**FORMAN CHRISTIAN COLLEGE**

**(A CHARTERED UNIVERSITY)**

****

**COMP 311 B**

**SP24**

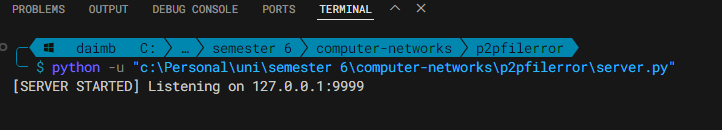
**TCP P2P Chat & File Sharing**

**Hafsah Shahbaz – 251684784**

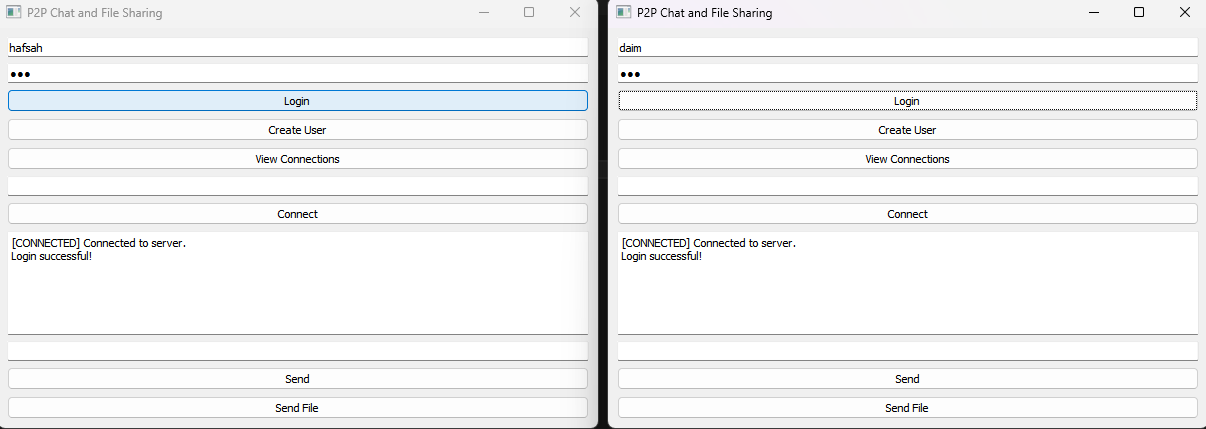
**Daim Bin Khalid - 251686775**

**14th May 2024**

**Demonstration:**



Logging in as users:

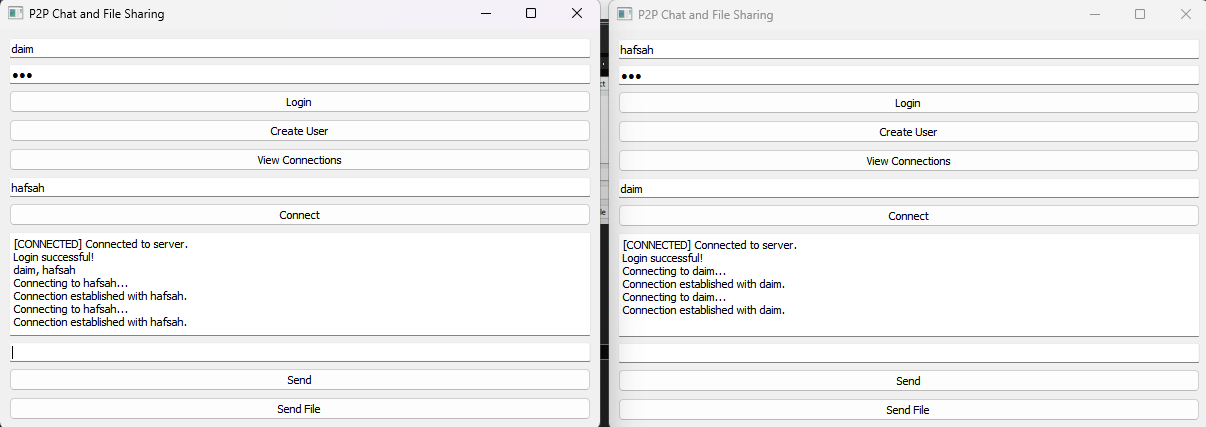


Checking connected users to server:

A screenshot of a computer

Description automatically generated

Establish connection to another peer:



Start Chatting:

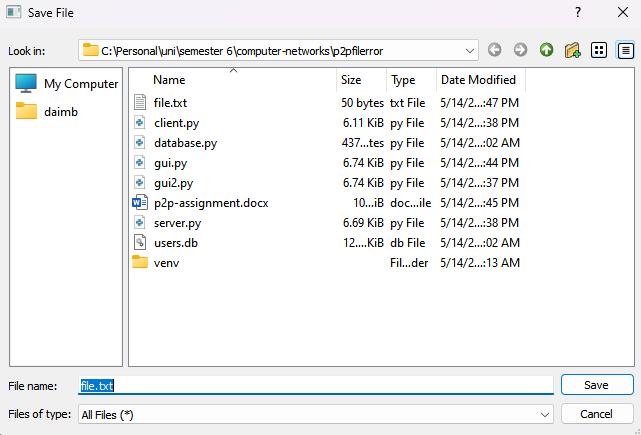
A screenshot of a computer

Description automatically generated

Share Files:

A screenshot of a computer

Description automatically generated



# -------------------

server.py:

# -------------------

import socket

import threading

import os

import sqlite3

clients = {} # store connected users

active\_sessions = {} # store active chat sessions

def handle\_client(client\_socket, client\_address):

global active\_sessions

print(f"[NEW CONNECTION] {client\_address} connected.")

while True:

data = client\_socket.recv(1024).decode('utf-8')

if not data:

break

print(f"[{client\_address}] {data}")

if data.startswith("login"):

\_, username, password = data.split()

if authenticate\_user(username, password):

clients[username] = client\_socket

client\_socket.sendall("Login successful!".encode('utf-8'))

else:

client\_socket.sendall(

"Invalid username or password.".encode('utf-8'))

elif data.startswith("create\_user"):

\_, username, password = data.split()

if create\_user(username, password):

client\_socket.sendall(

"User created successfully!".encode('utf-8'))

else:

client\_socket.sendall(

"Failed to create user. Please try again.".encode('utf-8'))

elif data.startswith("request\_clients"):

# send list of currently connected clients

client\_socket.sendall(", ".join(clients.keys()).encode('utf-8'))

elif data.startswith("connect"):

\_, sender\_username, recipient\_username = data.split()

establish\_connection(sender\_username, recipient\_username)

elif data.startswith("file"):

file\_path = data.split()[1]

broadcast\_file(username, recipient\_username, file\_path)

elif data.startswith("msg"):

message\_parts = data.split(' ', 2)

if len(message\_parts) >= 3:

recipient\_username = message\_parts[1]

actual\_message = message\_parts[2]

broadcast\_message(username, recipient\_username, actual\_message)

else:

print("Invalid message format.")

elif data == "disconnect":

print(

f"[{username}] Client requested to disconnect. Closing connection.")

if active\_sessions:

for key, \_ in list(active\_sessions.items()):

if username in key:

print("\n", active\_sessions[key])

del active\_sessions[key]

elif data == "quit":

print(

f"[{username}] Client requested to quit. Closing connection.")

client\_socket.close()

if username in clients:

del clients[username]

if active\_sessions:

for key, \_ in list(active\_sessions.items()):

if username in key:

del active\_sessions[key]

break

else:

broadcast\_message(username, recipient\_username, data)

print(f"[DISCONNECTED] {client\_address} disconnected.")

