Relazione Reti

Fausto Frasca

February 11, 2020

1 Premesse

Come punto di riferimento per il progetto si è scelto un computer di ultima generazione non troppo potente, ma comunque con risorse hardware minime come 8gb di ram, un processore quad-core 8 thread e un ssd.
Su tale modello si è cercato di perseguire determinate scelte che garantissero un giusto compromesso tra uso efficiente delle risorse, velocità e complessità generale del sistema.

2 Architettura generale

In questo paragrafo verrà riportata una breve introduzione di quella che è l'architettura generale del sistema. Il tutto verrà approfondito nei capitoli successivi.

2.1 Client

Il client fornisce una GUI per l'interazione con l'utetne. Si è preferita tale scelta, anzichè la cli, per una più facile interazione per l'utente.
Una volta connesso al server, il client avrà un processo principale per gestire la comunicazione client-server su socket TCP e un thread a parte che sta in ascolto su una socket UDP per le richieste di sfida.

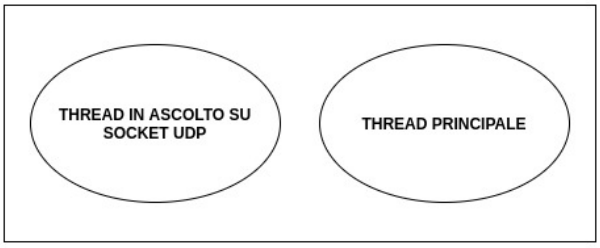


Figure 1: Schema client

2.2 Server

Il server è composto da un thread con selettore principale per la gestione della connessione TCP, un threadpool per svolgere operazioni "lunghe" delegate dal selettore e un oggetto RMI esportato per l'operazione di registrazione.
Sotto al server è presente un *user dispatcher*, con cui comunica richiedendo gli oggetti *User* in modo tale che sia il thread principale, sia i thread del threadpool operino sui medesimi oggetti.
Sotto allo user dispatcher è presente un *DBMS* che si occupa della gestione degli oggetti User in memoria, sia in lettura che in scrittura.
Come riportato nell'immagine sotto, il server comunica esclusivamente con lo user dispatcher e quest'ultimo comunica col DBMS.

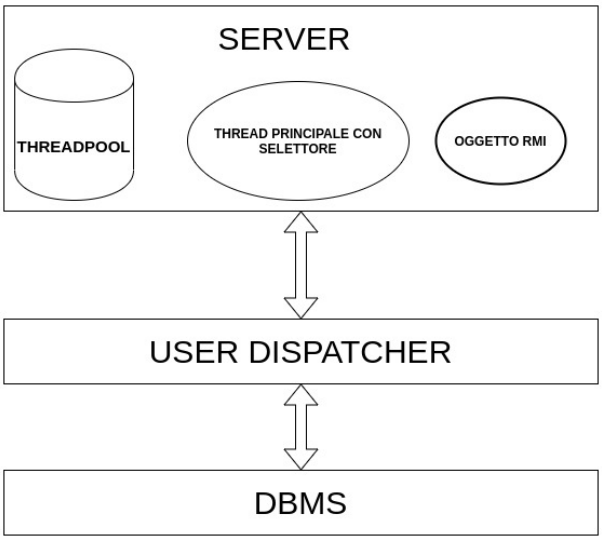


Figure 2: Schema server

3 Struttura generale delle directory del progetto

Il progetto è costituito da tre directory:

- Client
- Commons
- Server

3.1 Client

All'interno del path `Client/src/main/java` vi sono tutte le classi sfruttate dal client. Ognuna di queste classi permette di richiedere al server una delle operazioni indicate nel progetto (login, logout, aggiungi amico etc). È presente anche una directory `Costanti`. Al suo interno vi è un'interfaccia con le costanti generali usate dalle varie classi del client.

3.2 Commons

All'interno del path `Commons/src/main/java` sono presenti le interfacce comuni sia al client che al server.

3.3 Server

All'interno del path `Server/src/main/java` troviamo le classi sfruttate dal server. Alcune di queste sono raggruppate sottodirectory per una maggiore comprensione del lavoro svolto da ciascuna di loro.
Anche qui è presente una directory `Costanti` in cui è presente un'interfaccia con tutte le costanti usate dalle varie classi del server.

