**Project Report**

**D-Chat: A Lightweight Chatting application**

**Submitted by:** Misbah Sarfaraz

**Roll No:** SE-24096

**Institution:** NED University of Engineering and Technology

**Supervisor:** Miss Asma Khan

**Date:** 25-11-2024

* **Executive Summary:**

D-Chat is a lightweight, server-based chatting application designed to provide real-time communication using MQTT (Message Queuing Telemetry Transport). The project focuses on efficient message delivery through publish-subscribe mechanisms and server-client communication. This report outlines the design, implementation, and results of the project, highlighting its features, challenges, and future prospects.

* **Introduction:**

**Background**

In today's digital era, messaging applications play a vital role in enabling seamless communication. However, many existing platforms are resource-intensive or unsuitable for constrained environments. MQTT, a lightweight messaging protocol, offers a solution with its efficient publish-subscribe model.

**Objective**

The objective of D-Chat is to create a user-friendly, server-based messaging application that leverages MQTT for real-time message delivery while maintaining scalability and efficiency.

**Scope**

D-Chat focuses on individual messaging features. The server handles user authentication, session management, and message delivery, ensuring scalability for multiple users.

* **System Design and Architecture:**

**System Overview**

D-Chat operates on a client-server model. Clients send and receive messages via an MQTT broker, while the server manages authentication and routing. The MQTT broker ensures minimal latency and reliable delivery.

**Architecture**

The architecture of D-Chat includes:

* **Client:** A user-friendly interface for sending and receiving messages.
* **Server:** Handles authentication, session tracking, and user management.
* **MQTT Broker:** Facilitates message transfer using topics for publish-subscribe communication.

**Technology Stack**

* **Programming Language:** Python
* **Frameworks/Libraries:** Paho MQTT, Tkinter (for GUI)
* **Database:** PostgreSQL (for user data storage)
* **Tools:** HiveMQ Cloud (for messaging)
* **Implementation**

**Development Process**

The development involved the following steps:

1. Setting up the HiveMQ Cloud (MQTT Broker).
2. Developing the server to manage user authentication and topic subscriptions.
3. Creating a client application using Python’s Tkinter for GUI and Paho MQTT for message handling.

**Key Features**

* **Real-Time Messaging:** Messages are delivered instantly via MQTT topics.
* **User Authentication:** Secure login system using a server-based approach.
* **Scalability:** The application supports multiple simultaneous users.
* **Results**

**Functional Outcomes**

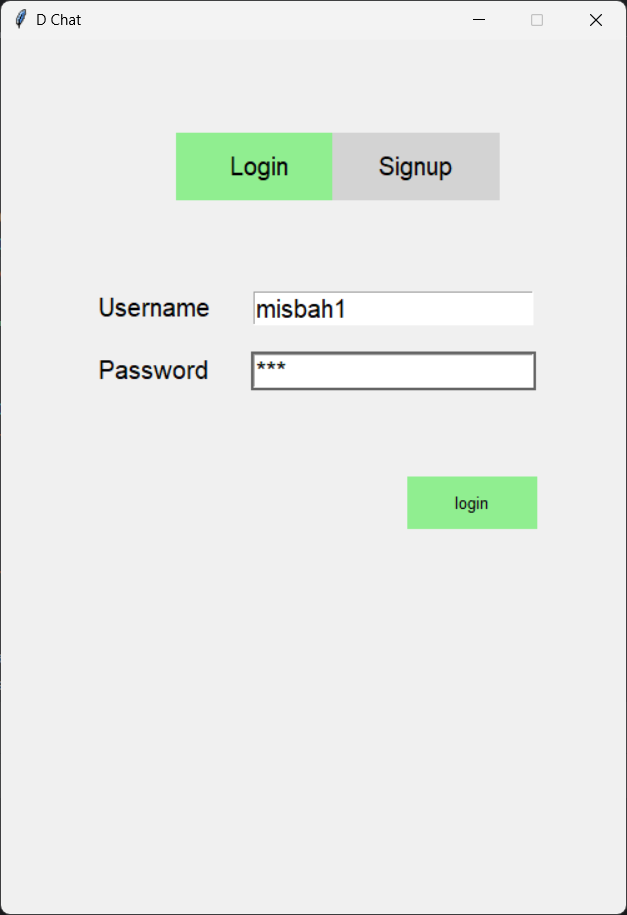
* + Successfully implemented real-time messaging with minimal latency.
  + The server reliably manages client connections and ensures secure message transfer**.**
* **Conclusion**

D-Chat achieved its objective of providing a lightweight and efficient chatting platform. The use of MQTT ensured real-time communication with minimal latency. Future enhancements include:

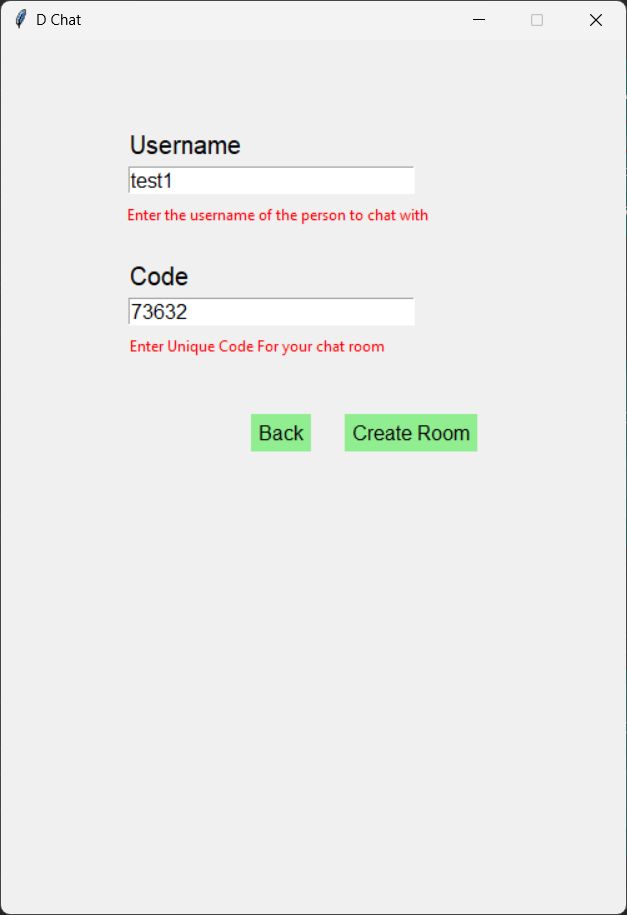
* File sharing capabilities.
* End-to-end encryption for secure communication.
* Offline message storage for improved user experience.

