

RELAZIONE PROGETTO

Reti di Calcolatori



Autore:

Sessione Estiva 2021/22

MQTT-CHAT ROOM

1.Introduzione

1.1 Specifica di progetto

Si è deciso di implementare una Chat Room che utilizzi il protocollo MQTT e che sia provvista di una minimale interfaccia grafica che permetta all'utente di inserire il testo e visualizzare i vari messaggi all'interno della chat.

Nello specifico, l'applicazione avrà le seguenti caratteristiche:

- Scegliere un proprio nickname
- Inserire e visualizzare messaggi, sia i propri che degli altri utenti
- Presenza di un admin che potrà utilizzare i comandi "espelli" (kick) e "banna" (ban). Ovviamente, per ricoprire tale ruolo, bisognerà essere in possesso di una determinata password.

Nota: La password stabilita per questo progetto è "admin".

1.2 Motivazione della scelta e linguaggio utilizzato

La scelta di questa specifica è stata fatta sia per la presenza di argomenti inerenti al corso di Reti di Calcolatori (ad esempio, il protocollo MQTT si basa su TCP/IP) sia per prendere conoscenza del protocollo MQTT e del suo utilizzo.

Inoltre, è stato utilizzato python come linguaggio di programmazione ed è stato scelto per la sua semplicità e per le numerose librerie messe a disposizione per agevolare la costruzione del software.

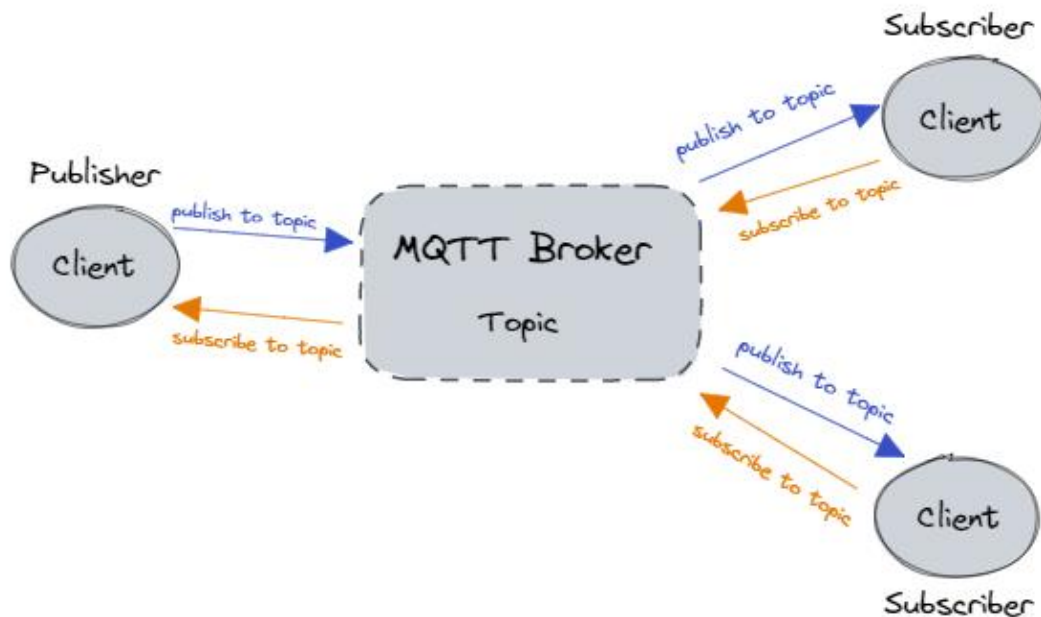
2. Discussione della tematica

Verranno illustrate le scelte progettuali prese per la realizzazione della Chat Room.

2.1 Protocollo MQTT

La scelta di progetto su cui si basa l'applicativo è sicuramente l'utilizzo del protocollo MQTT. Quest'ultimo è un protocollo binario e molto leggero, il quale risulta estremamente facile da implementare sul lato client. In più, è basato su TCP/IP (sia il client che il broker devono disporre di uno stack TCP/IP).