4 Descrizione classi

In questo capitolo verranno illustrate le singole classi presenti nelle tre directory descritte in precedenza.

4.1 Commons

4.1.1 RMIRegistrationInterface.java

Questa è l'interfaccia dell'oggetto RMI che poi sarà registrato nel registry.
Al suo interno è presente il metodo che permette a un utente di registrarsi e i vari codici ritornati del metodo stesso.
Sono inoltre definite le costanti per la porta e l'identificativo dell'oggetto remoto.

4.1.2 TCPConnection.java

All'interno di questa interfaccia sono definite le costanti per la connessione TCP tra client e server.

4.2 Client

Il metodo costruttore di tutte le classi contiene le istruzioni per stampare a video le varie componenti da inserire nel frame dell'interfaccia

4.2.1 MainClassClient.java

Questa classe ha il compito di settare il frame, che poi ospiterà i vari elementi della GUI, e richiamare un oggetto della classe startGUI.

4.2.2 StartGUI.java

Tale classe ha il compito di posizionare i vari componenti all'interno del frame creato in precedenza.
Le operazioni offerte all'utente sono: *login* e *registrazione*

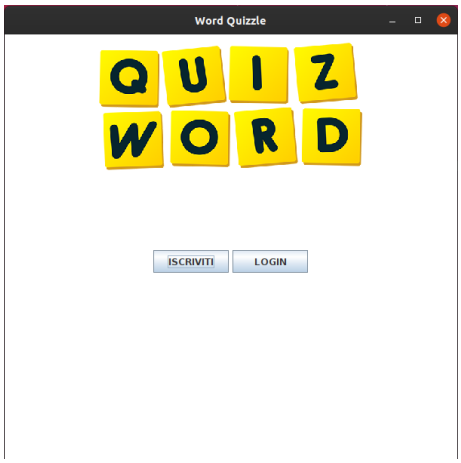


Figure 3: Schermata avvio client

4.2.3 Register.java

Questa classe sfrutta l'oggetto RMI remoto inserito nel registry, richiamando il metodo `registra_utente` che permette di effettuare la registrazione di un utente. Successivamente, tramite uno switch, si parsa il codice di ritorno del metodo richiamato.
Se il codice indica un errore, questo viene stampato nella parte inferiore del frame. Se invece la registrazione è andata a buon fine, viene creata in automantico un'istanza della classe *Login.java*.

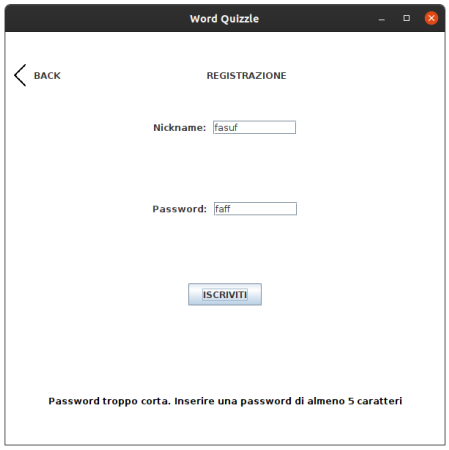
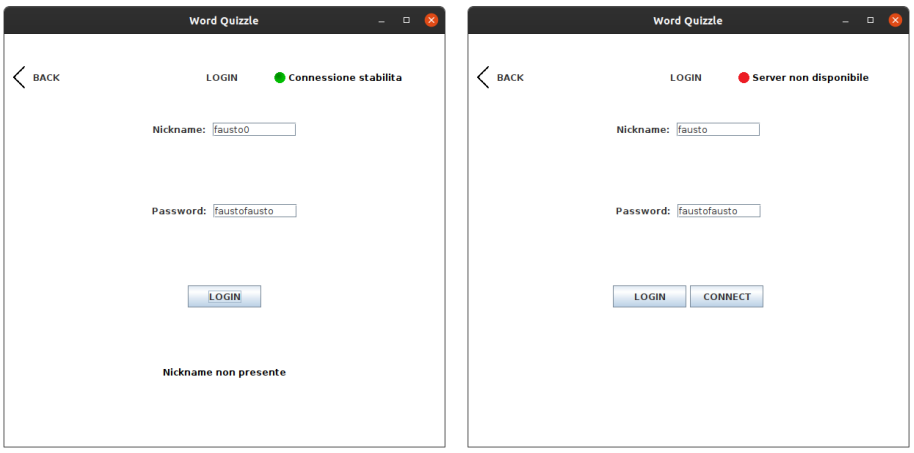