**Screen Shots:**

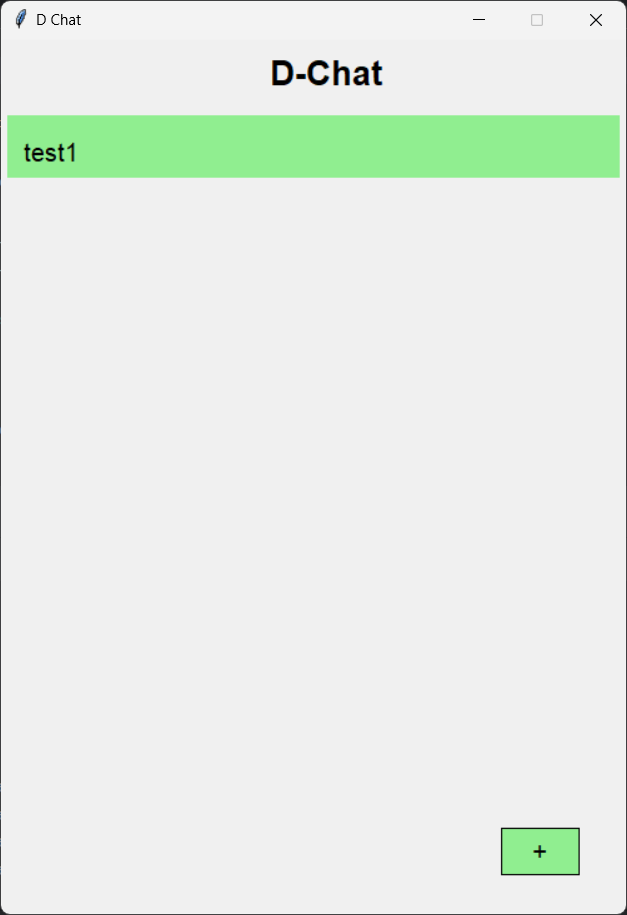
**Login/Signup Page**

****

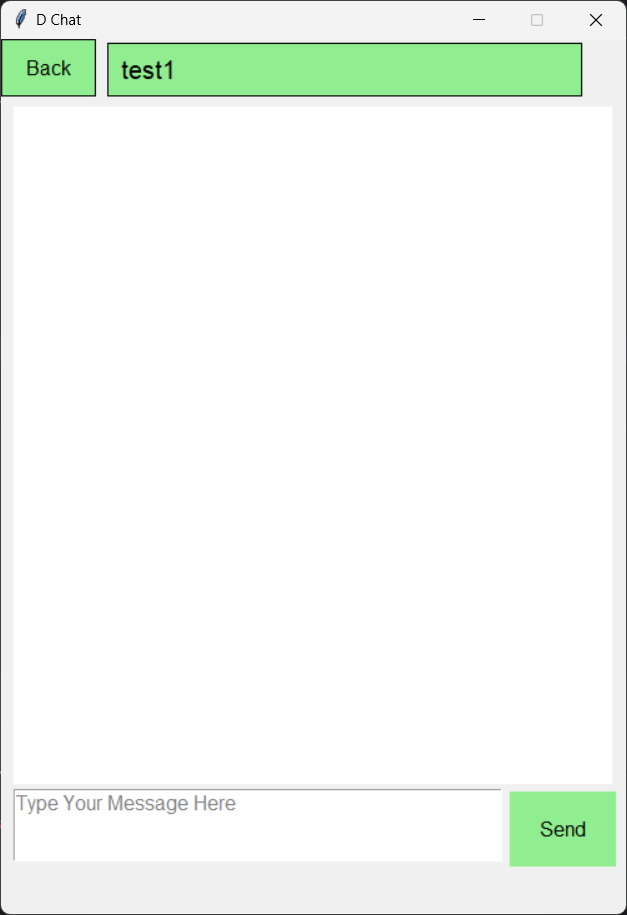
**Creating Chatroom**

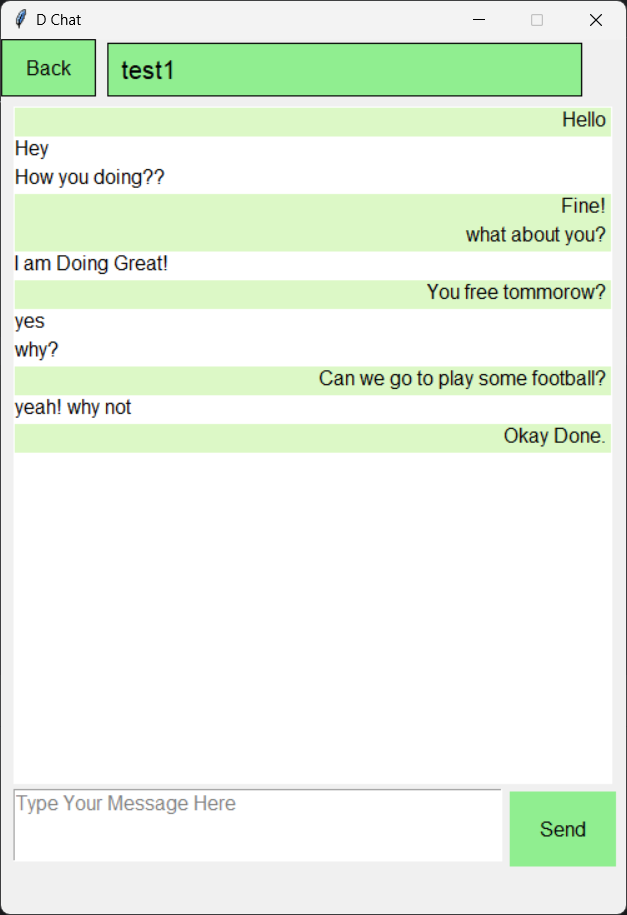
****

**Chat List Page**

****

**Chat room Page**

****

****

**Source Code:**

**GitHub Repositories:**

* **Application: https://github.com/Okaymisba/D-Chat (**[**Okaymisba/D-Chat**](https://github.com/Okaymisba/D-Chat)**)**
* **Server: https://github.com/Okaymisba/Server-for-D-chat (**[**Okaymisba/Server-for-D-chat**](https://github.com/Okaymisba/Server-for-D-chat)**)**

**Code:**

**For Application**

* **main.py**
* import json  
  import os  
  import tkinter as tk  
    
  import paho.mqtt.client as mqtt  
    
  from Add\_chatroom import add\_chatroom  
  from Functions import show\_frame, get\_username, add\_to\_room\_data  
  from chat\_list\_page import chat\_list\_frame  
  from page1 import page1, page2  
    
  root = tk.Tk()  
  root.geometry("500x700")  
  root.title("D Chat")  
  root.resizable(False, False)  
    
  add\_chatroom\_frame = add\_chatroom(root)  
  chat\_list\_frame, create\_chat\_room = chat\_list\_frame(root, add\_chatroom\_frame)  
  page1\_frame = page1(root, chat\_list\_frame)  
  page2\_frame = page2(root, chat\_list\_frame)  
    
  file\_for\_user\_credentials = "info.json"  
  file\_for\_room\_data = "room.txt"  
  room\_data = []  
    
  if os.path.exists(file\_for\_user\_credentials):  
   show\_frame(chat\_list\_frame)  
    
   if os.path.exists(file\_for\_room\_data):  
   with open(file\_for\_room\_data, 'r') as f:  
   room\_data = json.load(f)  
    
   for data in room\_data:  
   create\_chat\_room(root, data[0], data[1])  
  else:  
   show\_frame(page2\_frame)  
    
    
  def on\_connect(client, userdata, flags, rc, properties=None):  
   if rc == 0:  
   client.subscribe("application/create")  
   else:  
   print(f"Failed to connect, return code: {rc}")  
    
    
  def on\_message(client, userdata, msg, properties=None):  
   chat\_room\_data = json.loads(msg.payload.decode("utf-8"))  
   application\_username = get\_username()  
   if application\_username == chat\_room\_data["username"] and chat\_room\_data["code\_available"] == "True" and \  
   chat\_room\_data["recipient\_available"] == True:  
   room\_information = [chat\_room\_data["recipient"], chat\_room\_data["topic"]]  
   add\_to\_room\_data(room\_data, room\_information)  
   create\_chat\_room(root, chat\_room\_data["recipient"], chat\_room\_data["topic"])  
   print(chat\_room\_data)  
    
    
  broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
  port = 8883  
  username = "Reader"  
  password = "Reader123"  
    
  client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
  client.username\_pw\_set(username, password)  
  client.tls\_set()  
  client.on\_connect = on\_connect  
  client.on\_message = on\_message  
  client.connect(broker, port)  
  client.loop\_start()  
    
  root.mainloop()
* **page1.py**
* import json  
  import os  
  import tkinter as tk  
    
  import paho.mqtt.client as mqtt  
    
  from Add\_chatroom import add\_chatroom  
  from Functions import show\_frame, get\_username, add\_to\_room\_data  
  from chat\_list\_page import chat\_list\_frame  
  from page1 import page1, page2  
    
  root = tk.Tk()  
  root.geometry("500x700")  
  root.title("D Chat")  
  root.resizable(False, False)  
    
  add\_chatroom\_frame = add\_chatroom(root)  
  chat\_list\_frame, create\_chat\_room = chat\_list\_frame(root, add\_chatroom\_frame)  
  page1\_frame = page1(root, chat\_list\_frame)  
  page2\_frame = page2(root, chat\_list\_frame)  
    