Nello specifico, esso sfrutta il modello di publish/subscribe (può essere visto come un'alternativa al modello client-server) che disaccoppia il client che invia un messaggio (il publisher) dal client o dai client che ricevono i messaggi (i subscribers). Inoltre, i publisher e i subscribers non si contattano mai direttamente. Infatti, la connessione tra loro è gestita da un terzo componente (il broker) che filtra tutti i messaggi in arrivo e li distribuisce correttamente ai subscribers.



Proprio questo disaccoppiamento del mittente dal destinatario è l'aspetto più importante di questo modello e presenta le seguenti caratteristiche:

- Space decoupling: Publisher e Subscriber non hanno bisogno di conoscersi
- Time decoupling: Publisher e Subscriber non devono essere eseguiti contemporaneamente.
- Synchronization decoupling: le operazioni su entrambi i componenti non devono essere interrotte durante la pubblicazione o la ricezione.

Inoltre, è anche chiaro che il Broker svolge un ruolo fondamentale all'interno del modello Pub/Sub e per far sì che ogni Subscribers riceva solo messaggi di interesse, ha a disposizione diverse strategie di filtraggio:

- SUBJECT-BASED FILTERING: basato sul topic /argomento che fa parte di ogni messaggio. Il Client si iscrive al Broker con il topic di interesse e da quel momento in poi riceverà tutti i messaggi pubblicati su quell'argomento.
(La Chat Room realizzata utilizza questa opzione di filtraggio dei messaggi)
- CONTENT-BASED FILTERING: basato sul contenuto, il broker filtra il messaggio in base a uno specifico linguaggio di filtraggio del contenuto. I client ricevuti si iscrivono alle query di filtraggio dei messaggi a cui sono interessati. Un aspetto negativo di questo metodo è che il contenuto del messaggio deve essere noto in anticipo e non può essere criptato o facilmente modificato.
- TYPE-BASED FILTERING: quando si utilizzano linguaggi orientati agli oggetti, il filtraggio basato sul tipo/classe di un messaggio (evento) è una pratica comune.

In aggiunta, è bene notare che il modello Pub/Sub ha una migliore scalabilità rispetto al Client/Server in quanto le operazioni sul broker possono essere altamente parallelizzate e i messaggi possono essere elaborati in modo event-driven (ovvero, quando un sistema registra un evento, aggiunge semplicemente una nota a una coda di messaggistica).

All'atto pratico, nel progetto, questo protocollo è stato implementato tramite la libreria paho-mqtt per python, sfruttando un managed broker (broker remoto), ovvero "mqtt.eclipseprojects.io", e i vari Client si iscrivono al topic "ChatRoom".

Ovviamente, è necessario che le istruzioni seguano un certo ordine di esecuzione in modo tale che la Chat Room funzioni in modo adeguato grazie alla libreria paho-mqtt. Infatti, bisognerà:

1. Creare funzioni di callback (saranno illustrati in seguito)

```
# Definizione della funzione on_connect
def on_connect(client, userdata, flags, rc):
```

2. Istanziare un oggetto Client

```
# Istanza oggetto Client
client = mqtt.Client(client_id="", clean_session="True")
```

3. Assegnare la funzione a quella di callback

```
# Associazione delle funzioni alle callback di paho-mqtt
client.on_connect = on_connect
client.on_message = on_message
```

4. Connettersi al Broker

```
# Connessione al Broker
client.connect(HOST, PORT)
```

5. Eseguire il client

```
# Esecuzione del client
client.loop_start()
```

Oltre a ciò, è bene specificare che il Client Paho è progettato per utilizzare le funzioni di callback (se esistono), ovvero funzioni che vengono chiamate in risposta a un evento, ma non ne fornisce alcuna predefinita. Tuttavia, fornisce un meccanismo per impostarli e per fare ciò bisogna:

- Creare una funzione di callback
- Assegnare la funzione appena creata a quella di callback di paho.

