Sommario

[**1.** **INTRODUZIONE** 2](#_Toc31633092)

[**2.** **ARCHITETTURA** 2](#_Toc31633093)

[**2.1.** **Architettura Server** 2](#_Toc31633094)

[**2.1.1.** **Strutture Dati** 2](#_Toc31633095)

[**2.1.2.** **Threads** 2](#_Toc31633096)

[**2.1.3.** **Timers** 3](#_Toc31633097)

[**2.1.5.** **Schema di Funzionamento** 4](#_Toc31633098)

[**2.2.** **ARCHITETTURA CLIENT** 4](#_Toc31633099)

[**2.2.1.** **Strutture Dati** 5](#_Toc31633100)

[**2.2.2.** **Threads** 5](#_Toc31633101)

[**2.2.3.** **Interfaccia Grafica** 5](#_Toc31633102)

[**2.2.4.** **Schema delle Classi** 6](#_Toc31633103)

[**2.2.5.** **Schema di Funzionamento** 6](#_Toc31633104)

[**3.** **COMUNICAZIONE** 6](#_Toc31633105)

[**3.1.** **Comandi** 7](#_Toc31633106)

[**3.2.** **Risposte a Comandi** 7](#_Toc31633107)

[**3.3.** **Risposte Asincrone** 7](#_Toc31633108)

1. **INTRODUZIONE**

Con questo progetto si intende sviluppare un sistema client-server per gestire sfide di traduzione italiano-inglese tra gli utenti registrati al servizio.  
Gli utenti, una volta registrati, possono sfidare i propri amici in una gara di traduzione, il servizio implementerà quindi la gestione di una rete sociale tra gli utenti.

1. **ARCHITETTURA**

Il sistema è composto da un unico server al quale si connettono i client.  
Client e server comunicano tramite:

* + RMI: Per effettuare l’operazione di registrazione di un utente
  + UDP: Usato dal server per notificare la richiesta di sfida ad un utente
  + TCP: Per il resto delle comunicazioni  
    Inoltre dato che il sistema è progettato per operare in locale, si ha un problema di binding delle porte UDP per i client che quindi non possono essere statiche e che saranno inviate al server durante la procedura di login.
  1. **Architettura Server**

Il server è implementato senza interfaccia grafica utilizzando quindi la CLI.  
Tutte le impostazioni del server sono contenute nella classe Settings.java.  
Tutti i dati sugli utenti vengono mantenuti sul disco sotto forma di file JSON (utenti.json), all’avvio quindi il server verifica l’esistenza del file per caricare le informazioni e istanziare gli oggetti, altrimenti ne crea uno vuoto.  
Inoltre si occupa di caricare il file txt (dict.txt) contenente le parole che verranno utilizzate per le sfide.  
Poi viene inizializzato il servizio RMI sul quale espone il metodo di registrazione utente, il thread TCP che utilizzerà per le comunicazioni e la classe UDP per l’inoltro delle richieste di sfida.  
Infine il server permette i seguenti comandi da CGI:

* Effettua il salvataggio dello stato del server se si invia un numero>0.
* Salva lo stato e termina il server altrimenti.  
  + 1. **Strutture Dati**