  file\_for\_user\_credentials = "info.json"  
  file\_for\_room\_data = "room.txt"  
  room\_data = []  
    
  if os.path.exists(file\_for\_user\_credentials):  
   show\_frame(chat\_list\_frame)  
    
   if os.path.exists(file\_for\_room\_data):  
   with open(file\_for\_room\_data, 'r') as f:  
   room\_data = json.load(f)  
    
   for data in room\_data:  
   create\_chat\_room(root, data[0], data[1])  
  else:  
   show\_frame(page2\_frame)  
    
    
  def on\_connect(client, userdata, flags, rc, properties=None):  
   if rc == 0:  
   client.subscribe("application/create")  
   else:  
   print(f"Failed to connect, return code: {rc}")  
    
    
  def on\_message(client, userdata, msg, properties=None):  
   chat\_room\_data = json.loads(msg.payload.decode("utf-8"))  
   application\_username = get\_username()  
   if application\_username == chat\_room\_data["username"] and chat\_room\_data["code\_available"] == "True" and \  
   chat\_room\_data["recipient\_available"] == True:  
   room\_information = [chat\_room\_data["recipient"], chat\_room\_data["topic"]]  
   add\_to\_room\_data(room\_data, room\_information)  
   create\_chat\_room(root, chat\_room\_data["recipient"], chat\_room\_data["topic"])  
   print(chat\_room\_data)  
    
    
  broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
  port = 8883  
  username = "Reader"  
  password = "Reader123"  
    
  client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
  client.username\_pw\_set(username, password)  
  client.tls\_set()  
  client.on\_connect = on\_connect  
  client.on\_message = on\_message  
  client.connect(broker, port)  
  client.loop\_start()  
    
  root.mainloop()
* **chat\_list\_page.py**
* import json  
  import os  
  import tkinter as tk  
    
  import paho.mqtt.client as mqtt  
    
  from Add\_chatroom import add\_chatroom  
  from Functions import show\_frame, get\_username, add\_to\_room\_data  
  from chat\_list\_page import chat\_list\_frame  
  from page1 import page1, page2  
    
  root = tk.Tk()  
  root.geometry("500x700")  
  root.title("D Chat")  
  root.resizable(False, False)  
    
  add\_chatroom\_frame = add\_chatroom(root)  
  chat\_list\_frame, create\_chat\_room = chat\_list\_frame(root, add\_chatroom\_frame)  
  page1\_frame = page1(root, chat\_list\_frame)  
  page2\_frame = page2(root, chat\_list\_frame)  
    
  file\_for\_user\_credentials = "info.json"  
  file\_for\_room\_data = "room.txt"  
  room\_data = []  
    
  if os.path.exists(file\_for\_user\_credentials):  
   show\_frame(chat\_list\_frame)  
    
   if os.path.exists(file\_for\_room\_data):  
   with open(file\_for\_room\_data, 'r') as f:  
   room\_data = json.load(f)  
    
   for data in room\_data:  
   create\_chat\_room(root, data[0], data[1])  
  else:  
   show\_frame(page2\_frame)  
    
    
  def on\_connect(client, userdata, flags, rc, properties=None):  
   if rc == 0:  
   client.subscribe("application/create")  
   else:  
   print(f"Failed to connect, return code: {rc}")  
    
    
  def on\_message(client, userdata, msg, properties=None):  
   chat\_room\_data = json.loads(msg.payload.decode("utf-8"))  
   application\_username = get\_username()  
   if application\_username == chat\_room\_data["username"] and chat\_room\_data["code\_available"] == "True" and \  
   chat\_room\_data["recipient\_available"] == True:  
   room\_information = [chat\_room\_data["recipient"], chat\_room\_data["topic"]]  
   add\_to\_room\_data(room\_data, room\_information)  
   create\_chat\_room(root, chat\_room\_data["recipient"], chat\_room\_data["topic"])  
   print(chat\_room\_data)  
    
    
  broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
  port = 8883  
  username = "Reader"  
  password = "Reader123"  
    
  client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
  client.username\_pw\_set(username, password)  
  client.tls\_set()  
  client.on\_connect = on\_connect  
  client.on\_message = on\_message  
  client.connect(broker, port)  
  client.loop\_start()  
    
  root.mainloop()
* **add\_chatroom.py**
* import json  
  import threading  
  import time  
  import tkinter as tk  
  from tkinter import messagebox  
  import paho.mqtt.client as mqtt  
    
  from Functions import hide\_frame, create\_chatroom, get\_username  
    
    
  def on\_connect\_for\_application\_create(client, userdata, flags, rc):  
   if rc == 0:  
   print("Connected to broker")  
   else:  
   print(f"Failed to connect, return code: {rc}")  
    
    
  def on\_message\_for\_application\_create(client, userdata, msg):  
   message\_payload = json.loads(msg.payload.decode("utf-8"))  
    
   if message\_payload["username"] == get\_username():  
   global mqtt\_data\_received, mqtt\_data\_for\_create\_chatroom  
   mqtt\_data\_received = True  
   mqtt\_data\_for\_create\_chatroom = message\_payload  
    
    
  mqtt\_data\_received = False  
  mqtt\_data\_for\_create\_chatroom = None  
    
  broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
  port = 8883  
  topic = "application/create"  
  username = "Reader"  
  password = "Reader123"  
    