Figure 4: Schermata di registrazione con un eventuale errore riportato in basso

4.2.4 Login.java

Una volta entrati nella schermata di login, la classe instaura una connessione TCP col server, in base ai parametri definiti nell'interfaccia *TCPConnection*(mettere il riferimento).
L'avvenuto successo o meno della connessione viene riportato in alto, nel lato destro dell'interfaccia.
Qualora si fosse verificato un errore durante la connessione TCP, verrà mostrato un secondo pulsante denominato `CONNECT` per provare nuovamente a instaurarla.
Anche in questo caso, se ci fossero degli errori nell'autenticazione dell'utente, questi verrebbero riportati nella parte inferiore.
Se invece l'autenticazione del client è andata a buon fine si richiamano le classi *UDPListener.java* (e la si passa a un thread) e *HomePage.java*



(a) (b)

Figure 5: Schermata di login con un eventuale errore riportato in basso (a) Schermata di login con errore sulla connessione TCP (b)

4.2.5 HomePage.java

Superato il login, viene fatta comparire la home page. Qui l'utente può richiedere diverse operazioni:

- Aggiungere un amico
- Visualizzare la lista degli amici
- Visualizzare il proprio punteggio
- Sfidare un amico
- Richiedere il logout

Alcune risposte (come per esempio la conferma dell'avvenuta aggiunta di un amico) ed eventuali errori vengono riportati nella parte inferiore della schermata. Da qui in avanti, da qualsiasi punto della schermata, se la connessione TCP risultasse interrotta verrebbe mostrato un pop up di avviso e l'utente sarebbe reindirizzato alla schermata iniziale (4.2.2).

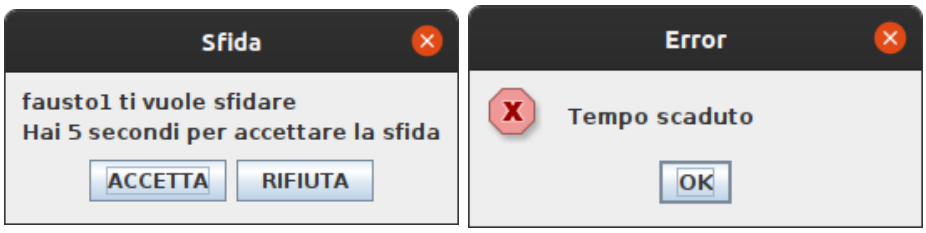
(a) (b)

Figure 6: Home page (a)
Pop up che indica l'interruzione della connessione TCP (b)

4.2.6 UDPListener.java

Questa è una classe di tipo singleton. Si è decisa tale caratteristica per far si che l'utente, per tutto il periodo che risulta online, si interfacci sempre con lo stesso oggetto della classe.

Qui viene gestita tutta la parte riguardante la socket UDP per le richieste di sfida tra utenti. Una volta avviata, la classe si occupa di creare la socket e di aprirla sulla stessa porta su cui è stata aperta la socket TCP.
Non appena riceve un datagramma, che rispetta il protocollo definito (inserire riferimento), setta il flag di sfida della classe `ChallengeFlag.java`(mettere riferimento) e mostra a video un pop up che permette all'utente di accettare o meno la sfida. All'interno del pop up viene mostato chi è lo sfidante e il tempo massimo che ha per accettare la sfida.
Se la sfida viene accettata viene creato un oggetto della classe *Challenge.java* e la sfida parte, altrimenti il flag di sfida viene resettato.
Se invece l'utente accetta oltre il tempo previsto, verrà mostrato un altro pop up che indica l'avvenuto timeout.