client\_socket.close()

def broadcast\_message(sender\_username, recipient\_username, message):

for (sender, recipient), client\_socket in active\_sessions.items():

if sender == recipient\_username:

try:

client\_socket.sendall(

f"[{sender\_username}]: {message}".encode('utf-8'))

except Exception as e:

print(f"Error broadcasting message to {recipient}: {e}")

def broadcast\_file(sender\_username, recipient\_username, file\_path):

try:

with open(file\_path, 'rb') as file:

file\_data = file.read()

for (sender, recipient), client\_socket in active\_sessions.items():

if sender == recipient\_username:

client\_socket.sendall(

f"FILE\_TRANSFER:[{sender\_username}] is sending a file: {os.path.basename(file\_path)}".encode('utf-8'))

client\_socket.sendall(file\_data)

except Exception as e:

print(f"Error broadcasting file to {recipient}: {e}")

def establish\_connection(sender\_username, recipient\_username):

sender\_socket = clients.get(sender\_username)

recipient\_socket = clients.get(recipient\_username)

if sender\_socket and recipient\_socket:

sender\_socket.sendall(

f"Connecting to {recipient\_username}...".encode('utf-8'))

recipient\_socket.sendall(

f"Connecting to {sender\_username}...".encode('utf-8'))

# store active session

active\_sessions[(sender\_username, recipient\_username)] = (

sender\_socket)

sender\_socket.sendall(

f"Connection established with {recipient\_username}.".encode('utf-8'))

recipient\_socket.sendall(

f"Connection established with {sender\_username}.".encode('utf-8'))

else:

sender\_socket.sendall(

f"User '{recipient\_username}' is not connected.".encode('utf-8'))

def create\_user(username, password):

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

try:

cursor.execute("SELECT \* FROM users WHERE username=?", (username,))

existing\_user = cursor.fetchone()

if existing\_user:

print("User already exists.")

return False

cursor.execute(

"INSERT INTO users (username, password) VALUES (?, ?)", (username, password))

conn.commit()

print("User created successfully.")

return True

except sqlite3.Error as e:

print(f"SQLite error: {e}")

return False

finally:

conn.close()

def authenticate\_user(username, password):

conn = sqlite3.connect('users.db')

cursor = conn.cursor()

cursor.execute(

"SELECT \* FROM users WHERE username=? AND password=?", (username, password))

user = cursor.fetchone()

conn.close()

return user is not None

def start\_server():

server\_host = "127.0.0.1"

server\_port = 9999

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((server\_host, server\_port))

server\_socket.listen(5)

print(f"[SERVER STARTED] Listening on {server\_host}:{server\_port}")

try:

while True:

client\_socket, client\_address = server\_socket.accept()

client\_thread = threading.Thread(

target=handle\_client, args=(client\_socket, client\_address))

client\_thread.start()

except KeyboardInterrupt:

print("[SERVER STOPPED] Server stopped.")

server\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

start\_server()

# -------------------

# -------------------

client.py:

# -------------------

import sys

from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QVBoxLayout, QWidget, QTextEdit, QLineEdit, QFileDialog, QMessageBox

import socket

import threading

import os

class ClientWindow(QMainWindow):

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

super().\_\_init\_\_()

self.setWindowTitle("P2P Chat and File Sharing")

self.setGeometry(100, 100, 600, 400)

self.central\_widget = QWidget()

self.setCentralWidget(self.central\_widget)

self.layout = QVBoxLayout()

self.central\_widget.setLayout(self.layout)

self.username\_field = QLineEdit()

self.layout.addWidget(self.username\_field)

self.password\_field = QLineEdit()

self.password\_field.setEchoMode(QLineEdit.Password)

self.layout.addWidget(self.password\_field)

self.login\_button = QPushButton("Login")

self.login\_button.clicked.connect(self.login)

self.layout.addWidget(self.login\_button)

self.create\_user\_button = QPushButton("Create User")

self.create\_user\_button.clicked.connect(self.create\_user)

self.layout.addWidget(self.create\_user\_button)

self.view\_connections\_button = QPushButton("View Connections")

self.view\_connections\_button.clicked.connect(self.view\_connections)

self.layout.addWidget(self.view\_connections\_button)

self.recipient\_field = QLineEdit()

self.layout.addWidget(self.recipient\_field)

self.connect\_button = QPushButton("Connect")

