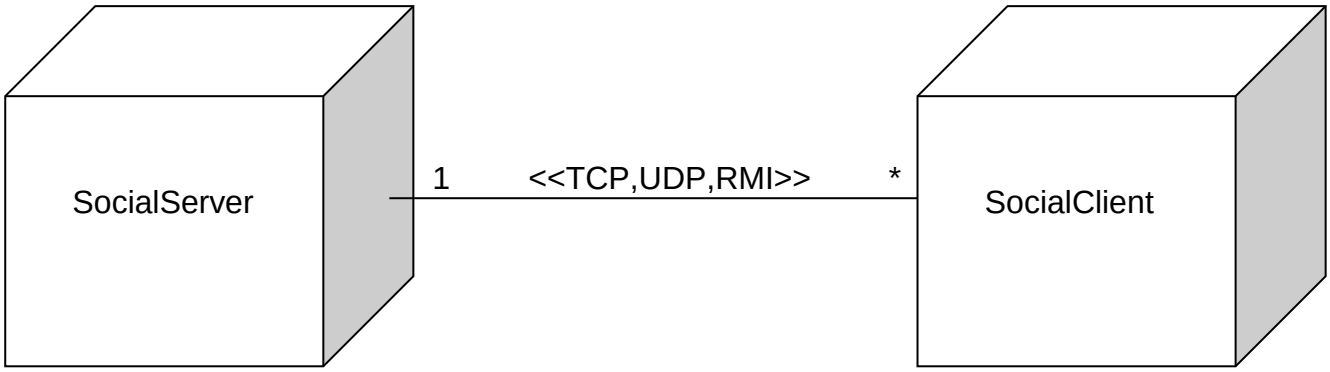
Relazione Online Social Network

Caterina Falchi

Questa relazione illustra l'architettura del sistema, le scelte che sono state effettuate nell'implementazione del progetto, e le classi usate.

ARCHITETTURA

L'architettura del sistema è molto semplice: ci sono un server e molti client che comunicano con TCP, UDP e RMI.



