Word Quizzle

Serafino Gabriele - Mat. 564411

8 gennaio 2020

Indice

1 Descrizione del problema

Il progetto consinste nell'implementazione di un sistema di sfide di traduzione italiano-inglese tra utenti registrati al servizio. Gli utenti possono sfidare i propri amici ad una gara il cui scopo è quello di tradurre correttamente, dall'italiano all'inglese, il maggior numero di parole proposte.

Il sistema consente inoltre la gestione di una rete sociale tra gli utenti iscritti, permette cioè di creare amicizie tra utenti e visualizzare una classifica in base ai punteggi ottenuti nelle sfide.

2 Architettura del sistema

L'applicazione è progettata come un sistema *client-server*, le due parti comunicano tramite socket TCP/UDP su Internet.

Il server è implementato a riga di comando e non necessita di nessuna interazione con l'utente una volta avviato.

Il client presenta invece una semplice interfaccia grafica, che permette all'utente di usufruire di tutte le funzionalità offerte in modo veloce ed intuitivo.

2.1 Server

Il server utilizza un *Selector* insieme ad un *Threadpool* per gestire le richieste degli utenti: ogni richiesta viene letta dal main thread ed incapsulata in un *Task* eseguito poi sul threadpool; una volta elaborata la risposta, il main thread utilizza il Selector per inviarla al Client.

In fase di inizializzazione il server schedula altri due thread che rimangono attivi durante tutto il suo funzionamento:

- **un thread di backup**, si occupa di caricare dalla memoria di massa i dati degli utenti all'avvio del server, e ad intervalli regolari salva, se presenti, le modifiche sul file.
- **un thread che gestisce una socket udp**, che rimane in attesa di leggere eventuali messaggi udp dai client in maniera asincrona (risposte a richieste di sfida).

Il server espone inoltre un oggetto remoto, che permette al client di effettuare l'iscrizione di un nuovo utente. Tutte le altre operazioni che effettua il client vengono trattate come messaggi di *richiesta-risposta* su una connessione *TCP persistente*, instaurata in fase di login, la quale segue un semplice protocollo di comunicazione descritto in seguito.

Oltre a questi due tipi di connessione abbiamo una comunicazione *UDP* utilizzata dal server solo per inviare eventuali richieste di sfida in modo asincrono, e per ricevere la risposta dall'utente.

2.2 Client

Il client è formato da un main thread, che si occupa dell'inizializzazione e della schedulazione degli altri thread:

- **un thread per ogni form della GUI**.
- **un thread che gestisce una socket UDP**, per la ricezione di richieste di sfida asincrone dal server.

2.3 Strutture dati e concorrenza

Il server mantiene in memoria di massa due file:

- il file **dictionary.dat** su cui sono memorizzate una lista di parole italiane, una per ogni riga, che costituiscono le possibili parole da tradurre durante le sfide.
- il file **users.json** che contiene tutte le informazioni su utenti, punteggi e relazioni di amicizia in formato json.

All'avvio del server queste informazioni vengono caricate in delle apposite strutture.

Il dizionario viene caricato su un semplice *array* di stringhe, una volta caricato infatti questo array viene utilizzato solo in lettura, quindi non presenta problemi di concorrenza. Inoltre, poichè il server sceglie una serie di parole casuali per ogni sfida, non viene effettuata nessuna ricerca vera e propria sull'array, ma solo un accesso diretto.

I dati sugli utenti vengono invece caricati su una *ConcurrentHashMap*, che permette la gestione della concorrenza e tempi di ricerca efficienti. L'hash map utilizza come chiave l'username dell'utente (univoco) e come dato un puntatore ad un oggetto di tipo *Utente*, contenente tutte le informazioni che lo riguardano.

Anche i dati relativi ad una sfida vengono mantenuti su una *ConcurrentHashMap*: un oggetto di tipo *Match* viene inserito nella hashmap alla creazione di una nuova sfida e viene eliminato non appena quest'ultima termina.

