Advanced Programming Languages

MeetingApp

Capodicasa Francesco – Giuffrida Giuseppe

Link al repository su GitHub:
https://github.com/giuseppegiuffry/MeetingApp

Introduzione

Si è pensato di realizzare un'applicazione che permetta a degli utenti di comunicare con persone a loro compatibili, tramite una chat. Lo scopo è quindi appunto quello di far conoscere altra gente con interessi simili e magari, se lo si desidera, incontrarsi successivamente.

Inizialmente, bisogna registrare il proprio account inserendo i propri dati e interessi in un apposito form; successivamente, si può procedere con il login inserendo username e password precedentemente impostati.

I dati inseriti per la registrazione e quelli per il login verranno memorizzati in un database SQLite, rispettivamente all'interno delle tabelle "user" e "account".

Una volta effettuato il login da parte dell'utente, questo potrà iniziare a chattare con altri utenti potenzialmente a lui compatibili, tramite una semplice interfaccia grafica in cui saranno mostrati tutti i vari messaggi scambiati nel corso della conversazione.

Server

Il server è stato realizzato utilizzando il linguaggio Python. Si avvia da linea di comando tramite uno script chiamato *Server.py.*

Questo non fa altro che stare in ascolto delle richieste di connessione provenienti dai vari client. Per ciascuna richiesta viene instanziato un nuovo thread che richiama il metodo *clientThread* per gestire la richiesta.

Questo metodo ha la funzione di interfacciarsi con il database SQLite e sulla base del tipo di operazione effettuata dal client, stampa in output un messaggio in formato JSON con le relative informazioni.

A seconda infatti del tipo di operazione (login, registrazione, messaggio inviato tramite chat), verrà costruito un tipo di messaggio specifico, in formato JSON, che poi verrà mostrato appunto dal server.

Il server inoltre provvederà a monitorare lo stato del/i client connesso/i, informando su quando questo si connetterà/disconnetterà, attraverso sempre la stampa di messaggi di avviso sul terminale.

Per effettuare il matching tra due utenti compatibili, è stata realizzata una funzione, chiamata *matching*, che sfrutta la suite di librerie di NLTK per algoritmi di stemming. Questa funzione, in un primo momento, calcola le radici di tutte le parole presenti nel campo biografia dell'utente loggato e del possibile utente con cui vogliamo chattare; successivamente, fa un confronto tra tutte le parole così ottenute nelle due biografie e restituisce un valore tra 0 e 100 sulla base di quanto le parole confrontate siano simili (0 → completamente diverse, 100 → parole uguali). Se il valore restituito è maggiore di una soglia impostata, nel nostro caso 60, allora significa che c'è un numero di parole simili abbastanza elevato e quindi i due utenti potranno iniziare a scambiarsi messaggi via chat.

Per la comunicazione tra il server e i vari client a esso connessi, sono state utilizzate delle socket TCP, sempre in linguaggio Python, per ottenere comunicazione full duplex.

Client

Il Client è stato implementato utilizzando QT Creator, un IDE che utilizza librerie basate su C++. La scelta dell'utilizzo di QT (detto "cutie") è derivata dal bisogno di implementare una o più interfacce grafiche di una certa complessità, così da dare all'utente finale un utilizzo pratico e veloce della stessa applicazione.

Il Client è formato da quattro file .cpp, con i loro header, escludendo il *main.cpp*, utilizzato solo per avviare la finestra principale.

Tali file sono:

1. *login.cpp*: definita anche come *MainWindow*, è la prima finestra che viene aperta all'avvio del programma. Questa mostra l'interfaccia per eseguire il login al sistema, e quindi cominciare a messaggiare, oppure offre la possibilità di registrarsi tramite l'apposito bottone.

2. *home.cpp*: questa è la finestra utilizzata per interfacciarsi col mondo esterno. Se il server ha trovato qualche tipo di match, allora l'utente vedrà l'arrivo dei vari messaggi a lui destinati, e quest'ultimo potrà poi a sua volta rispondere.

3. *registration.cpp*: finestra utilizzata per la registrazione dell'utente al sistema, dove vanno inseriti tutti i campi necessari per costruire il proprio profilo.

4. *socket.cpp*: componente C++ che deriva da *QTcpSocket,* creato esclusivamente per avere un maggior controllo sulle socket utilizzate dall'applicazione, implementando funzioni aggiuntive con scopi specifici per questo tipo di applicativo.

La comunicazione si basa sull'utilizzo di socket, scritte in linguaggio C++, implementando quindi delle socket TCP. Inoltre, tutti i dati mandati attraverso queste socket sono in formato JSON.

Per la ricezione dei messaggi, è stata creata una funzione apposita, *OnReadyRead()*, che viene attivata quando abbiamo il segnale *readyread()*, cioè quando il buffer di ricezione della socket ha qualcosa da leggere. In questo modo, riusciamo a riconoscere che tipo di JSON sia arrivato e quindi a rispondere correttamente.

Infine, la comunicazione tra le varie finestre viene implementata tramite la funzione *connect*, che crea una connessione tra il segnale del mittente e lo slot del destinatario.