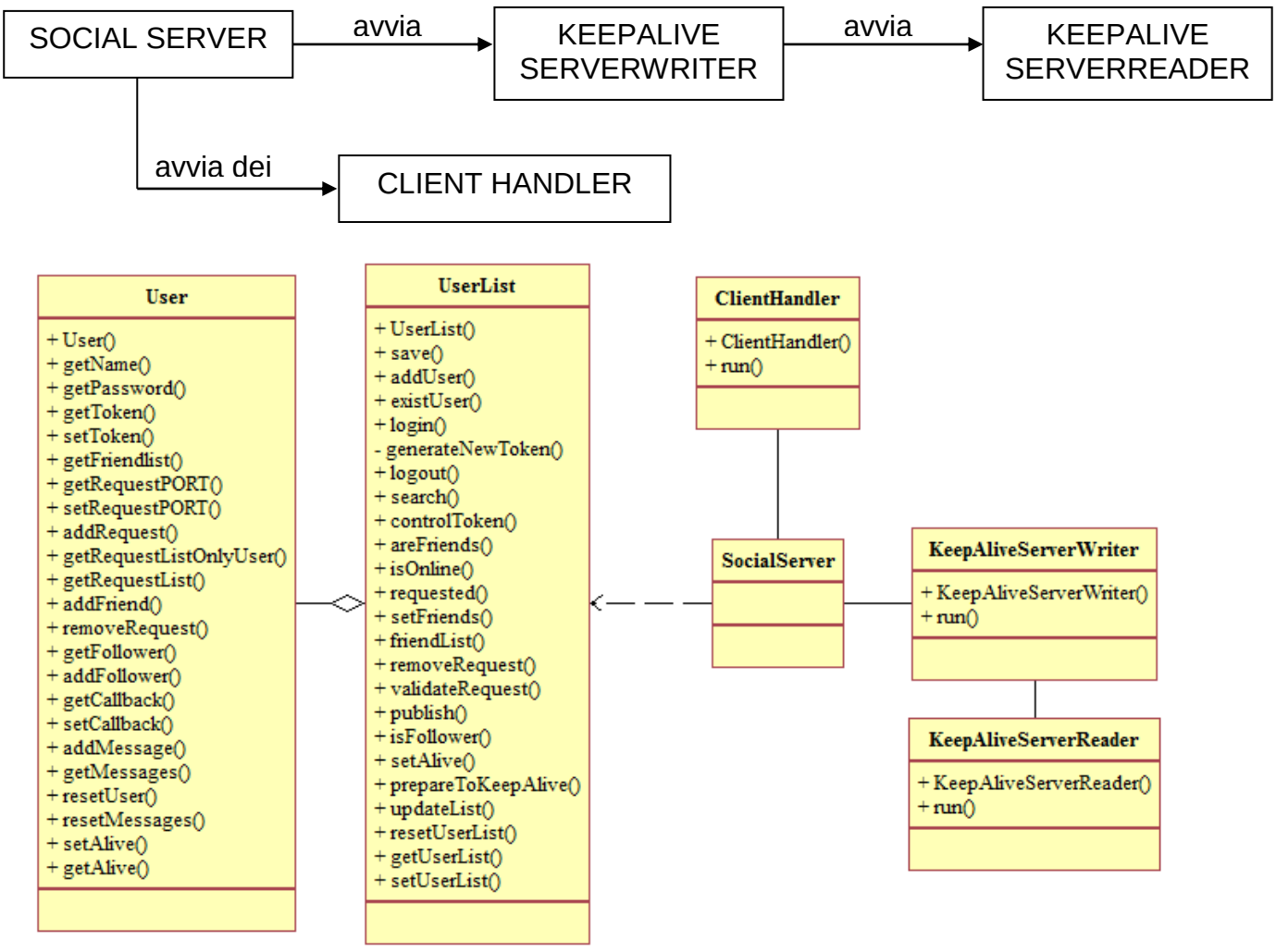
SERVER – Spiegazione delle classi e della logica utilizzata

Il Package del server si compone di:

- SocialServer: è la classe con il main del server. Carica le costanti dal file di configurazione, si occupa del caricamento da file della lista utenti (se esiste), effettua le inizializzazioni della parte di comunicazione con RMI, avvia il thread per i messaggi di KeepAlive. Quindi entra in un ciclo infinito dove ad ogni richiesta che gli arriva dai client crea un handler per rispondere. Utilizza normali socket bloccanti.
- ClientHandler: a seconda del tipo di pacchetto ricevuto dal client, l'handler crea una risposta da mandargli e cambia la lista degli utenti. Utilizza normali socket bloccanti. Per le operazioni di registrazione, login, logout, e richiesta di messaggi il token non viene controllato in quanto non serve; per tutte le altre viene invece controllato.
- KeepAliveServerWriter: è il thread che manda i messaggi di KeepAlive e in base alle risposte che riceve aggiorna la lista utenti, settando offline quelli che non hanno risposto. Ovviamente non riceve lui direttamente le risposte: prima di entrare nel ciclo dove manda i messaggi, avvia un thread (KeepAliveServerReader) che riceverà le risposte. Utilizza MulticastSocket.
- KeepAliveServerReader: è il thread che riceve le risposte di KeepAlive degli utenti. Utilizza DatagramSocket. La strategia usata per tenere traccia delle risposte è la seguente: KeepAliveServerWriter prepara la lista utenti mettendo i flag alive di ogni utente a false, manda il messaggio in multicast e dorme 10 secondi. Nel frattempo KeepAliveServerReader riceve i messaggi con l'username di ogni utente che ha risposto: per ogni risposta setta il flag alive a true. Quando poi KeepAliveServerWriter si sveglia, aggiorna la lista degli utenti mettendo offline tutti coloro che risultavano online (token non nullo) ma che non hanno risposto (alive=false).
- User: è la classe usata per memorizzare tutto ciò che riguarda un utente: username, password, token (se l'utente è online è una stringa generata casualmente, altrimenti è nullo), porta sulla quale l'utente aspetta le richieste di amicizia, lista amici, lista richieste di amicizia, lista follower, lista di messaggi, flag

alive. Le liste usate sono tutte CopyOnWriteArrayList per non avere problemi riguardo la concorrenza.