3 Struttura di una sfida

3.1 Avvio di una sfida

Per iniziare una sfida tra due utenti A e B è necessario che uno dei due invii una richiesta di sfida all'altro. L'utente A esegue il login al servizio e ha la possibilità di inviare una richiesta di sfida ad uno dei suoi amici(utente B). Il server esegue quindi un semplice controllo per verificare che l'utente B risulti online, in caso affermativo inoltra la richiesta a B, altrimenti risponde all'utente A con un apposito messaggio di errore.

Se online, l'utente B riceve la richiesta e può decidere di accettare o meno la sfida (se non risponde entro un certo intervallo di tempo viene automaticamente declinata la richiesta).

In caso di accettazione della sfida i due utenti ricevono le specifiche della partita e la prima parola da tradurre. Ogni utente invia quindi la traduzione e riceve la successiva parola da tradurre; al termine delle parole o dopo un certo intervallo di tempo (durata della partita), ogni utente visualizza un messaggio con il risultato della partita ed il punteggio ottenuto.

3.2 Abbandono di una sfida

Durante una sfida un utente può abbandonare quest'ultima in modo volontario (surrend) o involontario (crash dell'applicazione), in entrambi i casi il punteggio della sfida per quel determinato utente viene azzerato e la partita viene vinta dall'altro utente. L'utente che non abbandona continua comunque la sua partita fino alla fine.

Nel caso in cui entrambi gli utenti abbandonino la sfida, il punteggio degli utenti non viene modificato, perciò è come se la sfida non fosse mai avvenuta.

3.3 Punteggio partita e punteggio giocatore

In ogni sfida entrambi i giocatori hanno un punteggio, chiamato *punteggio partita*, che equivale alla somma dei punti assegnati per ogni parola tradotta correttamente o incorrettamente. Il giocatore con punteggio partita maggiore sarà dichiarato vincitore e gli saranno assegnati dei *punti bonus*.

Al termine della partita, il punteggio partita e gli eventuali punti bonus saranno sommati al *punteggio giocatore*, utilizzato per la classifica.

In caso di parità entrambi i giocatori vengono dichiarati vincitori e ricevono i punti bonus.

4 Servizio di traduzione

Il server WQ si appoggia ad un servizio esterno per ottenere la traduzione delle parole di una sfida. L'host *mymemory.translated.net* offre una **API** che permette di effettuare una richiesta di traduzione. Il servizio risponde alla richiesta **GET** con un file **JSON**, contenente una lista di possibili traduzioni della parola richiesta.

All'inizio di ogni sfida vengono effettuate diverse richieste al servizio esterno per ottenere la traduzione di ogni parola scelta per quella sfida (le traduzioni vengono mantenute in memoria per la sola durata della sfida).

Quando il server WQ riceve una traduzione dall'utente controlla che quella traduzione sia presente nella lista restituita dal servizio per decidere se considerarla corretta o meno.

La richiesta inviata dal server WQ è una semplice **GET HTTP** che contiene come parametri la parola da tradurre e la coppia di lingue per effettuare la traduzione.

Durante la realizzazione del progetto si è notato che il servizio chiude la connessione HTTP anche se si specifica l'opzione di keep-alive, perciò ad ogni parola da tradurre è necessario effettuare l'apertura della connessione, aumentando notevolmente il tempo necessario ad inizializzare la sfida.

5 File di configurazione

La prima operazione effettuata dal server al suo avvio è quella di effettuare il parsing del file di configurazione *server_ config.properties*.

