**server.py**

from socket import \*

import threading

import mysql.connector

import json

class ZchatDB:

def \_\_init\_\_(self) -> None:

self.connect = mysql.connector.connect(host='127.0.0.1', user='root', passwd='smfsql123',database="zchat")

self.cur = self.connect.cursor()

print("Database connected successfully")

def add\_message(self):

pass

def check\_user(self,mobile\_number,active\_ip):

try:

self.cur.execute(f"SELECT \* FROM users where mobile\_number = dataenc({mobile\_number})")

result=self.cur.fetchall()

except:

return False

if len(result)>0:

self.cur.execute(f"UPDATE users SET active\_ip = '{active\_ip}' where mobile\_number = '{mobile\_number}'")

self.connect.commit()

try:

self.connect.close()

except:

pass

self.\_\_init\_\_()

return True

else:

return False

def get\_username(self,mobile\_number):

self.cur.execute(f"SELECT username FROM users where mobile\_number = dataenc({mobile\_number})")

result=self.cur.fetchone()

if len(result)==1:

return result[0]

else:

return "Nick"

def new\_user(self,username,mobile\_number,active\_ip):

insert\_query = '''

INSERT INTO users(username,mobile\_number,active\_ip)

VALUES (%s, %s, %s)

'''

print(f"New use created.\nusername: {username}\nMobile Number: {mobile\_number}\nActive IP: {active\_ip}")

data = (username,mobile\_number,active\_ip)

try:

self.cur.execute(insert\_query, data)

except:

self.\_\_init\_\_()

self.new\_user()

self.connect.commit()

try:

self.connect.close()

except:

pass

self.\_\_init\_\_()

return True

db=ZchatDB()

class server:

def \_\_init\_\_(self) -> None:

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

server.bind(("127.0.0.1", 5555))

server.listen()

print("Server is listening for incoming connections...")

self.clients = []

while True:

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

print(f"Accepted connection from {client\_addr}")

print(client\_addr,client\_socket)

self.clients.append(client\_socket)

client\_handler = threading.Thread(target=self.handle\_client, args=(client\_socket,))

client\_handler.start()

def handle\_client(self,client\_socket:socket):

while True:

try:

# Receive data from the client

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

if not data:

break

# Broadcast the received message to all connected clients

self.process\_data(client\_socket,data)

except Exception as e:

print(f"Error: {e}")

break

# Remove the disconnected client

self.clients.remove(client\_socket)

client\_socket.close()

def process\_data(self,client\_socket:socket,data):

data\_json={}

try:

dict\_data=json.loads(data)

if dict\_data['process']=="user\_check":

if db.check\_user(dict\_data['mobile\_number'],client\_socket.getsockname()[0]):

data\_json['process']="found"

client\_socket.send(json.dumps(data\_json).encode('utf-8'))

else:

data\_json['process']="notfound"

client\_socket.send(json.dumps(data\_json).encode('utf-8'))

return

if dict\_data['process']=="new\_user":

if db.new\_user(dict\_data['username'],dict\_data['mobile\_number'],client\_socket.getsockname()[0]):

dict\_data={

"process":"new\_user\_created",

"status":True

}

client\_socket.send(json.dumps(dict\_data).encode("utf-8"))

return

if dict\_data['process']=="message\_boardcast":

dict\_data['username']=db.get\_username(dict\_data['mobile\_number'])

self.broadcast(dict\_data,client\_socket)

return

except Exception as e:

data\_json['process']="error"

client\_socket.send(json.dumps(data\_json).encode('utf-8'))

raise e

def broadcast(self,message,client\_socket):

print(message)

for client in self.clients:

if client!=client\_socket:

try:

client.send(json.dumps(message).encode('utf-8'))

except Exception as e:

print(f"Error: {e}")

client.close()

self.clients.remove(client)

server()