- UserList: è la struttura dati usata per tenere traccia degli utenti registrati e online. Contiene inoltre tutti i metodi che permettono al server di soddisfare le richieste degli utenti (come per esempio rendere amici due utenti). La lista di utenti è materialmente realizzata con una ConcurrentHashMap<String,User> dove la chiave è l'username utente. Poteva essere implementata anche un'altra soluzione dove venivano create una lista di utenti registrati e una di utenti online, ma mi sembrava più preciso avere una lista sola. Nella soluzione usata un utente è considerato online se il token è diverso da null, è offline se è null.
  Questa classe contiene anche il metodo che si occupa del salvataggio della lista utenti (realizzato tramite serializzazione della lista suddetta) nel file userList.dat che si trova nella cartella File. Il metodo viene richiamato ogni qualvolta si effettua un'operazione che modifica la lista utenti.
  Un altro metodo importante è validateRequest che serve per aggiornare la lista di richieste di amicizia di un utente, rimuovendo quelle non più valide (è trascorso il TIMEFRIENDREQUEST). La validateRequest viene richiamata ogni qualvolta un utente vuole confermare o cancellare una richiesta di amicizia.



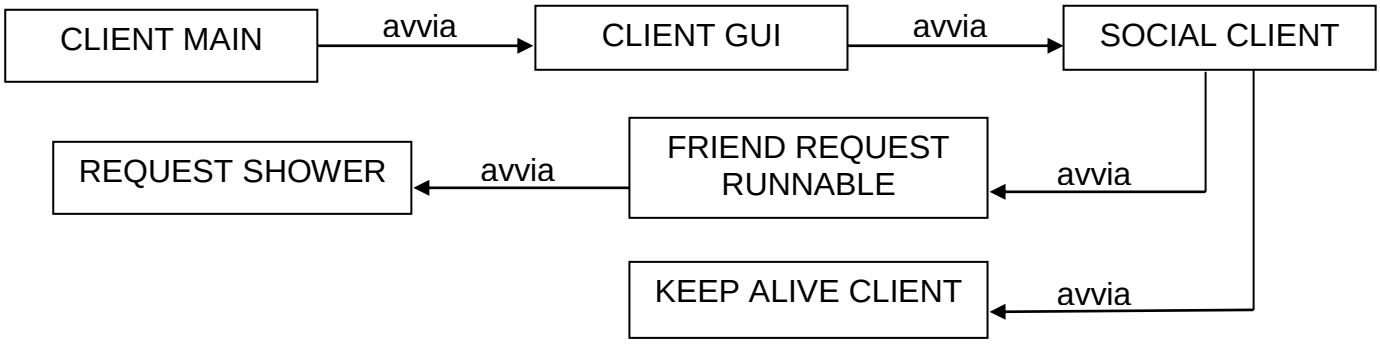
CLIENT

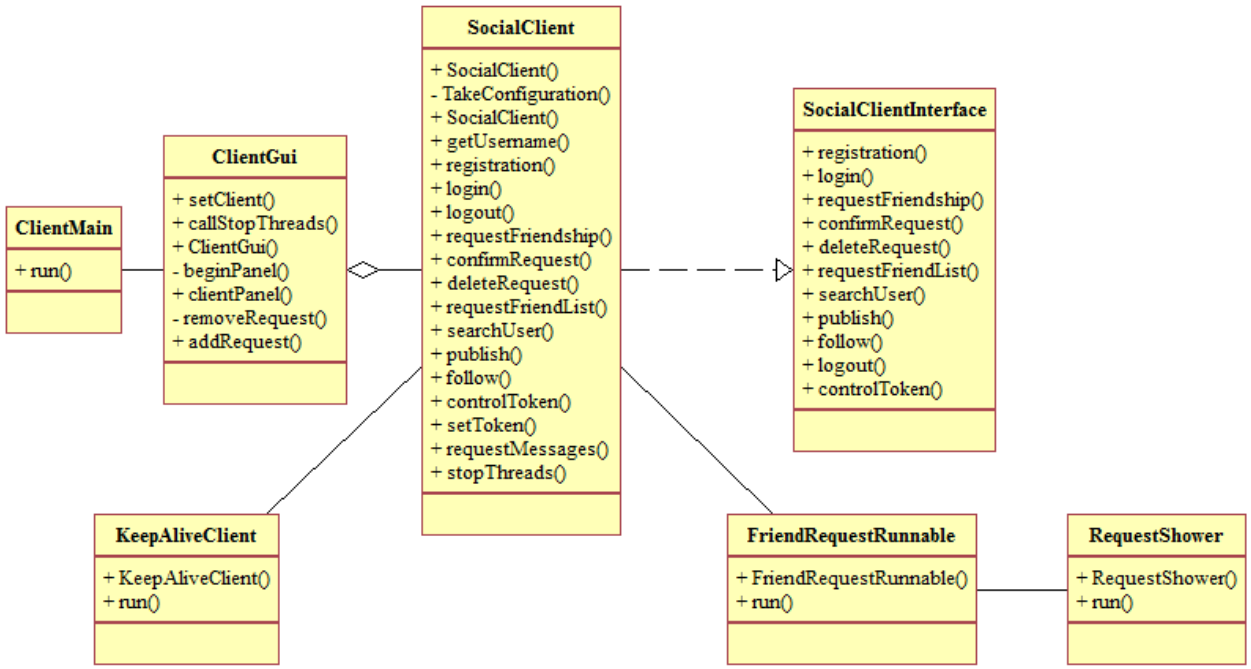
Il Package del client si compone di:

- ClientMain: è il main del client, che permette quindi di avviarlo
- ClientGui: è l'interfaccia grafica grazie alla quale l'utente può usufruire dei servizi offerti dal server e fare richieste. È composta da due pannelli: uno iniziale dove ci si può iscrivere al social network o loggarsi, e un altro che appare una volta che l'utente è loggato e gli permette di fare azioni. L'interfaccia ha un SocialClient che al momento del login viene associato all'interfaccia e viene dissociato al momento del logout o della chiusura dell'interfaccia. Ogni volta che viene premuto un bottone nell'interfaccia, viene richiamata la funzionalità apposita del SocialClient associato e viene visualizzata una risposta. L'interfaccia contiene una JList per le richieste di amicizia che arrivano all'utente che merita attenzione in quanto ho creato due metodi appositi per la rimozione di richieste visualizzate e l'aggiunta di richieste da visualizzare.
- SocialClientInterface: è semplicemente l'interfaccia che descrive i metodi del SocialClient.