Questo file contiene, nel formato *.properties*, tutti i parametri necessari al funzionamento del server insieme ad una breve descrizione per ognuno di essi. E' possibile quindi modificare i seguenti valori nel file di configurazione:

- **registry_port**: la porta su cui viene esposto l'oggetto remoto per la registrazione di un utente.
- **main_port**: la porta principale sulla quale il server è in ascolto per le connessioni TCP.
- **side_port**: la porta utilizzata per la socket UDP.
- **translation_server_address**: l'indirizzo mnemonico del servizio di traduzione esterno.
- **dictionary_path**: il pathname del file *dictionary.dat* contenente la lista di parole da tradurre.
- **thread_pool_size**: la dimensione del thread pool che gestisce le richieste degli utenti.
- **backup_time**: la durata dell'intervallo di tempo con cui il thread di backup salva i dati degli utenti su memoria di massa.
- **match_request_timeout**: l'intervallo di tempo a disposizione dell'utente che riceve una richiesta di sfida per accettare o rifiutare quest'ultima.
- **match_word_number**: il numero di parole da tradurre in una sfida tra due utenti.
- **match_duration**: il tempo a disposizione degli utenti per tradurre tutte le parole di una sfida.
- **correct_word_score**: il punteggio assegnato per ogni parola tradotta correttamente.
- **incorrect_word_score**: il punteggio assegnato per ogni parola tradotta in modo errato.
- **match_winner_score**: il punteggio bonus assegnato al vincitore della sfida.

Analogamente il client carica all'avvio il proprio file di configurazione, chiamato *client_ config.properties*, contenente i seguenti parametri:

- **server_adddress**: l'indirizzo IP del server in notazione decimale puntata.
- **registry_port**: la porta su cui il server espone il servizio di registrazione.
- **main_port**: la porta su cui il server è in ascolto di connessioni TCP.

6 Protocollo di comunicazione

La comunizione tra client e server avviene su socket TCP/UDP secondo il paradigma domanda-risposta, utilizzando il seguente protocollo.

- **Login**: effettua il login dell'utente *username* con la relativa *password*. Invia inoltre la porta sul quale il client è in ascolto con la socket UDP per le richieste di sfida.

  Cient: login username password side_port

  Server: login_r result [reason]

  $$result \in \{ok, ko\}$$

  $$reason \in \{wrong\_username, wrong\_password, already\_logged, bad\_request\}$$

- **Logout**: effettua il logout dell'utente se è loggato e non è impegnato in una sfida.

  Client: logout

  Server: logout_r result [reason]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted\}$$

- **Add friend**: crea, se non esiste, una relazione di amicizia tra l'utente che invia la richiesta e l'utente *friend_username*, se quest'ultimo esiste.

  Client: add_friend friend_username

  Server: add_friend_r result [reason]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, wrong\_username, already\_friend, not\_permitted, bad\_request\}$$

- **Score**: restituisce il punteggio dell'utente.

  Client: score

  Server: score_r result [reason] [user_score]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted\}$$

- **Friends list**: restituisce, in formato *JSON*, la lista degli amici dell'utente.

  Client: friends_list

  Server: friends_list_r result [reason]\n[friends_list_json]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted\}$$

- **Leaderboard**: restituisce, in formato *JSON*, la classifica contenente l'utente e tutti i suoi amici ordinati per punteggio.

  Client: leaderboard

  Server: leaderboard_r result [reason]\n[leaderboard_json]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted\}$$

- **Match**: invia la richiesta di sfida all'utente *friend_username*.

  Client: match friend_username

  Server: match_r result [reason] [response]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, wrong\_username, not\_friend, user\_offline, user\_busy, not\_permitted\}$$

  $$response \in \{accepted, refused\}$$

- **Notify match request (UDP)**: il server inoltra la richiesta di sfida all'utente su socket UDP.

  Server: match_request friend_username match_id timeout

  Client: match_request_r match_id response

  $$response \in \{accepted, refused\}$$

- **Match info**: restituisce le informazioni riguardanti la sfida in corso.

  Client: match_info

  Server: match_info_r result [reason] [word_count] [session_time] [word]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted\}$$

- **Send translation**: invia la traduzione di una parola e restituisce la successiva parola da tradurre.

  Client: translation translated_word

  Server: translation_r result [reason] [next_word]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted, bad\_request\}$$

- **Match results**: restituisce il risultato della partita.

  Client: match_results

  Server: match_results_r result [reason] [correct_count] [wrong_count] [untranslated_count] [score] [friend_score] [bonus_points]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted\}$$

- **Surrend**: notifica al server la resa volontaria dell'utente dalla sfida in corso.