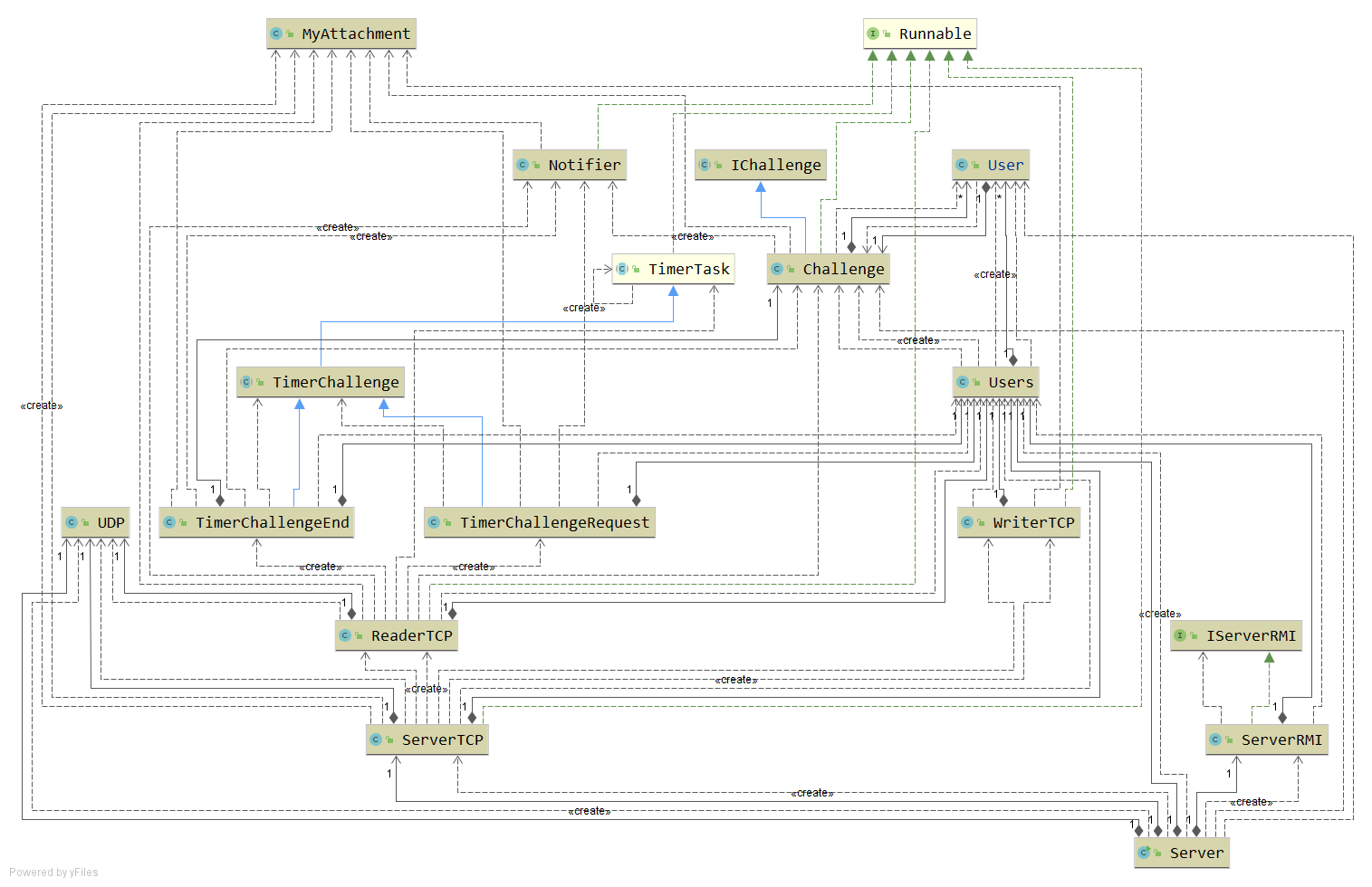
Il sistema utilizza le seguenti strutture dati:

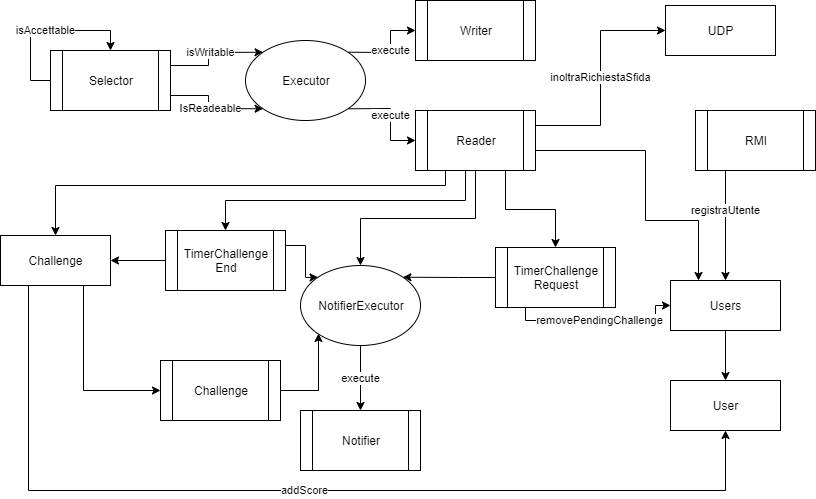
* **User**  
  Rappresenta un utente all’interno del sistema.
* **Users**  
  Rappresenta la raccolta degli utenti registrati nel sistema.
* **Challenge**  
  Rappresenta una sfida che avviene tra due utenti.
  + 1. **Threads**

Il sistema è formato dai seguenti threads:

* **Server**  
  Thread principale del server, inizializza le strutture dati per gli utenti e i servizi RMI, TCP, UDP.
* **ServerTCP**  
  Thread di gestione per i comandi TCP.  
  Viene utilizzato un selector reso multithread tramite l'utilizzo di worker di lettura (ReaderTCP) e scrittura (WriterTCP) inseriti in un threadpool.  
  Questo per velocizzare le operazioni ed evitare che il server si blocchi mentre sta servendo un client.  
  La funzione di accettazione delle connessioni invece viene fatta direttamente dato che non richiede molte risorse.  
  Quando una key del selector viene affidata ad un worker il suo set di interestOps viene settato a 0 in modo da non essere riselezionata.
* **ReaderTCP**  
  Si occupano della lettura e gestione dei messaggi ricevuti dal server e dell’assegnazione della risposta.  
  Quando ha terminato le operazioni setta la chiave in scrittura e risveglia il selector.
* **WriterTCP**  
  Si occupano dell’invio della risposta da parte del server.  
  Quando ha terminato le operazioni setta la chiave in lettura e risveglia il selector.
* **Notifier**  
  Questi thread, contenuti in un threadpool, si occupano della notifica di risposte che il server vuole notificare ai client.  
  Il funzionamento prevede che il thread si arresti fino a che la key non abbia come interestOps()=OP\_Read cioè che non la si sta utilizzando.  
  Quando viene sbloccato, il thread imposta la key in scrittura e assegna la risposta.
* **Challenge**  
  Il thread Challenge viene creato per ogni sfida istanziata e si occupa di effettuare le richieste API per la traduzione delle parole selezionate in modo da non bloccare il server.
  + 1. **Timers**

Per la gestione dei timeout vengono utilizzati i seguenti timer:

* TimerChallengeEnd  
  Si occupa di notificare la fine della sfida alla scadenza del timer e aggiornare le classifiche.
* TimerChallengeRequest  
  Si occupa di notificare la scadenza della richiesta di sfida.
  + 1. **Diagramma delle Classi**
    2. **Schema di Funzionamento**



* 1. **ARCHITETTURA CLIENT**

Il client presenta un’interfaccia grafica per l’interazione da parte dell’utente.  
Le informazioni delle porte del server a cui deve connettersi sono contenute in Settings.java  
Una volta avviato, il client, avvia il servizio RMI e i thread per il TCP e UDP ed apre la schermata di Login/Registrazione.  
Una volta che l’utente si è autenticato viene aperta la finestra principale del client.  
I thread comunicano con l’interfaccia grafica attraverso un sistema produttore-consumatore su strutture condivise, in questo modo si separa la gestione dei messaggi dalla GUI.