  client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
  client.username\_pw\_set(username, password)  
  client.tls\_set()  
  client.on\_connect = on\_connect\_for\_application\_create  
  client.on\_message = on\_message\_for\_application\_create  
  client.connect(broker, port)  
  client.loop\_start()  
    
  client.subscribe(topic)  
    
    
  def validate\_inputs\_for\_chatroom\_creation(username, chat\_with, code):  
   with open("room.txt", "r") as f:  
   room\_data = json.load(f)  
   for data in room\_data:  
   if chat\_with == data[0]:  
   messagebox.showerror("Invalid Username", "A chatroom with this user already exists")  
   return False  
   if chat\_with == username:  
   messagebox.showerror("Invalid Username", "You can't chat with yourself")  
   return False  
    
   if not username or ' ' in username:  
   messagebox.showerror("Invalid Input", "Username cannot be empty or contain spaces.")  
   return False  
    
   if not chat\_with or ' ' in chat\_with:  
   messagebox.showerror("Invalid Input", "Person to chat with cannot be empty or contain spaces.")  
   return False  
    
   if not code or ' ' in code:  
   messagebox.showerror("Invalid Input", "Code cannot be empty or contain spaces.")  
   return False  
    
   return True  
    
    
  def add\_chatroom(parent):  
   add\_chatroom\_frame = tk.Frame(parent, width=500, height=700)  
   add\_chatroom\_frame.grid(row=0, column=0, sticky='nsew')  
    
   label\_for\_username = tk.Label(add\_chatroom\_frame, text="Username", font=("Helvetica", 15))  
   label\_for\_username.place(x=100, y=70)  
    
   label\_for\_username\_of\_the\_person\_to\_chat = tk.Label(add\_chatroom\_frame,  
   text="Enter the username of the person to chat with", fg="red")  
   label\_for\_username\_of\_the\_person\_to\_chat.place(x=98, y=130)  
    
   input\_field\_for\_person\_to\_chat\_with = tk.Entry(add\_chatroom\_frame, width=25, font=("Helvetica", 12),  
   highlightthickness=2)  
   input\_field\_for\_person\_to\_chat\_with.place(x=100, y=100)  
    
   label\_for\_code = tk.Label(add\_chatroom\_frame, text="Code", font=("Helvetica", 15))  
   label\_for\_code.place(x=100, y=175)  
    
   input\_field\_for\_code = tk.Entry(add\_chatroom\_frame, width=25, font=("Helvetica", 12), highlightthickness=2)  
   input\_field\_for\_code.place(x=100, y=205)  
    
   label\_for\_chatroom\_code = tk.Label(add\_chatroom\_frame, text="Enter Unique Code For your chat room", fg="red")  
   label\_for\_chatroom\_code.place(x=100, y=235)  
    
   def handle\_create\_room():  
   username = get\_username()  
   chat\_with = input\_field\_for\_person\_to\_chat\_with.get()  
   code = input\_field\_for\_code.get()  
    
   if validate\_inputs\_for\_chatroom\_creation(username, chat\_with, code):  
   # hide\_frame(add\_chatroom\_frame)  
   create\_chatroom(username, chat\_with, code)  
   button\_for\_creating\_room.config(state="disabled", text="Creating Room")  
    
   threading.Thread(target=creating\_room).start()  
    
   def creating\_room():  
   global mqtt\_data\_received, mqtt\_data\_for\_create\_chatroom  
    
   for \_ in range(50):  
   if mqtt\_data\_received:  
   break  
   time.sleep(0.1)  
    
   if mqtt\_data\_received:  
   if mqtt\_data\_for\_create\_chatroom and mqtt\_data\_for\_create\_chatroom.get("code\_available") == "False":  
   messagebox.showerror("Invalid Input", "This Chatroom code is not available.")  
   elif mqtt\_data\_for\_create\_chatroom and mqtt\_data\_for\_create\_chatroom.get("recipient\_available") == False:  
   messagebox.showerror("Invalid Input", "The Username of the person doesn't exist.")  
   else:  
   hide\_frame(add\_chatroom\_frame)  
   else:  
   messagebox.showerror("Error", "Failed to connect, Please try again")  
    
   mqtt\_data\_received = False  
   mqtt\_data\_for\_create\_chatroom = None  
    
   button\_for\_creating\_room.config(state="normal", text="Create Room")  
    
   button\_for\_creating\_room = tk.Button(add\_chatroom\_frame, text="Create Room", font=("Helvetica", 12), padx=2, pady=2,  
   bd=0, bg="light green", cursor="hand2", relief="solid",  
   activebackground="light green",  
   command=handle\_create\_room)  
   button\_for\_creating\_room.place(x=275, y=300)  
    
   back\_button = tk.Button(add\_chatroom\_frame, text='Back', font=("Helvetica", 12), padx=2, pady=2,  
   bd=0, bg="light green", cursor="hand2", relief="solid",  
   activebackground="light green", command=lambda: hide\_frame(add\_chatroom\_frame))  
   back\_button.place(x=200, y=300)  
    
   return add\_chatroom\_frame
* **chat\_room.py**
* import json  
  import tkinter as tk  
  import paho.mqtt.client as mqtt  
    
  from Functions import get\_username, add\_placeholder\_for\_text, hide\_frame  
    
  chat\_rooms = {}  
    
    
  def on\_connect(client, userdata, flags, rc, properties=None):  
   if rc == 0:  
   client.subscribe("#")  
   else:  
   print(f"Failed to connect, return code: {rc}")  
    