  Client: surrend

  Server: surrend_r result [reason]

  $$result \in \{ok, ko\}$$

  $$reason \in \{not\_logged, not\_permitted\}$$

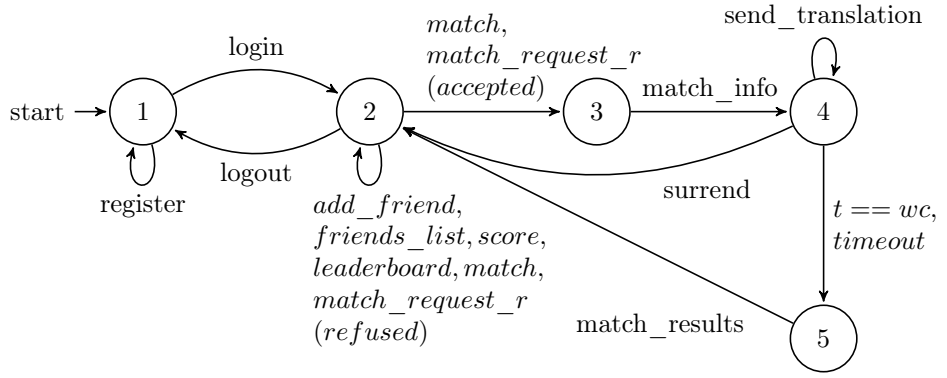
- **Ping**: messaggio inviato dal server per verificare che il client sia ancora online. Il client ignora il messaggio.

  Server: ping

  Client: (ignores the message)

7 Stato di un utente

Il server mantiene per ogni utente una variabile con il suo stato. Di seguito è mostrato l'ASF per lo stato di un utente.



No.	State
1	Started
2	Logged
3	Match Started
4	Match Playing
5	Match Finished

8 Classi

8.1 Classi server

- **Server.java**: contiene la funzione main del server. Si occupa di inizializzare il server, gestisce il selector e schedula i task sul threadpool. Si occupa della comunicazione TCP con i client (accept, read, write).
- **RequestHandler.java**: rappresenta il task che gestisce una singola richiesta di un utente, riceve la stringa contenente la richiesta ed elabora la risposta, salvandola nella struttura utente per poi essere inviata tramite socket dal main.
- **RegistrationService.java**, **RegistrationServiceImplementation.java**: interfaccia e relativa implementazione del servizio *RMI* per la registrazione di un utente.
- **BackupService.java**: servizio eseguito su un thread che si occupa del caricamento e del salvataggio dei dati utente su memoria di massa.
- **User.java**: rappresenta un utente registrato al servizio. Oltre ai dati persistenti(*username*, *punteggio*, *password*, *lista amici*) contiene informazioni temporanee come lo stato (*logged*, *match_started...*) ed un eventuale puntatore al match in corso.
- **Match.java**: rappresenta un match in corso, contiene un puntatore ai due utenti coinvolti, lo stato del match, la lista di parole da tradurre con relative traduzioni corrette, e il progresso di ogni giocatore.
- **UserState.java**: *enumerator* con i possibili stati utente.
- **MatchState.java**: *enumerator* con i possibili stati del match.
- **MatchNotificationService.java**: servizio eseguito su un thread che si occupa di leggere su socket *UDP* le risposte alle richieste di match.
- **TranslationService.java**: espone un metodo statico per la traduzione di una lista di parole, utilizzando il servizio esterno.

8.2 Classi client

- **Client.java**: Contiene la funzione main del client. Si occupa di inizializzare il client e di avviare il form di login.
- **LoginForm.java**, **LoginForm.form**: implementazione e interfaccia grafica del form di login, permette di registrare o effettuare il login di un utente.
- **ProfileForm.java**, **ProfileForm.form**: implementazione ed interfaccia grafica del form contenente le informazioni sul profilo utente e la classifica degli utenti amici. Permette di effettuare il logout, aggiungere un amico e sfidare un amico.
- **MatchForm.java**, **MatchForm.form**: implementazione ed interfaccia grafica del form che permette di giocare una sfida. Mostra un timer di sfida, la parola da tradurre e permette di tradurre una parola o abbandonare volontariamente la sfida.