Un esempio, preso dal codice, può essere la funzione `on_message()`.

```
# Definizione della funzione on_message
def on_message(client, user_data, msg):
    global nickname
    decrypted_message = cipher.decrypt(msg.payload)
    message = decrypted_message.decode("utf-8")

    # Verifica della presenza ed esecuzione del comando "espelli"
    if message.find('/espelli') >= 0:
        write_onscreen(message)
        user = message.partition('/espelli ')[2]
        if user.strip('\n') == nickname:
            # L'utente viene disconnesso
            disconnection("kick")
    # Verifica della presenza ed esecuzione del comando "banna"
    elif message.find('/banna') >= 0:
        write_onscreen(message)
        user = message.partition('/banna ')[2]
        if user.strip('\n') == nickname:
            # L'utente viene disconnesso
            disconnection("ban")
    else:
        write_onscreen(message)
```

Viene creata la funzione.

Qui, eseguiamo l'associazione.

```
client.on_message = on_message
```

Grazie a ciò, ogni volta che un messaggio viene ricevuto (evento), la funzione entra in azione ed esegue ciò che si trova al suo interno.

Nel codice, viene fatto uso della funzione di callback:

- `on_connection()` = in risposta all'evento generato dalla connessione al broker (`client.connect(HOST, PORT)`)
- `on_message()` = in risposta all'evento generato dalla ricezione di un messaggio

2.2 Crittografia del Payload

Come precisato all'inizio, MQTT si basa sul protocollo di trasporto TCP. Per impostazione predefinita, le connessioni TCP non utilizzano una comunicazione crittografata. Per questo motivo si è deciso di non trasmettere "in chiaro" il payload (ovvero, il contenuto effettivo

del messaggio) a vantaggio della sicurezza in quanto i dati vengono crittografati in maniera end-to-end e non solo tra Broker e Client.

Per implementare ciò, è stato utilizzato un pacchetto di crittografia supportato da Python e che, grazie al suo modulo “Fernet”, genera una chiave con la quale riesce a cifrare e decifrare il testo.

Più precisamente, la chiave è stata generata esternamente e poi inserita nel codice tramite la variabile *cipher_key* in quanto tutti i Client devono utilizzare la stessa chiave di decifratura. Inoltre, l’effettiva locazione di quest’ultima si trova all’interno di un file diverso (settings.py) da quello principale che contiene tutto il codice.

Di seguito verranno spiegati i vari step che descrivono l’esecuzione del metodo di crittografia.

1. Il codice importa la chiave dal file in cui è memorizzata e la utilizza per la cifratura e decifratura dei messaggi.

```
from settings import cipher_key  
cipher = Fernet(cipher_key)
```

2. Il messaggio da cifrare deve essere in bytes, perciò viene trasformato.

```
send_message = bytes(send_message, encoding='utf8')
```

3. Il messaggio da inviare viene effettivamente cifrato

```
encrypted_message = cipher.encrypt(send_message)
```

4. Bisogna creare una stringa codificata UTF-8 da passare come payload del messaggio al metodo MQTT publish.

```
out_message = encrypted_message.decode()  
# Il messaggio viene pubblicato  
client.publish(ROOM, out_message)
```

5. Nella funzione “on_message”, il payload viene decifrato

```
decrypted_message = cipher.decrypt(msg.payload)
```

6. Infine, il messaggio viene codificato in UTF-8

```
message = decrypted_message.decode("utf-8")
```

2.3 Password dell’Admin

Per rivestire il ruolo di admin della chat, bisogna inserire una password. Per semplicità, la password stabilita per questo progetto è “admin” e il controllo di quest’ultima avviene localmente, prima della connessione al Broker. Inoltre, per ragioni di sicurezza, essa ha subito un processo di “hashing” per essere memorizzata in un file esterno (lo stesso di dove si trova la chiave di cifratura – settings.py).

Si ricorda che l’hashing è un processo unidirezionale di protezione delle password di testo semplice mediante la creazione di una stringa di bit di dimensioni fisse, chiamata hash,

utilizzando una funzione hash crittografica (funzioni di questo genere sono progettate per essere unidirezionali). In questo progetto, è stato utilizzato lo schema di hashing *Argon2*.