**software frame.py**

import datetime

import json

import socket

import threading

import base64

from cryptography.fernet import Fernet

from tkinter import messagebox

from tkinter import \*

class VerticalScrolledFrame:

def \_\_init\_\_(self, master, \*\*kwargs):

width = kwargs.pop('width', None)

height = kwargs.pop('height', None)

bg = kwargs.pop('bg', kwargs.pop('background', None))

self.outer = Frame(master, \*\*kwargs)

self.vsb = Scrollbar(self.outer, orient=VERTICAL)

self.vsb.pack(fill=Y, side=RIGHT)

self.canvas = Canvas(self.outer, highlightthickness=0, width=width, height=height, bg=bg)

self.canvas.pack(side=LEFT, fill=BOTH, expand=True)

self.canvas['yscrollcommand'] = self.vsb.set

self.canvas.bind("<Enter>", self.\_bind\_mouse)

self.canvas.bind("<Leave>", self.\_unbind\_mouse)

self.vsb['command'] = self.canvas.yview

self.inner = Frame(self.canvas, bg=bg)

self.canvas.create\_window(4, 4, window=self.inner, anchor='nw',width=width)

self.inner.bind("<Configure>", self.\_on\_frame\_configure)

self.outer\_attr = set(dir(Widget))

def \_\_getattr\_\_(self, item):

if item in self.outer\_attr:

return getattr(self.outer, item)

else:

return getattr(self.inner, item)

def \_on\_frame\_configure(self, event=None):

x1, y1, x2, y2 = self.canvas.bbox("all")

height = self.canvas.winfo\_height()

self.canvas.config(scrollregion = (0,0, x2, max(y2, height)))

def \_bind\_mouse(self, event=None):

self.canvas.bind\_all("<4>", self.\_on\_mousewheel)

self.canvas.bind\_all("<5>", self.\_on\_mousewheel)

self.canvas.bind\_all("<MouseWheel>", self.\_on\_mousewheel)

def \_unbind\_mouse(self, event=None):

self.canvas.unbind\_all("<4>")

self.canvas.unbind\_all("<5>")

self.canvas.unbind\_all("<MouseWheel>")

def \_on\_mousewheel(self, event:Event):

if event.num == 4 or event.delta > 0:

self.canvas.yview\_scroll(-1, "units" )

elif event.num == 5 or event.delta < 0:

self.canvas.yview\_scroll(1, "units" )

def \_\_str\_\_(self):

return str(self.outer)

class Software:

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

if self.connect():

threading.Thread(target=self.software).start()

def software(self):

self.main=Tk()

self.main.title("Z Chat")

self.main.geometry("1200x700")

self.main.iconbitmap("icon.ico")

self.main.resizable(False,False)

self.main.withdraw()

self.main.bind("<Key>",self.clicker)

self.message\_entry = Text(self.main,height=4,width=80,font=("Arial",15))

self.message\_entry.place(x=200,y=610)

self.message\_entry.bind("<Return>",self.send\_message)

info\_frame = Frame(self.main,background="#AFEEEE",width=200,height=700)

info\_frame.place(x=1,y=1)

self.usernamelabel=Label(info\_frame,text="Username",font=(("Arial",18)),background="#AFEEEE",foreground="#000080")

self.usernamelabel.place(x=10,y=40)

self.mobilelabel=Label(info\_frame,text="Mobile",font=(("Arial",18)),background="#AFEEEE",foreground="#000080")

self.mobilelabel.place(x=10,y=100)

load\_history\_button=Button(info\_frame,text="Load History",font=("Arial",20),foreground="#000080",background="#87CEFA",command=self.fake\_history)

load\_history\_button.place(x=10,y=620)

self.chatframe = VerticalScrolledFrame(self.main,background="#87CEFA",width=980,height=610)

self.chatframe.place(x=200,y=1)

send\_button=Button(self.main,text="Send",font=("Arial",20),foreground="#000080",background="#AFEEEE",command=lambda:self.send\_message\_thread)

send\_button.place(x=1090,y=625)

self.login()

self.loginpage.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

self.main.mainloop()

def send\_message\_thread(self):

self.message\_send\_thread=threading.Thread(self.send\_message)

self.message\_send\_thread.start()

def connect(self):