8.3 Classi comuni

- **ErrorCode.java**: *enumerator* con i possibili errori delle richieste utente.
- **FileInteraction.java**: espone metodi statici per la lettura e scrittura su file.
- **SocketInteraction.java**: espone metodi statici per la lettura e scrittura su socket *TCP* e *UDP*.
- **Leaderboard.java**: rappresenta una classifica, contiene una lista di *LeaderboardUser*, ordinata per punteggio. Viene creata dal server ed inviata al client in formato *JSON* in risposta al comando *"leaderboard"*.
- **LeaderboardUser.java**: rappresenta un utente (*username* e *punteggio*) in una leaderboard.

9 Compilazione e compatibilità

9.1 Compatibilità

Il sistema è stato testato su JAVA 11.0.4, sui seguenti sistemi operativi:

- Windows 10
- Kubuntu 19.04

9.2 Librerie esterne

Per la realizzazione del progetto sono state utilizzate le seguenti librerie:

- **JSON Simple** (*json-simple-1.1.1.jar*): per il parsing e la gestione dei dati in formato *json*.
- **Forms RT** (*forms_ rt.jar*): per la compilazione dell'interfaccia grafica (form) del client.

9.3 Compilazione del server

La compilazione del server si può effettuare con il comando:

javac -cp ".:./json-simple-1.1.1.jar" *.java (Linux)
javac -cp ".;.\json-simple-1.1.1.jar" *.java (Windows)

E' necessario compilare le classi presenti nelle sezioni 10.1 e 10.3, oltre ad avere nella directory corrente anche il file .jar della librearia JSON Simple.

Per eseguire il server si usa:

java -cp ".:./json-simple-1.1.1.jar" Server <configfile> (Linux)
java -cp ".;.\json-simple-1.1.1.jar" Server <configfile> (Windows)

Una volta avviato, il server mostrerà i log d'esecuzione, i quali comprendono anche la fase di inizializzazione. Il server è operativo quando viene visualizzato il log "[INIT] Server ready".

Per terminare l'esecuzione è necessario inviare il segnale di interruzione (CTRL+C).

9.4 Compilazione del client

La compilazione del client si può effettuare con il comando:

javac -cp ".:./json-simple-1.1.1.jar:./forms_rt.jar" *.java (Linux)
javac -cp ".;.\json-simple-1.1.1.jar;.\forms_rt.jar" *.java (Windows)

E' necessario compilare le classi presenti nelle sezioni 10.2 e 10.3, oltre ad avere nella directory corrente anche il file .jar della librearia JSON Simple e Forms RT.

Per eseguire il client si usa:

java -cp ".:./json-simple-1.1.1.jar:./forms_rt.jar" Client <configfile> (Linux)
java -cp ".;.\json-simple-1.1.1.jar;.\forms_rt.jar" Client <configfile> (Windows)

Una volta avviato il client mostrerà il form di login.

10 Manuale utente

10.1 Esecuzione del server

Durante la sua esecuzione, il server mostrerà a riga di comando log autoesplicativi delle operazioni effettuate dagli utenti.

Alcuni log possono essere preceduti dal tag "ERROR", se si verificano problemi in fase di inizializzazione, come l'impossibilità di caricare il file di configurazione o il dizionario.

Un'ultimo tipo di log, preceduto dal tag "WARNING" può riferire il fatto che per un determinato termine del dizionario non è stata trovata nessuna traduzione sul servizio esterno. In questo caso qualsiasi traduzione inviata dall'utente sarà considerata errata, è consigliabile perciò rimuovere tali termini dal dizionario.

10.2 Esecuzione del client

All'avvio del programma client verrà mostrata una schermata di login (Figura 1).