Nello specifico, una volta che l'utente digita la password, viene importata e verificata l'uguaglianza con quella che ha subito il processo di hashing. Ovviamente, per fare ciò, sia l'hashing della password che il metodo di verifica devono avere degli stessi parametri riguardanti il processo di cifratura. Esse sono:

```
# Definizione del PasswordHasher
argon2Hasher = argon2.PasswordHasher(
    time_cost=3,          # numero di iterazioni
    memory_cost=64 * 1024, # 64mb
    parallelism=1,        # thread paralleli da utilizzare
    hash_len=32,          # dimensione della chiave derivata
    salt_len=16           # dimensione del salt casuale generato in byte
)
```

La verifica viene eseguita attraverso la funzione “argon2Hasher.verify” che ha come parametri le due password e restituisce il valore booleano True se riscontra l'uguaglianza delle due stringhe.

2.4 Ulteriori scelte progettuali

Per tenere traccia degli utenti bannati, il programma utilizza un semplice file “.txt” (bans.txt).

Inoltre, per facilitare l'utilizzo dei comandi speciali da parte dell'admin, si è deciso di rendere univoco il nickname con cui ogni utente sarà riconosciuto all'interno della chat. Per fare ciò, viene utilizzato un altro file “.txt” (online.txt) con cui tenere traccia degli utenti online e controllare che nessuno acceda alla chat con un nickname già utilizzato (di conseguenza, ci sarà anche un solo admin). Ovviamente, ogni qual volta che un utente si disconnette, il suo nickname viene cancellato dal file.

Infine, è stato scelto di gestire i vari casi di disconnessione attraverso un'unica funzione “disconnect(flag)”, che in base al motivo per cui ciò avviene, viene stampato sullo schermo della chat uno specifico messaggio (alcuni visibili a tutti, altri solo all'utente).

3. Conclusioni

In conclusione, il software offre tutte le funzionalità poste come obiettivo iniziale con l'aggiunta di alcuni meccanismi per cercare di migliorare la sicurezza della Chat Room (si vedano la cifratura dei messaggi e della password).

Una nota critica del progetto è l'assenza di poter vedere i messaggi inviati precedentemente alla connessione dell'utente alla chat e ciò è dovuto anche al funzionamento del protocollo MQTT. Però costruire un qualche meccanismo che salvasse lo storico dei messaggi avrebbe complicato inutilmente il progetto che si sarebbe allontanato dall'utilizzo stesso del protocollo MQTT sia dalla specifica.

4. Installazione pacchetti e utilizzo del programma

Se non presenti già sul proprio computer, è necessario installare i seguenti pacchetti:

Protocollo MQTT: **pip install paho-mqtt**

Schema hashing Argon2: **pip install argon2**

Crittografia cryptography.fernet: **pip install cryptography**

L'utente che vorrà rivestire il ruolo di admin dovrà connettersi con il nickname "admin" e la password "admin".

Utilizzo dei comandi speciali "espelli" e "banna", utilizzabili dall'admin:

Per espellere un utente, digitare */espelli <nickname utente>*

Per bannare un utente, digitare */banna <nickname utente>*

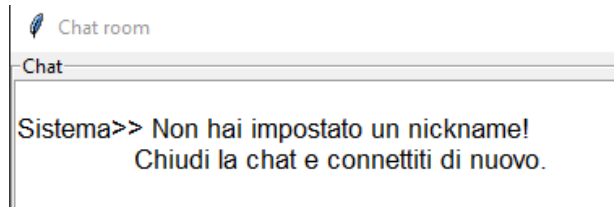
Nota: bisogna lasciare uno spazio tra il comando e il nickname. Solo in questo modo l'utente verrà effettivamente bannato o cacciato dalla chat.

Per chiudere la chat e disconnettersi dal Broker, basterà cliccare sull'icona "Chiudi" (X) presente nella barra del titolo.

5. Testing

Di seguito verranno visualizzati alcuni casi di testing dell'applicativo

5.1 Non è stato inserito nessun nickname

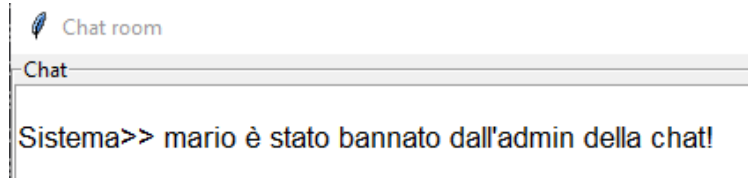


Chat room

Chat

Sistema>> Non hai impostato un nickname!
Chiudi la chat e connettiti di nuovo.

