str(self.port), "System")

s.listen(1)

global conn\_init

conn\_init, addr\_init = s.accept()

serv = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

serv.bind(('', 0))

serv.listen(1)

portVal = str(serv.getsockname()[1])

if len(portVal) == 5:

conn\_init.send(portVal.encode())

else:

conn\_init.send(("0" + portVal).encode())

conn\_init.close()

conn, addr = serv.accept()

conn\_array.append(conn)

writeToScreen("Connected by " + str(addr[0]), "System")

global statusConnect

statusConnect.set("Disconnect")

connecter.config(state=NORMAL)

prime = random.randint(1000, 9000)

while not isPrime(prime):

prime = random.randint(1000, 9000)

base = random.randint(20, 100)

a = random.randint(20, 100)

conn.send(formatNumber(len(str(base))).encode())

conn.send(str(base).encode())

conn.send(formatNumber(len(str(prime))).encode())

conn.send(str(prime).encode())

conn.send(formatNumber(len(str(pow(base, a) % prime))).encode())

conn.send(str(pow(base, a) % prime).encode())

data = conn.recv(4)

data = conn.recv(int(data.decode()))

b = int(data.decode())

global secret\_array

secret = pow(b, a) % prime

secret\_array[conn] = secret

conn.send(formatNumber(len(username)).encode())

conn.send(username.encode())

data = conn.recv(4)

data = conn.recv(int(data.decode()))

if data.decode() != "Self":

username\_array[conn] = data.decode()

contact\_array[str(addr[0])] = [str(self.port), data.decode()]

else:

username\_array[conn] = addr[0]

contact\_array[str(addr[0])] = [str(self.port), "No\_nick"]

passFriends(conn)

threading.Thread(target=Runner, args=(conn, secret)).start()

Server(self.port).start()

class Client (threading.Thread):

def \_\_init\_\_(self, host, port):

threading.Thread.\_\_init\_\_(self)

self.port = port

self.host = host

def run(self):

global conn\_array

global secret\_array

conn\_init = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

conn\_init.settimeout(5.0)

try:

conn\_init.connect((self.host, self.port))

except socket.timeout:

writeToScreen("Timeout issue. Host possible not there.", "System")

connecter.config(state=NORMAL)

raise SystemExit(0)

except socket.error:

writeToScreen(

"Connection issue. Host actively refused connection.", "System")

connecter.config(state=NORMAL)

raise SystemExit(0)

porta = conn\_init.recv(5)

porte = int(porta.decode())

conn\_init.close()

conn = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

conn.connect((self.host, porte))

writeToScreen("Connected to: " + self.host +

" on port: " + str(porte), "System")

global statusConnect

statusConnect.set("Disconnect")

connecter.config(state=NORMAL)

conn\_array.append(conn)

data = conn.recv(4)

data = conn.recv(int(data.decode()))

base = int(data.decode())

data = conn.recv(4)

data = conn.recv(int(data.decode()))

prime = int(data.decode())

data = conn.recv(4)

data = conn.recv(int(data.decode()))

a = int(data.decode())

b = random.randint(20, 100)

conn.send(formatNumber(len(str(pow(base, b) % prime))).encode())

conn.send(str(pow(base, b) % prime).encode())

secret = pow(a, b) % prime

secret\_array[conn] = secret

conn.send(formatNumber(len(username)).encode())

conn.send(username.encode())

data = conn.recv(4)

data = conn.recv(int(data.decode()))

if data.decode() != "Self":

username\_array[conn] = data.decode()

contact\_array[

conn.getpeername()[0]] = [str(self.port), data.decode()]

else:

username\_array[conn] = self.host

contact\_array[conn.getpeername()[0]] = [str(self.port), "No\_nick"]

threading.Thread(target=Runner, args=(conn, secret)).start()

def Runner(conn, secret):

global username\_array

while 1:

data = netCatch(conn, secret)

if data != 1:

writeToScreen(data, username\_array[conn])

def QuickClient():

window = Toplevel(root)

window.title("Connection options")

window.grab\_set()