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

try:

self.client.connect(("127.0.0.1", 5555))

self.client\_connected=True

return True

except Exception as error:

print(error)

self.nickname = messagebox.showwarning(title="Connection Error",message="Please check your internet and try again.\nIf your internet isn't problem Please contact admin to resolve this issue")

self.on\_closing()

return False

def showmainpage(self):

self.main.deiconify()

def login(self):

self.loginpage=Toplevel(self.main,background="azure3")

self.loginpage.title("Z Chat Login Page")

self.loginpage.geometry("300x450")

self.loginpage.iconbitmap("icon.ico")

self.loginpage.resizable(False,False)

usernamelabel=Label(self.loginpage,text="Enter User Name",background="azure3")

usernamelabel.place(relx=0.2, rely=0.3, anchor=W)

self.usernameentry=Entry(self.loginpage,width=30)

self.usernameentry.place(relx=0.21, rely=0.35, anchor=W)

mobilenumberlabel=Label(self.loginpage,text="Enter Mobile Number",background="azure3")

mobilenumberlabel.place(relx=0.2, rely=0.4, anchor=W)

self.number\_entry=Entry(self.loginpage,width=30)

self.number\_entry.place(relx=0.21, rely=0.45, anchor=W)

self.number\_entry.bind("<Return>",self.check\_user)

self.loginbutton=Button(self.loginpage,text="Login",background="azure3",width=15,height=2,command=self.check\_user,justify="center")

self.loginbutton.place(x=100, y=300, anchor=W,bordermode="inside")

self.usernameentry.focus\_force()

self.loginpage.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

def connect\_user(self,mobile\_number,active\_ip):

dict\_data={

"process":"user\_check",

"mobile\_number":mobile\_number,

"active\_ip":active\_ip

}

self.client.send(json.dumps(dict\_data).encode('utf-8'))

server\_response = self.client.recv(1024).decode("utf-8")

print(server\_response)

server\_response=json.loads(server\_response)

if server\_response['process']=="found":

return True

elif server\_response['process']=="notfound":

return False

def check\_user(self,event=None):

if self.usernameentry.get()=="" or self.number\_entry.get() == "" or self.usernameentry.get().startswith(" ") or self.number\_entry.get().startswith(" "):

messagebox.showwarning(title="Invalid Entry",message="Please correct the given Data")

self.loginpage\_reload()

return

mobile\_number=self.number\_entry.get()

active\_ip=socket.gethostbyname(socket.gethostname())

username=self.usernameentry.get()

try:

int(mobile\_number)

except:

messagebox.showerror(title="Invalid Mobile Number",message="Please Enter a valid Mobile Number")

return

if self.connect\_user(mobile\_number=mobile\_number,active\_ip=active\_ip):

self.loginpage.destroy()

self.main.deiconify()

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

self.receive\_thread.start()

self.usernamelabel.configure(text=f"{username}")

self.mobilelabel.configure(text=f"{mobile\_number}")

else:

if messagebox.askokcancel(title="Register New User",message="Are you want to register as new user ?"):

dict\_data={

"process":"new\_user",

"username":username,

"mobile\_number":mobile\_number

}

self.client.send(json.dumps(dict\_data).encode("utf-8"))

server\_response = json.loads(self.client.recv(1024).decode("utf-8"))

print(server\_response)

if server\_response["status"]:

messagebox.showinfo(title="New Login Registered",message=f"Username: {username}\nMobile Number: {mobile\_number}\nActive ID: {active\_ip}")

self.loginpage\_reload()

else:

self.loginpage\_reload()

def loginpage\_reload(self):

for i in self.loginpage.winfo\_children():

if isinstance(i,Entry):

i.delete(0,END)

def reload(self):

try:

self.chatframe.destroy()

except:

pass

self.chatframe = VerticalScrolledFrame(self.main,background="#87CEFA",width=980,height=610)

self.chatframe.place(x=200,y=1)

self.update\_scrollbar

def update\_scrollbar(self):