- SocialClient: è il cuore della componente utente, usa socket bloccanti e RMI (dove richiesto dalla specifica). SocialClient memorizza alcuni pochi dati identificativi dell'utente, il token, il tempo in cui il token è stato assegnato (per poter controllare che sia ancora valido), la porta su cui l'utente aspetta richieste di amicizia, l'interfaccia grafica a cui è associato, i due componenti che gli servono per la parte di comunicazione con RMI. SocialClient ha due costruttori: uno usato quando si vuole registrare un utente, l'altro usato quando si vuole fare il login e le altre operazioni, infatti contiene anche l'esportazione dell'oggetto remoto del client e la ricerca di quello del server. Se il login di un utente va a buon fine, vengono fatte tre cose: creazione e avvio del thread che aspetta richieste di amicizia, creazione e avvio del thread che riceve e risponde ai messaggi di KeepAlive, registrazione della callback. Inoltre dall'interfaccia viene richiamata la funzionalità di richiesta messaggi con la quale si invia un pacchetto con TCP di tipo REQUESTMESSAGES per dire al server di mandare (tramite RMI) i messaggi degli utenti di cui si è follower che sono arrivato quanto l'utente era offline. In generale, ogni volta che l'utente deve mandare un messaggio al server per richiedere qualcosa, SocialClient crea un pacchetto di un certo tipo contenente il token (se necessario) e altri dati che servono per l'operazione, lo manda (TCP), attende una risposta che può essere un pacchetto di tipo RESPONSE (risposta semplice) con l'esito dell'operazione o TOKENERROR (se l'utente nella lista utenti del server ha token nullo). Una funzione interessante in questa classe è stopThreads(), ma ne parleremo più avanti.
- FriendRequestRunnable: è il Runnable che aspetta le richieste di amicizia che arrivano all'utente. Usa una ServerSocket per attendere connessioni, ed ha l'interfaccia associata al client per poter visualizzare le richieste a video. Si compone di un ciclo dove riceve semplicemente pacchetti, che gli possono arrivare
    - Dal server: in questo caso sono di tipo FRIENDREQUEST e contengono il nome dell'utente che vuole diventare amico
    - Dal SocialClient stesso che lo ha avviato, e in questo caso sono pacchetti di tipo STOP, e indicano che il thread deve terminare.
- RequestShower: ogni volta che arriva una richiesta di amicizia, FriendRequestRunnable invoca il thread RequestShower che la visualizza semplicemente nell'interfaccia associata al SocialClient che rappresenta l'utente