(a) (b)

Figure 7: Richiesta di sfida (a)
Pop up che indica l'avvenuto timeout (b)

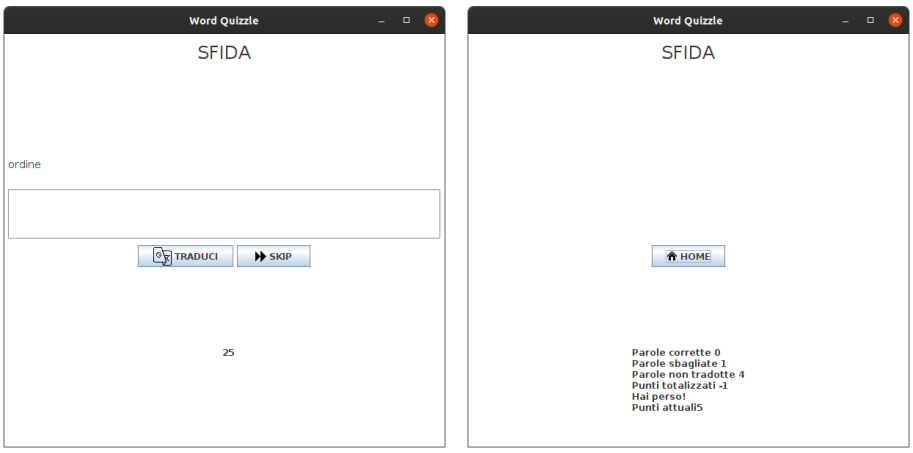
Il flag di sfida serve per evitare che l'utente riceva richieste di sfida mentre a sua volta è già impegnato in una sfida o ha già una richiesta di sfida in sospeso. Quando il flag è settato tutte le nuove richieste sono rifiutate in automatico.

4.2.7 ChallengeFlag.java

Classe singleton per far si che il thread principale e quello UDP operino sulla medesima istanza.
Quando si invia o si riceve una richiesta di sfida, tale flag viene settato a 1 e una volta terminata la sfida o rifiutata la richiesta viene posto a 0.
Il flag è un `AtomicInteger` che garantisce il cambio di valore in maniera atomica, facendo si che il valore sia sempre consistente per entrambi i thread.

4.2.8 Challenge.java

Questa classe si occupa della gestione vera e propria della sfida.
Legge le parole in italiano da tradurre e invia al server le parole tradotte dall'utente. Permette anche di poter saltare la traduzione di una parola tramite l'apposito pulsante `SKIP`.
Nella parte inferiore è anche riportato il tempo massimo a disposizione per l'utente per terminare la sfida.
Finita la sfida, sempre nella zona inferiore, viene riportato un riassunto di quelle che sono le parole tradotte in modo corretto, errato o non tradotte. Viene riportato anche un riassunto di quello che è il guadagnato e quello attuale dell'utente.



(a) (b)

Figure 8: Schermata durante la sfida (a)
Riassunto post sfida (b)

4.2.9 ShowFriends.java e ShowRank.java

Queste due classi hanno caratteristiche simili.
ShowFriends mosta la lista degli amici dell'utente, mentre la seconda mostra la classifica in base ai punti accumulati con le varie sfide.

4.3 Server

Le classi presenti nella directory `Task` sono classi le cui istanze sono eseguite da un thread del threadpool.

4.3.1 Database/DBMS.java

Questa classe, contenuta nella directory `Database`, si occupa di gestire gli utenti memorizzati.
È l'unica classe che legge e scrive sulla memoria.
Ogni utente è salvato all'interno cartella `Dati` e tutti i suoi dati sono scritti su un file json denominato secondo lo schema `nicknameUtente.json` (ad esempio fausto.json).
Si è optato per avere un file per ogni utente, anzichè uno contenente tutti gli utenti, per velocizzare tutte quelle operazioni semplici (come per esempio controllare se un utente è già registrato). Infatti, sfruttando la struttura ad albero del filesystem, la ricerca di un file relativa a un utente è effettuata in tempo logartmico.
Per serializzare e deserializzare l'oggetto User, si fa uso della libreria Gson.
Presenta due metodi thread-safe: `existUser` e `registerUser`
Questo perchè, in realtà, l'oggetto RMI per la registrazione bypassa lo user dispatcher, comunicando direttamente col DBMS.
Di conseguenza, per gestire la conccorrenza tra RMI e user dispatcher tali metodi sono stati resi `syncronized`.
Perciò una vera rappresentazione dell'architettura server è riportata nell'immagine in basso