self.chatframe.canvas.yview\_moveto(1)

def send\_message(self,event=None):

message=self.message\_entry.get("1.0",END)

mobile\_number=self.mobilelabel.cget("text")

if message.startswith("\n") and len(message.split("\n"))<1:

messagebox.showwarning(title="Empty Message",message="The first Line is Empty")

self.message\_entry.after(10,self.clear\_text)

return

message\_json = {

"process":"message\_boardcast",

"mobile\_number":mobile\_number

}

if message.endswith("\n"):

message=message[:-1]

message\_json["message"]=ZS.encrypt\_message(message,mobile\_number).decode('utf-8')

message\_json['hash']=hash(message)

labelframe=LabelFrame(self.chatframe,text=datetime.datetime.now().\_\_format\_\_("%d-%m-%Y %H:%M:%S"))

labelframe.pack(padx=10,pady=10,anchor=E)

Label(labelframe,text=message,font=("Arial",12)).pack()

self.chatframe.outer.after(10,self.update\_scrollbar)

self.message\_entry.after(10,self.clear\_text)

self.client.send(json.dumps(message\_json).encode("utf-8"))

self.clear\_text

def receive(self):

print("Listening Server")

while True:

try:

dict\_data = json.loads(self.client.recv(1024).decode('utf-8'))

if dict\_data['process'] == 'message\_boardcast':

if dict\_data['mobile\_number']!=self.mobilelabel.cget('text'):

labelframe=LabelFrame(self.chatframe,text=f"{dict\_data['username']} {datetime.datetime.now().\_\_format\_\_('%d-%m-%Y %H:%M:%S')}")

labelframe.pack(padx=10,pady=10,anchor=W)

message=ZS.decrypt\_message(dict\_data['message'],dict\_data['mobile\_number']).encode('utf-8')

Label(labelframe,text=message,font=("Arial",12)).pack()

self.chatframe.outer.after(10,self.update\_scrollbar)

except Exception as e:

print(f"Error: {e}")

break

def on\_closing(self):

self.running = False

try:

self.client.close()

except:

pass

try:

self.main.destroy()

except:

pass

def clear\_text(self):

self.message\_entry.delete("1.0",END)

def fake\_history(self):

self.reload()

for i in range(1,110):

if i%2==0:

labelframe=LabelFrame(self.chatframe,text=datetime.datetime.now().\_\_format\_\_("%d-%m-%Y %H:%M:%S"))

labelframe.pack(padx=10,pady=10,anchor=E)

else:

labelframe=LabelFrame(self.chatframe,text=datetime.datetime.now().\_\_format\_\_("%d-%m-%Y %H:%M:%S"))

labelframe.pack(padx=10,pady=10,anchor=W)

Label(labelframe,text=i,font=("Arial",12)).pack(anchor=E)

self.chatframe.outer.after(10,self.update\_scrollbar)

def clicker(self,event:Event):

if event.keysym=="slash":

self.message\_entry.focus()

class ZSecure:

def encrypt\_message(self, message, mobile\_number):

cipher\_suite = Fernet(self.key\_generator(mobile\_number=mobile\_number))

encrypted\_message = cipher\_suite.encrypt(message.encode())

return encrypted\_message

def key\_generator(self,mobile\_number,timestamp=False):

if not timestamp:

timestamp=str(int(datetime.datetime.now().timestamp()))

str\_key=timestamp+mobile\_number

if len(str\_key) > 32 or len(str\_key) < 32:

str\_key= '0' \* (32 - len(str\_key)) + str\_key

key=base64.b64encode(f"{str\_key}".encode('utf-8'))

return key

def decrypt\_message(self,message,mobile\_number):

timestamp=str(int(datetime.datetime.now().timestamp()))

while True:

key=self.key\_generator(mobile\_number=mobile\_number,timestamp=timestamp)

try:

fernet=Fernet(key=key)

decrypted\_message = fernet.decrypt(message).decode()

return decrypted\_message

except:

timestamp=str(int(timestamp)-1)

ZS=ZSecure()

Software()