- KeepAliveClient: è il thread che si occupa della ricezione dei messaggi di KeepAlive in multicast, e della spedizione della risposta al server con DatagramSocket. In multicast può ricevere:
    - La stringa “Are you alive?” mandatagli dal KeepAliveServerWriter a cui deve rispondere con il proprio username su DatagramSocket
    - Un’altra stringa il cui mittente può essere qualsiasi SocialClient: se la stringa è uguale all’username del SocialClient dell’utente che ha avviato il thread, questo deve terminare, altrimenti ignora ciò che ha ricevuto.

La funzione stopThreads() nel SocialClient permette di far terminare il FriendRequestRunnable e il KeepAliveClient con la strategia già spiegata prima. Viene richiamata dopo che l’utente ha fatto logout con il bottone apposito, ed anche quando l’utente chiude l’interfaccia senza aver fatto logout. In quest’ultimo caso infatti, anche se l’utente non ha esplicitamente fatto logout, non essendoci più il thread che risponde ai messaggi di KeepAlive, il server non riceve più nulla e mette l’utente offline, come richiesto dalla specifica.





ALTRE CLASSI USATE

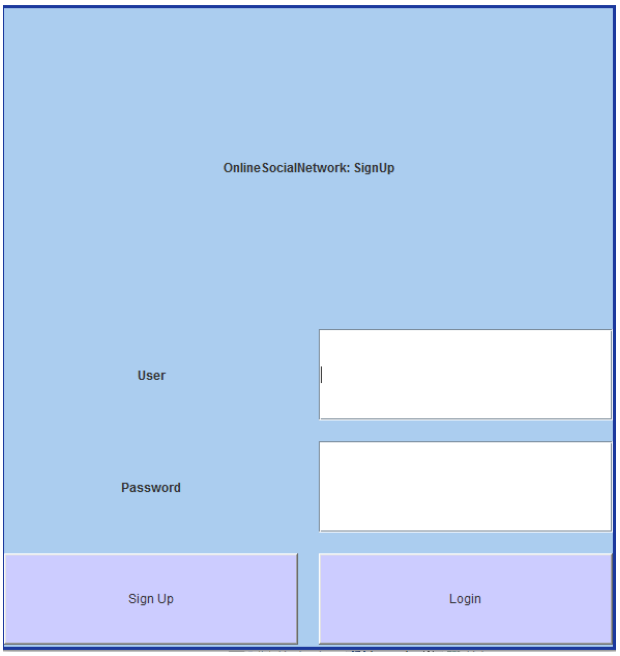
- Client Manager: contiene i metodi che gli utenti possono richiamare con RMI e che sono: registrazione della callback (che avviene al momento del login di un utente) e follow di un utente.
- ClientManagerImpl: è l'implementazione dell'interfaccia sopra descritta.
- CallbackInterface: racchiude i metodi che servono al server per interagire con i client: in questo caso abbiamo solo la sendMessage con la quale il server può mandare il messaggio di un mittente direttamente all'utente che è suo follower. Se il follower è online, viene immediatamente visualizzato a video il messaggio nello showcase, altrimenti viene memorizzato in una lista e quando il follower ne farà richiesta al momento del suo successivo login (con il pacchetto REQUESTMESSAGES), gli verrà mandato e verrà visualizzato.
- CallbackInterfaceImpl: è l'implementazione dell'interfaccia sopra descritta. Precisiamo che il metodo sendMessage contiene la sola istruzione che stampa nell'interfaccia il messaggio. La parte di verifica dello stato dell'utente e dell'inserimento del messaggio in lista o meno è effettuata dall'handler che richiama il metodo publish nella userList.
- Packet: i pacchetti sono ciò che i client e il server si scambiano con TCP. Ogni pacchetto è caratterizzato da un tipo, una stringa che è il token, e un'ArrayList di stringhe contenente i parametri per l'operazione richiesta al server, o il risultato dell'operazione richiesta dal client. Il tipo dei pacchetti può essere:
    - REGISTRATION: quando un utente vuole registrarsi
    - LOGIN: quando un utente vuole fare login
    - RESPONSE: quando il server ha un risultato da comunicare al client
    - LOGOUT: quando un utente vuole fare logout
    - SEARCH: quando un utente vuole fare una ricerca
    - FRIENDREQUEST: quando un utente vuole fare una richiesta di amicizia ad un altro
    - TOKENERROR: quando il server non trova riscontro nella lista utenti con il token che gli è stato mandato
    - CONFIRMREQUEST: quando un utente vuole confermare una richiesta di amicizia che ha ricevuto
    - DELETEREQUEST: quando un utente vuole cancellare una richiesta di amicizia che ha ricevuto
    - FRIENDLIST: quando un utente richiede la propria lista di amici
    - PUBLISH: quando un utente vuole pubblicare qualcosa
    - REQUESTMESSAGES: quando, dopo il login andato a buon fine, il SocialClient richiede al server se ci sono messaggi arrivati quando l'utente era offline
    - STOP: quando SocialClient necessita di fermare il thread FriendRequestRunnable.
- Request: è la classe usata per memorizzare le richieste di amicizia: contiene l'username dell'utente che ha mandato la richiesta e il tempo in cui questa è arrivata. Mi occorre memorizzare ciò per eliminare poi le richieste che non sono più valide e mantenere invece quelle valide (se ne occupa il server richiamando la validateRequest sulla lista utenti ogni volta che l'utente vuole accettare o cancellare una richiesta di amicizia).
- SocketStructure: è la classe che ho usato semplicemente per risparmiare istruzioni. Ho deciso di crearmi lì la socket, gli stream di input e output: in questo modo ogni volta che il client ha bisogno di comunicare col server crea un'istanza di questa classe, la usa e poi la richiude semplicemente con il metodo close().