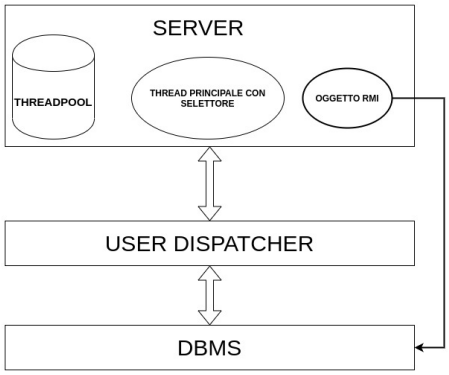


Figure 9: Architettura server

4.3.2 User/User.java

Questa è la classe che rappresenta l'utente. Al suo interno sono presenti dati come nickname, password, lista di amici e punteggio.

Poichè gli oggetti di tale classe saranno risorse condivise da più thread, il punteggio è un `AtomicInteger`. Inoltre le istanze della classe sono "rientranti", ovvero che contengono un contatore denominato `use` (`AtomicInteger` anche lui) che viene incrementato quando l'istanza viene usata da un thread e decrementato quando non ne ha più bisogno.

4.3.3 User/UserDispatcher.java

Questa classe, singleton, gestisce tutte quelle che sono le istanze degli utenti richiesti dal server a livello superiore, richiedendole al DBMS a sua volta qualora non risultassero presenti nella propria struttura dati struttura dati.
La struttura dati usata è una `ConcurrentHashMap`.
La scelta è ricaduta su tale struttura per la velocità e l'efficienza offerta.
All'interno dello user dispatcher è presente anche un thread daemon cleaner, che viene risvegliato a intervalli di tempo regolari per ripulire la map da tutti quegli utenti il cui contatore `use` risulta azzeraro.
Se si guarda bene il codice, si può notare che, nonostante la struttura dati apparentemente sia thread-safe, in realtà non lo è.
Infatti, com'è riportato nella documentazione ufficiale, tale struttura dati ammette che alcune operazioni, come per esempio la `get` e la `delete`, possano avvenire in contemporanea. Di conseguenza per garantire la consistenza della map, si è dovuto optare per rendere i metodi `syncronized`.

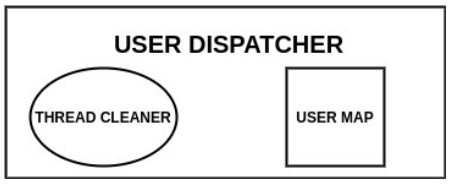


Figure 10: Architettura user dispatcher

4.3.4 Server/ConKey.java

Gli oggetti di questa classe vengono allegati alla key che viene generata quando un client si connette al server.
Contiene diverse informazioni e flag utili per la gestione delle request client-server e response server-client.
È presente anche un flag che indica se l'utente è già impegnato in una sfida, in modo da porre un ulteriore filtro alle richieste quando un utente ne sta già svolgendo una.

4.3.5 Server/ConChallenge.java

Gli oggetti di questa classe invece vengono allegati alle key generate qunado i due sfidanti vengono registrati nel selettore di sfida.

Anche in questo caso sono presenti diversi flag utili per la gestione della sfida.