Label(window, text="Server IP:").grid(row=0)

destination = Entry(window)

destination.grid(row=0, column=1)

go = Button(window, text="Connect", command=lambda:

client\_options\_go(destination.get(), "9999", window))

go.grid(row=1, column=1)

def QuickServer():

Server(9999).start()

def saveHistory():

global main\_body\_text

file\_name = asksaveasfilename(

title="Choose save location",

filetypes=[('Plain text', '\*.txt'), ('Any File', '\*.\*')])

try:

filehandle = open(file\_name + ".txt", "w")

except IOError:

print("Can't save history.")

return

contents = main\_body\_text.get(1.0, END)

for line in contents:

filehandle.write(line)

filehandle.close()

def connects(clientType):

global conn\_array

connecter.config(state=DISABLED)

if len(conn\_array) == 0:

if clientType == 0:

client\_options\_window(root)

if clientType == 1:

server\_options\_window(root)

else:

for connection in conn\_array:

connection.send("-001".encode())

processFlag("-001")

def toOne():

global clientType

clientType = 0

def toTwo():

global clientType

clientType = 1

if len(sys.argv) > 1 and sys.argv[1] == "-cli":

print("Starting command line chat")

else:

root = Tk()

root.title("Chat")

menubar = Menu(root)

file\_menu = Menu(menubar, tearoff=0)

file\_menu.add\_command(label="Save chat", command=lambda: saveHistory())

file\_menu.add\_command(label="Change username",

command=lambda: username\_options\_window(root))

file\_menu.add\_command(label="Exit", command=lambda: root.destroy())

menubar.add\_cascade(label="File", menu=file\_menu)

connection\_menu = Menu(menubar, tearoff=0)

connection\_menu.add\_command(label="Quick Connect", command=QuickClient)

connection\_menu.add\_command(

label="Connect on port", command=lambda: client\_options\_window(root))

connection\_menu.add\_command(

label="Disconnect", command=lambda: processFlag("-001"))

menubar.add\_cascade(label="Connect", menu=connection\_menu)

server\_menu = Menu(menubar, tearoff=0)

server\_menu.add\_command(label="Launch server", command=QuickServer)

server\_menu.add\_command(label="Listen on port",

command=lambda: server\_options\_window(root))

menubar.add\_cascade(label="Server", menu=server\_menu)

menubar.add\_command(label="Contacts", command=lambda:

contacts\_window(root))

root.config(menu=menubar)

main\_body = Frame(root, height=20, width=50)

main\_body\_text = Text(main\_body)

body\_text\_scroll = Scrollbar(main\_body)

main\_body\_text.focus\_set()

body\_text\_scroll.pack(side=RIGHT, fill=Y)

main\_body\_text.pack(side=LEFT, fill=Y)

body\_text\_scroll.config(command=main\_body\_text.yview)

main\_body\_text.config(yscrollcommand=body\_text\_scroll.set)

main\_body.pack()

main\_body\_text.insert(END, "Welcome to the chatroom!")

main\_body\_text.config(state=DISABLED)

text\_input = Entry(root, width=60)

text\_input.bind("<Return>", processUserText)

text\_input.pack()

statusConnect = StringVar()

statusConnect.set("Connect")

clientType = 1

Radiobutton(root, text="Client", variable=clientType,

value=0, command=toOne).pack(anchor=E)

Radiobutton(root, text="Server", variable=clientType,

value=1, command=toTwo).pack(anchor=E)

connecter = Button(root, textvariable=statusConnect,

command=lambda: connects(clientType))

connecter.pack()

load\_contacts()

root.mainloop()

dump\_contacts()

* **Conclusion :**

Thus our project provides a better alternative for the traditional communication systems and makes the process of communicating more efficient, free from manual errors and less time consuming while ensuring maximum efficiency and safe way to communicate in these times of the pandemic .

* **Future Scopes and Enhancements :**

This project has an immensely boundless scope in future. It can be amended as and when requirement emerges, as it is versatile in terms of the extension. It can also be integrated and used as is in online forums data storage can be made server-based and can be integrated on multiple devices at the same time.

* **References :**

1. Official python documentation
2. Real python
3. Stack overflow
4. Geek For Geeks
5. YouTube
6. Programminghero