    
  def on\_message(client, userdata, msg, properties=None):  
   message = json.loads(msg.payload.decode("utf-8"))  
   print(message)  
   try:  
   if message["topic"] in chat\_rooms and message["to"] == get\_username():  
   display\_widget = chat\_rooms[msg.topic]["display"]  
   add\_message\_for\_sender(display\_widget, message["message"])  
   print(message)  
   except KeyError:  
   print("Nothing to worry")  
    
    
  broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
  port = 8883  
  username = "Reader"  
  password = "Reader123"  
    
  client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
  client.username\_pw\_set(username, password)  
  client.tls\_set()  
  client.on\_connect = on\_connect  
  client.on\_message = on\_message  
  client.connect(broker, port)  
  client.loop\_start()  
    
    
  def chat\_room(parent, topic, recipient\_name):  
   chat\_room = tk.Frame(parent, width=500, height=700)  
    
   label\_for\_recipient\_name = tk.Label(chat\_room, text=recipient\_name, font=("Helvetica", 15), padx=10, pady=9,  
   bg="light Green", anchor="w", bd=1, relief="solid")  
   label\_for\_recipient\_name.place(x=85, y=3, width=380)  
    
   message\_box = tk.Text(chat\_room, wrap="word", font=("Helvetica", 12), width=43, height=3)  
   message\_box.place(x=10, y=600)  
   add\_placeholder\_for\_text(message\_box, "Type Your Message Here")  
    
   # temporary solution  
   scrollbar = tk.Scrollbar(chat\_room, command=message\_box.yview)  
   scrollbar.place(x=1000, y=600)  
   message\_box.config(yscrollcommand=scrollbar.set)  
    
   message\_display = tk.Text(chat\_room, wrap="word", state="disabled", height=30, width=53, font=("Helvetica", 12),  
   bd=0)  
   message\_display.place(x=10, y=54)  
    
   chat\_rooms[topic] = {"frame": chat\_room, "display": message\_display}  
    
   back\_button = tk.Button(chat\_room, text="Back", font=("Helvetica", 12), padx=15, pady=9,  
   bd=1, bg="light green", cursor="hand2", relief="solid",  
   activebackground="light green", command=lambda: hide\_frame(chat\_room))  
   back\_button.place(x=0, y=0)  
    
   send\_button = tk.Button(chat\_room, text="Send", font=("Helvetica", 12), padx=20, pady=17,  
   bd=0, bg="light green", cursor="hand2", relief="solid",  
   activebackground="light green",  
   command=lambda: send\_message(topic, message\_box, message\_display, recipient\_name))  
   send\_button.place(x=407, y=602)  
    
   def send\_message\_and\_break(event=None):  
   send\_message(topic, message\_box, message\_display, recipient\_name)  
   return "break"  
    
   message\_box.bind("<Return>", send\_message\_and\_break)  
    
   return chat\_room  
    
    
  def add\_message\_for\_sender(text\_widget, message):  
   text\_widget.tag\_configure("sender", justify="left", foreground="black", spacing3=5)  
   text\_widget.config(state=tk.NORMAL)  
   text\_widget.insert(tk.END, message + "\n", "sender")  
   text\_widget.config(state=tk.DISABLED)  
   text\_widget.see(tk.END)  
    
    
  def add\_message\_for\_me(text\_message, message):  
   text\_message.tag\_configure("me", justify="right", background="#dcf8c6", foreground="black", spacing3=5)  
   text\_message.config(state=tk.NORMAL)  
   text\_message.insert(tk.END, message + "\n", "me")  
   text\_message.config(state=tk.DISABLED)  
   text\_message.see(tk.END)  
    
    
  def send\_message(topic, entry\_widget, display\_widget, recipient):  
   message = entry\_widget.get("1.0", tk.END).strip()  
   if message:  
   json\_message = {  
   "topic": topic,  
   "message": message,  
   "to": recipient  
   }  
   client.publish(topic, json.dumps(json\_message))  
   add\_message\_for\_me(display\_widget, f" {message} ")  
   entry\_widget.delete("1.0", tk.END)  
   print(topic)
* **Functions.py**
* import json  
  import paho.mqtt.client as mqtt  
  import tkinter as tk  
    
    
  def get\_username():  
   with open("info.json", 'r') as read\_file:  
   info = json.load(read\_file)  
   username = info['username']  
   return username  
    
    
  def on\_connect(client, userdata, flags, rc, properties=None):  
   if rc == 0:  
   print("Connected to broker")  
   else:  
   print(f"Failed to connect, return code: {rc}")  
    
    
  def on\_message(client, userdata, msg, properties=None):  
   print(f"{msg.topic}: {msg.payload.decode()}")  
    
    
  def show\_frame(frame):  
   frame.grid(row=0, column=0, sticky="nsew")  
   frame.tkraise()  
    
    
  def hide\_frame(frame):  
   frame.grid\_forget()  
    
    
  def save\_user\_info\_on\_device(username, password):  
   with open("info.json", "w") as f:  
   user\_data = {"username": username,  
   "password": password}  
   json.dump(user\_data, f)  
    
    
  def save\_user\_info\_on\_database\_for\_signup(username, password, identity):  
   user\_data = {"identity": identity,  
   "username": username,  
   "password": password}  
    
   broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
   port = 8883  
   topic = "server/signup"  
   username = "Reader"  
   password = "Reader123"  
    
   client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
   client.username\_pw\_set(username, password)  
   client.tls\_set()  
   client.on\_connect = on\_connect  
   client.on\_message = on\_message  
   client.connect(broker, port)  
   client.loop\_start()  
    
   client.subscribe(topic)  
    
   client.publish(topic, json.dumps(user\_data))  
    
    
  def authenticate\_user\_for\_login(username, password, identity):  
   user\_data = {"identity": identity,  
   "username": username,  
   "password": password}  
    
   broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
   port = 8883  
   topic = "server/login"  
   username = "Reader"  
   password = "Reader123"  
    
   client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
   client.username\_pw\_set(username, password)  
   client.tls\_set()  
   client.on\_connect = on\_connect  
   client.on\_message = on\_message  
   client.connect(broker, port)  
   client.loop\_start()  
    
   client.subscribe(topic)  
    
   client.publish(topic, json.dumps(user\_data))  
    
    
  def create\_chatroom(username, recipient\_username, chatroom\_code):  
   recipient = recipient\_username  
   code = chatroom\_code  
   data = {  
   "username": username,  
   "recipient": recipient,  
   "code": code  
   }  
    
   broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
   port = 8883  
   topic = "server/create"  
   username = "Reader"  
   password = "Reader123"  
    
   client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
   client.username\_pw\_set(username, password)  
   client.tls\_set()  
   client.on\_connect = on\_connect\_for\_create\_chatroom  
   client.on\_message = on\_message\_for\_create\_chatroom  
   client.connect(broker, port)  
   client.loop\_start()  
    
   client.subscribe(topic)  
    
   client.publish(topic, json.dumps(data))  
   print(f"Data Published: {data}")  
    
    
  def on\_connect\_for\_create\_chatroom(client, userdata, flags, rc):  
   if rc == 0:  
   print("Connected to broker")  
   else:  
   print(f"Failed to connect, return code: {rc}")  
    
    
  def on\_message\_for\_create\_chatroom(client, userdata, msg):  
   print(f"{msg.topic}: {msg.payload.decode()}")  
    
    
  def add\_placeholder(entry, placeholder):  
   entry.insert(0, placeholder)  
   entry.config(fg="grey")  
    
   def on\_focus\_in(event):  
   if entry.get() == placeholder:  
   entry.delete(0, tk.END)  
   entry.config(fg="black")  
    
   def on\_focus\_out(event):  
   if entry.get() == "":  
   entry.insert(0, placeholder)  
   entry.config(fg="grey")  
    
   entry.bind("<FocusIn>", on\_focus\_in)  
   entry.bind("<FocusOut>", on\_focus\_out)  
    
    
  def add\_placeholder\_for\_password(entry, placeholder):  
   entry.insert(0, placeholder)  
   entry.config(fg="grey", show="")  
    
   def on\_focus\_in(event):  
   if entry.get() == placeholder:  
   entry.delete(0, tk.END)  
   entry.config(fg="black", show="\*")  
    
   def on\_focus\_out(event):  
   if entry.get() == "":  
   entry.insert(0, placeholder)  
   entry.config(fg="grey", show="")  
    
   entry.bind("<FocusIn>", on\_focus\_in)  
   entry.bind("<FocusOut>", on\_focus\_out)  
    
    
  def add\_placeholder\_for\_text(text\_widget, placeholder\_text):  
   text\_widget.insert("1.0", placeholder\_text)  
   text\_widget.config(fg="grey")  
    
   def on\_focus\_in(event):  
   if text\_widget.get("1.0", "end-1c") == placeholder\_text:  
   text\_widget.delete("1.0", "end")  
   text\_widget.config(fg="black")  
    
   def on\_focus\_out(event):  
   if text\_widget.get("1.0", "end-1c") == "":  
   text\_widget.insert("1.0", placeholder\_text)  
   text\_widget.config(fg="grey")  
    
   text\_widget.bind("<FocusIn>", on\_focus\_in)  
   text\_widget.bind("<FocusOut>", on\_focus\_out)  
    
    
  def add\_to\_room\_data(room\_data, list\_to\_add):  
   room\_data.append(list\_to\_add)  
    
   with open("room.txt", "w") as file:  
   file.write(json.dumps(room\_data))

**For Server**

* + **main.py**
* import json  
  import psycopg2  
  import paho.mqtt.client as mqtt  
    
  from Functions import username\_exists, hash\_password, authenticate\_user  
    
    
  def on\_connect(client, userdata, flags, rc, properties=None):  
   print("Connected to MQTT broker.")  
   client.subscribe("#")  
    
    
  def on\_message(client, userdata, msg, properties=None):  
   print("Received a message on topic ", msg.topic)  
    
   if msg.topic == "server/signup":  
   json\_login\_info = json.loads(msg.payload.decode("utf-8"))  
   print(f"Received message: {json\_login\_info}")  
    
   try:  
   execute\_in\_database(  
   "INSERT INTO user\_data (username, password) VALUES (%s, %s);",  
   (json\_login\_info["username"].lower(), hash\_password(json\_login\_info["password"]))  
   )  
   except psycopg2.IntegrityError as e:  
   if "unique constraint" in str(e):  
   returning\_data = {  
   "identity": json\_login\_info["identity"],  
   "user\_taken": "True"  
   }  
    