4.3.6 Server/DictionaryDispatcher.java

Poichè l'istanza di questa classe è sfruttata da più thread, è stata resa singleton. Il suo scopo è quello di scegliere un determinato numero di parole casuali dal dizionario, richiedere la traduzione in inglese e ritornare la lista contenente la coppia {`parola italiana, traduzione inglese`}.
Per quanto riguarda la traduzione, si è notato che alcune traduzioni restituite non erano esatte. Perciò è stato aggiunto un ulteriore servizio di traduzione a supporto del primo. Il servizio è Yandex che garantisce una traduzione migliore. Anche in questo caso la traduzione è effettuata tramite il metodo `Get` e la risposta è restituita in json.
Lo schema scelto è il seguente:

1. Richiedo la traduzione della parola a Yandex
2. Richiedo la traduzione della parola a MyMemory
3. Confronto la lista delle possibili traduzioni restituita da MyMemory con la traduzione fornita da Yandex.
   Se trovo due traduzioni che combaciano allora ritorno la taduzione di MyMemory, altrimenti ritorno quella di Yandex.

Vengono gestiti diversi errori, tra cui il caso in cui uno dei servizi di traduzione risultasse irraggiungibile. **inserire foto illustrativa**

4.3.7 Task/AddFriend.java

Il compito di questa classe è quella di aggiungere l'username di due utenti nelle rispettive liste di amici.
Prima di tutto vengono effettuati diversi controlli, come il caso in cui un utente provi ad aggiungersi tra gli amici.
Superati i controlli, si acquista la lock dell'oggetto che rappresenta la lista di amici e si effettua l'operazione.

4.3.8 Task/ShowFriends.java

Qui viene acquistata la lock sull'oggetto che rappresenta la lista di amici di un utente.
Successivamente viene creata una stringa json contente la relativa lista in modo da essere inviata al client che ne ha fatto richiesta.

4.3.9 Task/ShowRank.java

Anche qui viene acquisita la lock sull'oggetto che rappresenta la lista di amici di un utente. In questo caso però vengono fatte delle richieste allo user dispatcher per ottenere gli oggetti associati ai nickname degli amici, in modo tale da creare

una lista contenenti le coppie {`nickname,score`}.
Tale lista poi viene ordinata in maniera decrescente di score e convertita in stringa, in modo da essere spedita in risposta al client che ne ha fatto richiesta.

4.3.10 Task/ChallengeRequest.java

Tale classe è quella che si fa carico di tutta la parte pre-sfida.
Crea una socket UDP in modo da inviare la richiesta di sfida allo sfidante, secondo il protocollo stabilito (inserire riferimento).
Successivamente registra la key dello sfidante in un selettore temporaneo ed attende per un periodo di tempo predefinito.
Se la risposta non arriva entro l'arco di tale periodo allora la richiesta di sfida si considera rifiutata.
In caso contrario, si legge e si parsa la risposta dello sfidato. Se la risposta è negativa, viene riportata allo sfidante e il tutto termina.
Se invece la risposta è positiva viene creata un'istanza della classe *Challenge.java*.
Vengono tenuti in considerazioni diversi casi, come per esempio il caso in cui lo sfidato si disconnette senza dare una risposta alla richiesta di sfida, oppure il caso in cui lo sfidante, una volta inviata la richiesta, si disconnette senza attendere la risposta dello sfidato.

4.3.11 Task/Challenge.java

Prima di avviare effettivamente la sfida, vengono eseguite determiante operazioni.
Innanzitutto si richiede la lista di parole da usare per la sfida alla classe *DictionaryDispatcher* (4.3.6). Se la lista viene restituita senza che si verifichino errori si prosegue registrando le key dei due sfidanti su un selettore creato appositamente per la sfida.
Alle nuove key generate viene allegata un'istanza della classe *ConChallenge* (4.3.5).
Terminata tutta questa fase di setup la sfida comincia.
Agli sfidanti viene riportato il timer che indica il massimo tempo a disposizione per completare la sfida e poi, in successione, avviene uno scambio richieste-risposte tra client e server in cui il client invia la parola tradotta e il server invia la nuova parola da tradurre.
Una volta che entrambi gli sfidanti hanno concluso le parole da tradurre, o è scattato il timeout, viene eseguita la parte finale della sfida.
Consiste nel contare le parole tradotte correttamente, quelle sbagliate e quelle non date e assegnare il relativo punteggio (2 punti per ogni parola corretta, -1 per quelle errate e 0 per quelle non tradotte). Il tutto viene inserito in una stringa e spedito al client come risposta finale.
Finita la sfida le due key vengono nuovamente registrate sul selettore principale del server.