ISTRUZIONI PER L'ESECUZIONE

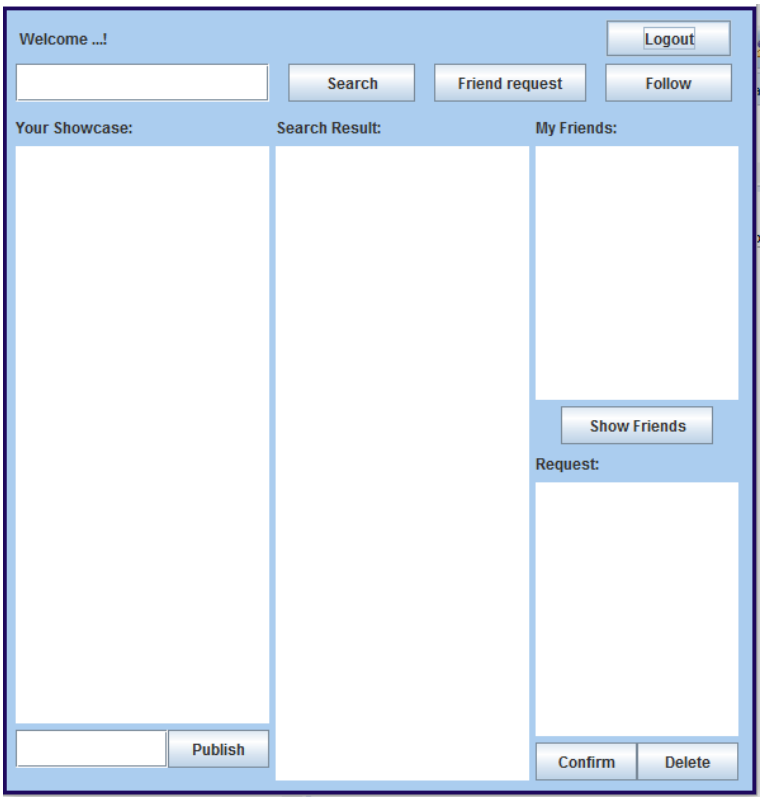
Per l'esecuzione del programma ci sono due soluzioni:

- Doppio clic sui jar
- Da linea di comando, posizionandosi nella directory contenente i jar con il comando: **java –jar SocialServer.jar** (per avviare il server) e **java –jar SocialClient.jar** (per avviare ogni client).

ISTRUZIONI PER L'USO DELL'INTERFACCIA GRAFICA



Il primo pannello (beginPanel) è quello rappresentato in figura.
Ho usato i layout soltanto per il fatto di volerli provare, in quanto era la prima volta che usavo Swing. Per registrarsi basta inserire username e password, e poi premere sul bottone "Sign Up". Per loggarsi occorre fare lo stesso, stavolta premendo il bottone di "Login". L'username non può contenere spazi e può essere lungo al massimo 14 caratteri.



Questo è il secondo pannello e appare dopo che un utente si è loggato. Per fare logout, molto semplicemente si preme sul bottone "Logout". Per fare una ricerca si digita nella JText in alto a sinistra ciò che si vuole cercare e si preme il tasto "Search": i risultati appariranno nella JTextArea centrale. Per richiedere amicizia ad un utente si digita il suo username sempre nella JText in alto a sinistra e si preme il bottone "Friend request": la richiesta verrà inoltrata al server che la manderà all'utente interessato (se questo è online) dove gli apparirà nella JList in basso a destra. Qui l'utente, selezionando direttamente l'username dalla lista, può scegliere se confermare o meno la richiesta premendo i bottoni

“Confirm” o “Delete”. Per diventare follower di un utente basta scrivere il suo username nella JText in alto a sinistra e premere il bottone Follower. Per vedere la lista dei propri amici, con il loro stato attuale (ONLINE – OFFLINE) basta premere il bottone “Show Friends” e la lista apparirà automaticamente nella JTextArea sopra il bottone stesso. La JTextArea a sinistra rappresenta la bacheca dell’utente: qui arriveranno i messaggi di tutti gli utenti da lui seguiti. Quando l’utente vuole pubblicare un messaggio deve scriverlo nella JText in basso a sinistra e premere il bottone “Publish”.