   client.publish("application/signup", json.dumps(returning\_data))  
    
   except Exception as e:  
   print("An error occurred while inserting data:", e)  
    
   else:  
   returning\_data = {  
   "identity": json\_login\_info["identity"],  
   "user\_taken": "False"  
   }  
    
   client.publish("application/signup", json.dumps(returning\_data))  
    
   if msg.topic == "server/login":  
   json\_login\_info = json.loads(msg.payload.decode("utf-8"))  
   print(f"Received message: {json\_login\_info}")  
    
   success = authenticate\_user(json\_login\_info["username"].lower(), json\_login\_info["password"])  
   data\_to\_return = {  
   "identity": json\_login\_info["identity"],  
   "success": success  
   }  
    
   client.publish("application/login", json.dumps(data\_to\_return))  
    
   if msg.topic == "server/create":  
   data = json.loads(msg.payload.decode("utf-8"))  
   print(f"Received message: {data}")  
   username = data["username"].lower()  
   recipient\_username = data["recipient"].lower()  
   chatroom\_code = data["code"]  
   topic = f"application/chatrooms/{chatroom\_code}"  
    
   try:  
   execute\_in\_database(  
   "INSERT INTO chatroom\_data (chatroom\_code) VALUES (%s);",  
   (chatroom\_code,)  
   )  
   recipient\_available = username\_exists(recipient\_username)  
   except psycopg2.IntegrityError as e:  
   if "unique constraint" in str(e):  
   data\_for\_creator = {  
   "username": username,  
   "recipient": recipient\_username,  
   "topic": topic,  
   "code\_available": "False"  
   }  
    
   data\_for\_recipient = {  
   "username": recipient\_username,  
   "recipient": username,  
   "topic": topic,  
   "code\_available": "False"  
   }  
    
   client.publish("application/create", json.dumps(data\_for\_creator))  
   client.publish("application/create", json.dumps(data\_for\_recipient))  
   else:  
   data\_for\_creator = {  
   "username": username,  
   "recipient": recipient\_username,  
   "topic": topic,  
   "code\_available": "True",  
   "recipient\_available": recipient\_available  
   }  
    
   data\_for\_recipient = {  
   "username": recipient\_username,  
   "recipient": username,  
   "topic": topic,  
   "code\_available": "True",  
   "recipient\_available": recipient\_available  
   }  
    
   client.publish("application/create", json.dumps(data\_for\_creator))  
   client.publish("application/create", json.dumps(data\_for\_recipient))  
    
    
  def on\_connect\_for\_create\_chatroom(client, userdata, flags, rc, properties=None):  
   if rc == 0:  
   print("Connected to MQTT broker.")  
   else:  
   print("Failed to connect to MQTT broker.", rc)  
    
    
  def on\_message\_for\_create\_chatroom(client, userdata, msg):  
   print("Received a message on topic ", msg.topic)  
    
    
  def execute\_in\_database(query, params=None):  
   conn = psycopg2.connect(  
   dbname="healthsync",  
   user="postgres",  
   password="12345",  
   host="localhost",  
   port="5432"  
   )  
   cursor = conn.cursor()  
   cursor.execute(query, params)  
   conn.commit()  
   conn.rollback()  
   cursor.close()  
   conn.close()  
    
    
  broker = "4dbbebee01cb4916af953cf932ac5313.s1.eu.hivemq.cloud"  
  port = 8883  
  username = "Reader"  
  password = "Reader123"  
    
  client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)  
  client.username\_pw\_set(username, password)  
  client.tls\_set()  
  client.on\_connect = on\_connect  
  client.on\_message = on\_message  
  client.connect(broker, port)  
    
  client.loop\_start()  
    
  try:  
   while True:  
   pass  
  except KeyboardInterrupt:  
   print("Exiting...")  
  finally:  
   client.loop\_stop()  
   client.disconnect()
* **Functions.py**
* import psycopg2  
  import bcrypt  
    
    
  def hash\_password(password):  
   return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')  
    
    
  def username\_exists(username):  
   try:  
   conn = psycopg2.connect(  
   dbname="healthsync",  
   user="postgres",  
   password="12345",  
   host="localhost",  
   port="5432"  
   )  
   cursor = conn.cursor()  
    
   query = "SELECT 1 FROM user\_data WHERE username = %s;"  
   cursor.execute(query, (username,))  
    
   result = cursor.fetchone()  
    
   return result is not None  
   except Exception as e:  
   print(f"Error checking username: {e}")  
   return False  
   finally:  
   if cursor:  
   cursor.close()  
   if conn:  
   conn.close()  
    
    
  def authenticate\_user(username, password):  
   try:  
   conn = psycopg2.connect(  
   dbname="healthsync",  
   user="postgres",  
   password="12345",  
   host="localhost",  
   port="5432"  
   )  
   cursor = conn.cursor()  
    
   query = "SELECT password FROM user\_data WHERE username=%s;"  
   cursor.execute(query, (username,))  
   result = cursor.fetchone()  
    
   if result is None:  
   return False  
    
   stored\_hashed\_password = result[0]  
   print(stored\_hashed\_password)  
    
   return bcrypt.checkpw(password.encode('utf-8'), stored\_hashed\_password.encode('utf-8'))  
    
   except Exception as e:  
   print(f"Error during authentication: {e}")  
   return False  
   finally:  
   if cursor:  
   cursor.close()  
   if conn:  
   conn.close()