Qui è possibile decidere se registrarsi al servizio o effettuare il login. Basterà inserire username e password negli appositi campi e premere sull'operazione desiderata.

Nel caso di registrazione verrà visualizzato un messaggio di successo o un eventuale errore.

Premendo il pulsante Login, verrà mostrata la finestra principale con le informazioni profilo e la classifica, o eventualmente un messaggio d'errore.

Figura 1: Schermata di login

Una volta effettuato correttamente l'accesso, la schermata del profilo (Figura 2) visualizza l'username utente ed il suo punteggio sul lato sinistro, e la classifica dei propri amici sul lato destro. Qui è possibile:

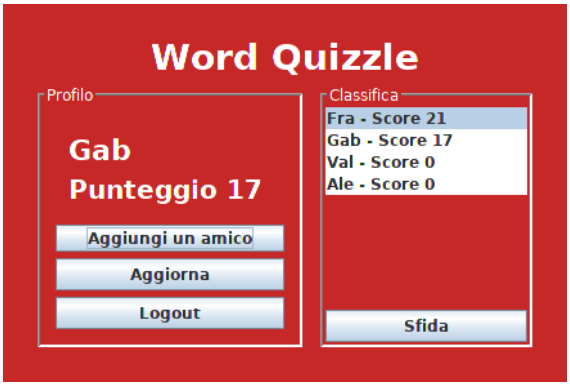


Figura 2: Schermata del profilo

- Aggiungere un amico ("**Aggiungi un amico**"), inserendo l'username dell'utente amico nel campo che viene visualizzato.
- Aggiornare i dati presenti nella schermata (punteggio e classifica) ("**Aggiorna**").

- Effetturare il logout.
- Sfidare un amico, selezionando l'amico da sfidare dalla classifica ("**Sfida**").

Quando un utente A invia una richiesta di sfida all'utente B, se quest'ultimo è online, entrambi visualizzeranno un messaggio. L'utente A visualizzerà un messaggio di attesa (Figura 3(a)), mentre l'utente B visualizzerà un messaggio dove potrà decidere se accettare o rifiutare la sfida (Figura 3(b)).

(a) Attesa di risposta (b) Ricezione richiesta

Figura 3: Richiesta di una sfida

Se l'utente B accetta la sfida entrambi visualizzeranno una schermata in cui potranno conoscere il tempo rimanente per la sfida (in alto) e la parola da tradurre (Figura 4).

Qui l'utente può inserire la traduzione nell'apposito campo ed inviarla (ricevendo così la prossima parola da tradurre) o arrendersi volontariamente.

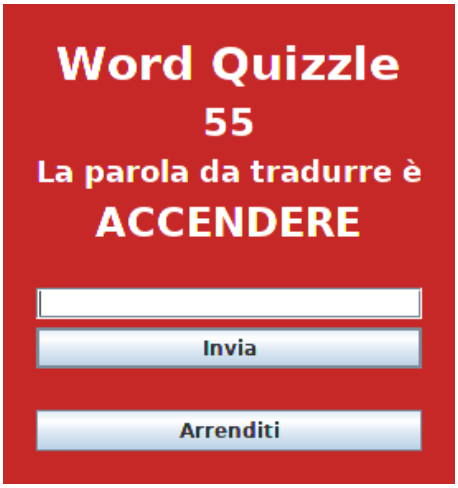


Figura 4: Schermata di una sfida

Una volta terminata la sfida l'utente visualizzerà il risultato (Figura 5), con il numero di parole corrette, errate e non tradotte ed il punteggio di entrambi i giocatori, e potrà tornare alla schermata del profilo.

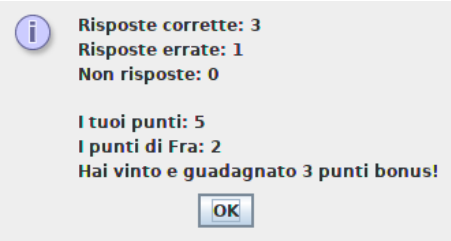


Figura 5: Risultato della sfida