* + 1. **Strutture Dati**

Le strutture dati principali del client sono quelle relative alla comunicazione con la GUI:

* pendingFriendsList  
  Contiene la lista delle richieste di amicizia ricevute in sospeso
* friendsList  
  Contiene la lista di amici dell’utente
* classificaList  
  Contiene la classifica
* richiesteSfida  
  Contiene le richieste di sfida ricevute in sospeso
* sfida  
  Utilizzata per le informazioni sulla sfida accettata
  + 1. **Threads**

Per notificare alla GUI l’aggiornamento sulle liste vengono utilizzati dei thread in ascolto su di esse che vengono risvegliate dai thread che gestiscono i comandi quando vengono aggiunge delle informazioni ad esse e quindi c’è bisogno di aggiornare la GUI.  
Inoltre i thread principali di cui fa uso il server sono:

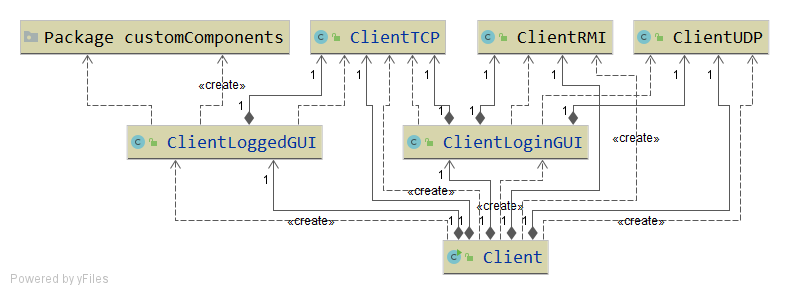
* ClientTcp  
  Questo thread si occupa della gestione di tutti i messaggi TCP ricevuti dal server.  
  Il thread è sempre bloccato su una read e alla ricezione, un messaggio di risposta, può essere di due tipi:
  + Risposta a una richiesta: queste risposte sono tutte precedute da un OK/NOK e sono messaggi di “ACK” ricevuti in seguito a una richiesta  
    Quando viene ricevuto un messaggio di risposta viene quindi inserito in una variabile che contiene sempre l’ultimo messaggio di risposta ricevuto accessibile tramite il metodo getResponse();
  + Risposte asincrone: questi messaggi sono preceduti direttamente dal tipo di risposta e possono essere ricevuti in ogni momento.  
    Alla ricezione di questo tipo di messaggi il thread si occupa di aggiornare le liste condivise e notificare cosi ai thread nella GUI che sono state modificate.

In questo modo è quindi possibile risolvere il problema del request-response che non permetterebbe la ricezione di messaggi senza prima aver inviato qualcosa.

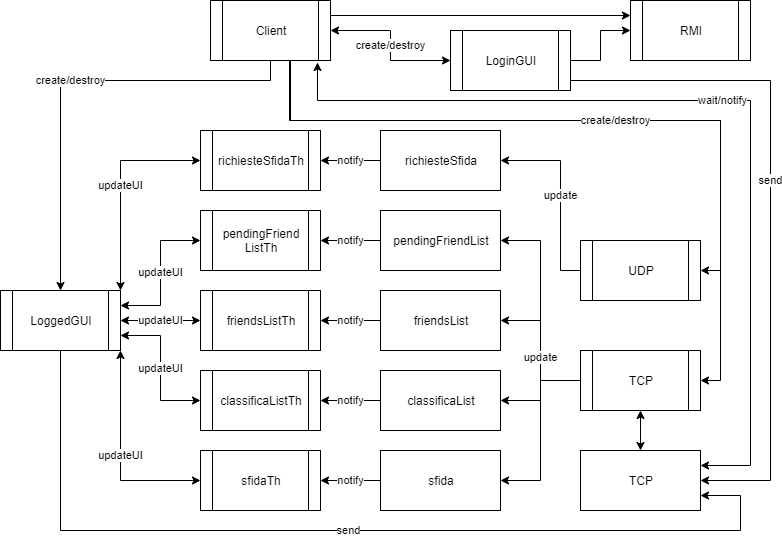
* ClientUdp  
  Questo thread, anch’esso bloccato in read si occupa della gestione delle richieste di sfida ricevute ed anche lui comunica con la GUI tramite la lista richiesteSfida.  
  + 1. **Interfaccia Grafica**