self.connect\_button.clicked.connect(self.connect\_to\_recipient)

self.layout.addWidget(self.connect\_button)

self.messages\_display = QTextEdit()

self.messages\_display.setReadOnly(True)

self.layout.addWidget(self.messages\_display)

self.input\_field = QLineEdit()

self.layout.addWidget(self.input\_field)

self.send\_button = QPushButton("Send")

self.send\_button.clicked.connect(self.send\_message)

self.layout.addWidget(self.send\_button)

self.file\_button = QPushButton("Send File")

self.file\_button.clicked.connect(self.choose\_file)

self.layout.addWidget(self.file\_button)

self.client\_socket = None

self.username = None

self.start\_client()

def start\_client(self):

server\_host = "127.0.0.1"

server\_port = 9999

self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

self.client\_socket.connect((server\_host, server\_port))

self.display\_message("[CONNECTED] Connected to server.")

# message receiving thread

receive\_thread = threading.Thread(target=self.receive\_messages)

receive\_thread.start()

except ConnectionRefusedError:

self.display\_message(

"[CONNECTION ERROR] Connection refused. Make sure the server is running.")

def login(self):

username = self.username\_field.text()

password = self.password\_field.text()

if not username or not password:

QMessageBox.warning(

self, "Warning", "Please enter username and password.")

return

login\_attempt = f"login {username} {password}"

self.client\_socket.sendall(login\_attempt.encode('utf-8'))

self.username = username

def create\_user(self):

username = self.username\_field.text()

password = self.password\_field.text()

if not username or not password:

QMessageBox.warning(

self, "Warning", "Please enter username and password.")

return

create\_user\_request = f"create\_user {username} {password}"

self.client\_socket.sendall(create\_user\_request.encode('utf-8'))

def view\_connections(self):

self.client\_socket.sendall("request\_clients".encode('utf-8'))

def connect\_to\_recipient(self):

recipient\_username = self.recipient\_field.text()

if not recipient\_username:

QMessageBox.warning(

self, "Warning", "Please enter recipient's username.")

return

connect\_request = f"connect {self.username} {recipient\_username}"

self.client\_socket.sendall(connect\_request.encode('utf-8'))

def receive\_messages(self):

while True:

try:

data = self.client\_socket.recv(1024)

if not data:

self.display\_message(

"[SERVER] Connection closed by server.")

break

message = data.decode('utf-8')

if message.startswith("FILE\_TRANSFER:"):

file\_info = message.split(": ")

file\_name = file\_info[1]

self.display\_message(f"Received file: {file\_name}")

save\_path, \_ = QFileDialog.getSaveFileName(

self, "Save File", os.path.basename(file\_name))

with open(save\_path, 'wb') as file:

while True:

file\_data = self.client\_socket.recv(1024)

if not file\_data:

break

file.write(file\_data)

self.display\_message("File saved successfully.")

else:

self.display\_message(message)

except ConnectionAbortedError:

self.display\_message("[CLIENT] Connection aborted by client.")

break

except ConnectionResetError:

self.display\_message("[CLIENT] Connection reset by client.")

break

def send\_message(self):

message = self.input\_field.text()

if message:

self.client\_socket.sendall(message.encode('utf-8'))

self.input\_field.clear()

self.display\_message(f"[{self.username}]: {message}")

def choose\_file(self):

file\_path, \_ = QFileDialog.getOpenFileName(self, "Choose File")

if file\_path:

self.send\_file(file\_path)

def send\_file(self, file\_path):

file\_name = os.path.basename(file\_path)

self.client\_socket.sendall(f"file {file\_name}".encode('utf-8'))

with open(file\_path, 'rb') as file:

while True:

chunk = file.read(1024)

if not chunk:

break

self.client\_socket.sendall(chunk)

def display\_message(self, message):

self.messages\_display.append(message)

def main():

app = QApplication(sys.argv)

window = ClientWindow()

window.show()

sys.exit(app.exec\_())

if \_\_name\_\_ == "\_\_main\_\_":

main()

# -------------------