5.2 Un utente bannato prova ad entrare nella chat

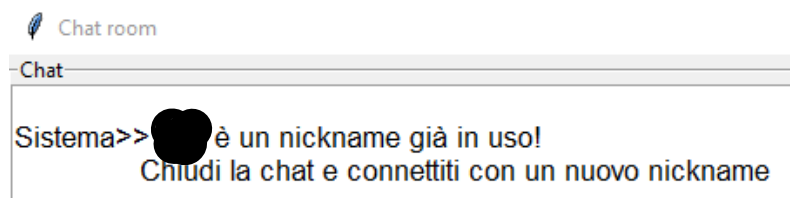


Chat room

Chat

Sistema>> mario è stato bannato dall'admin della chat!

5.3 Nickname già in uso

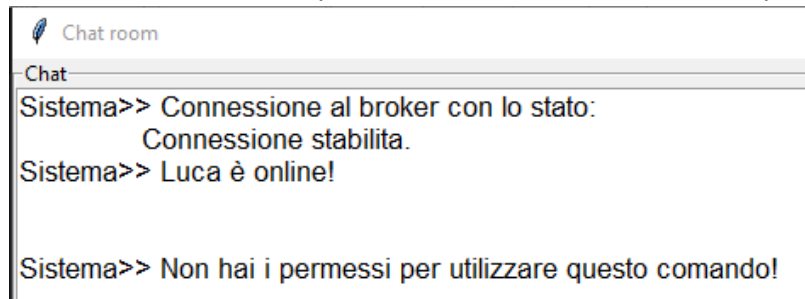


Chat room

Chat

Sistema>> [redacted] è un nickname già in uso!
Chiudi la chat e connettiti con un nuovo nickname

5.4 Un utente non admin prova a utilizzare un comando speciale



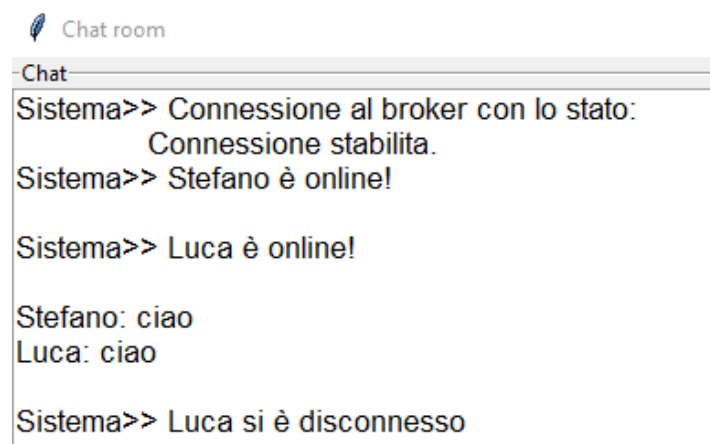
Chat room

Chat

Sistema>> Connessione al broker con lo stato:
Connessione stabilita.
Sistema>> Luca è online!

Sistema>> Non hai i permessi per utilizzare questo comando!

5.5 Un utente si disconnette e tutti gli altri partecipanti ricevono una notifica



Chat room

Chat

Sistema>> Connessione al broker con lo stato:
Connessione stabilita.
Sistema>> Stefano è online!

Sistema>> Luca è online!

Stefano: ciao
Luca: ciao

Sistema>> Luca si è disconnesso

6. Bibliografia:

Documentazione MQTT: [HiveMQ Introduction :: HiveMQ Documentation](#)

Documentazione paho-mqtt: [paho-mqtt · PyPI](#)

Argon2: [Argon2 Hash Generator, Validator & Verifier](#)

Fernet encryption: [Fernet \(symmetric encryption\) — Cryptography 38.0.0.dev1 documentation](#)

Altri siti utilizzati per la realizzazione del progetto:

YouTube, Stack-overflow