L’interfaccia grafica è stata progettata per essere semplice ed intuitiva.  
L’interfaccia è composta da due finestre:

* Finestra di login  
  Questa finestra permette all’utente di registrarsi o effettuare il login.
* Finestra utente loggato  
  Una volta effettuato il login si apre questa finestra che contiene tutte le informazioni relative all’utente come classifica, richieste di amicizia, amici, sfide ricevute e nella quale si svolgerà anche il gioco.
  + 1. **Schema delle Classi**



* + 1. **Schema di Funzionamento**

****

1. **COMUNICAZIONE**

Tutta la comunicazione client-server ad eccezione dell’RMI avviene tramite messaggi che seguono una precisa formattazione.

Per prevenire messaggi a nome di altri utenti viene creato un token per identificare la sessione di login.

Si potrebbe migliorare inoltre la sicurezza utilizzando un protocollo ibrido per crittografare interamente i messaggi.

* 1. **Comandi**

Per i comandi che il client invia al server la formattazione è la seguente:

* LOGIN NICK PW
* LOGOUT NICK TOKEN
* SFIDA NICK TOKEN FRIEND TYPE

Dove TYPE può essere RICHIEDI/ACCETTA/RIFIUTA

* AMICIZIA NICK TOKEN FRIEND TYPE

Dove TYPE può essere RICHIEDI/ACCETTA/RIFIUTA

* GET NICK TOKEN TYPE

Dove TYPE può essere AMICI/CLASSIFICA/PENDING

* PAROLA NICK TOKEN PAROLA

Dove parola indica la parola tradotta

* 1. **Risposte a Comandi**

Queste risposte vengono create e inviate dal server in risposta ai comandi ricevuti dal client come conferma.

* OK AMICIZIA TOKEN FRIEND TYPE

Dove:

* FRIEND è l’amico con cui si interagisce
* TYPE è il tipo di azione fatta RICHIESTA/ACCETTATA/RIFIUTATA
* OK SFIDA TOKEN FRIEND TYPE

Dove:

* FRIEND è l’amico con cui si interagisce
* TYPE è il tipo di azione per la sfida RICHIESTA/ACCETTATA/RIFIUTATA/INIZIATA/TERMINATA
* OK PAROLA TOKEN NEXTWORD/SCORE

Dove

* Se la sfida non è terminata allora NEXTWORD è la prossima parola da tradurre
* Se la sfida è terminata SCORE indica il punteggio totalizzato per la sfida
* OK TOKEN

Questa è la risposta per il LOGIN in cui il server comunica il token da utilizzare

* NOK ECCEZIONE TOKEN

Questa è la risposta in caso si verifichi un errore.

* 1. **Risposte Asincrone**

Queste sono le risposte che il server può inviare ad un client in qualsiasi momento quando vuole notificargli qualcosa.

* AMICI TOKEN JSON

Dove JSON contiene la lista degli amici

* CLASSIFICA TOKEN JSON

Dove JSON contiene la classifica

* PENDING TOKEN JSON

Dove JSON contiene la lista di richieste di amicizia

* AMICIZIA TOKEN FRIEND TYPE

Dove:

* FRIEND è l’amico al quale si riferisce
* TYPE è il tipo di notifica sull’amicizia RICHIESTA/ACCETTATA/RIFIUTATA
* SFIDA TOKEN FRIEND TYPE WORD/PUNTEGGIO

Dove:

* FRIEND è l’amico al quale si riferisce
* TYPE è il tipo di notifica sulla sfida RICHIESTA/ACCETTATA/RIFIUTATA/INIZIATA/TERMINATA/SCADUTA
* WORD/PUNTEGGIO è vuoto se TYPE=ACCETTATA, contiene il punteggio totalizzato per la sfida se TYPE=TERMINATA e contiene la prima parola se TYPE=INIZIATA