UNIVERSITA' DI PISA - A.A 2020/2021

RELAZIONE PROGETTO DI RETI E LAB.

realizzato da

SIMONE IOZIA N.549108

INDICE

1 Introduzione

Il progetto consiste nella realizzazione di un'architettura client-server che permetta all'utente di crearsi un profilo e gestire una propria rete sociale al fine di lanciare agli amici una sfida basata sulla traduzione di parole italiane in parole inglesi cercando di tradurne quanto più possibili. E' prevista inoltre una classifica con i punteggi realizzati dall'utente e dai suoi amici.

2 Implementazione Server

All'avvio del server, il programma crea un database all'interno del quale si potrà implementare la persistenza dei dati degli utenti nel corrispettivo file json "database.json". Caricherà online il servizio di registrazione dell'utente mediante **RMI**, cioè attraverso la creazione e l'esportazione di un registry, quindi si metterà in ascolto su una **socket TCP** in attesa del login di un utente. Ogni utente è gestito esclusivamente dal server mediante un **thread** dedicato (facente parte di un pool di thread) che soddisferà ogni sua richiesta.

2.1 Database (supportato da User e Code)

Il database dell'architettura è costituito da un **HashMap**, struttura dati basata sulle coppie chiave-valore, dove la chiave è rappresentata dall'username che l'utente dà al momento della registrazione e alla quale è associato un oggetto **User**. Un oggetto User contiene tutte le informazioni principali dell'utente, le quali saranno salvate nel file di persistenza (es. l'username, la password, il punteggio, la lista di amici).

La persistenza del database o il suo recupero è affidato alle librerie **JSONSimple** e **Gson**, le quali salveranno dati e punteggi aggiornati di un utente oppure restituiranno la lista di amici o la classifica su richiesta dall'utente stesso.

Inoltre la maggior parte dei metodi all'interno della classe Database ha come valore di ritorno uno dei codici presenti all'interno della classe **Code**, al fine di informare il client della riuscita (o del fallimento) delle azioni richieste dell'utente.

2.2 Task

Come detto nell'introduzione di questo capitolo, ogni utente è gestito da un thread esclusivo, rappresentato dalla classe Task. Tale classe interagirà con il client mediante la socket TCP creata dal server. La sua funzione è quella di fare da **filtro** tra il client e il database: il client, mediante particolari stringhe, informerà il server delle sue richieste; esso inoltrerà le richieste al database, il quale ritornerà dei codici di successo (o fallimento) che questa classe inoltrerà al client.

Nel caso in cui il client voglia sfidare un amico, la classe Task provvederà a creare un nuovo thread (**Challenge**) che gestisca la sfida, oltre **alla porta TCP** attraverso la quale far "viaggiare" le parole da tradurre.

Si occuperà inoltre, tramite Database, di informare l'amico dell'utente dell'eventuale sfida, che potrà essere accettata, rifiutata o ignorata (in tal caso un utente ha 15 secondi per accettare una sfida, dopo i quali verrà automaticamente rifiutata). Infine la classe Task, sempre mediante Database, informerà lo sfidante della risposta positiva o negativa dell'amico. Tutte queste transizioni di informazioni vengono implementate tramite una **socket UDP**.

2.3 Challenge (supportato da Session e TimerCh)

Avviato dalla classe Task, la classe Challenge si occuperà della sfida tra i due giocatori.

La prima funzione di tale classe è la scelta delle N parole da sottoporre agli sfidanti, scelte dal file json "**dizionario.json**" mediante la libreria Gson.

A questo punto creerà una **socket TCP** in modalità non bloccante, alla quale i clienti si collegheranno attraverso il **multiplexing** dei canali (tramute **NIO**). Appena gli utenti saranno collegati, il thread si salverà in un Arraylist la traduzione delle N parole scelte, tradotte con un **collegamento HTTP** con il servizio open-source MyMemory. In questa fase è anche azionato il timer della sfida tramite la classe **TimerCh**.

Il thread, mediante la select(), quindi si mette in attesa che i canali siano pronti, alternativamente, a scrivere o a leggere:

- in caso la chiave del selettore sia settata su **OP_WRITE**, il thread invierà la parola da tradurre oppure il codice di fine sfida o di time-out scaduto procedendo con la scelta del vincitore e l'assegnazione di punti bonus;
- in caso la chiave sia settata su **OP_READ**, il thread leggerà la parola scritta dall'utente e ne controllerà la correttezza assegnando i punteggi di conseguenza. Qualora al thread arrivasse la parola "EXIT", quest'ultimo chiuderà il canale corrispondente non restituendo alcun responso (i punteggi saranno comunque aggiornati).

In questo processo, la classe Challenge è supportato dalla classe **Session**, che rappresenta l'istanza dei dati di un utente giocatore della singola sfida, utili a costruire le statistiche finali della partita.

2 Implementazione Client

All'avvio del client, ricercherà subito il **servizio RMI** messo a disposizione dal server al fine di registrare l'utente. Quindi farà partire un'interfaccia grafica con la quale l'utente può facilmente interagire. Essa è gestita dalla libreria **JavaFX** ed è lanciata mediante la funzione "**launch**" che chiamerà a sua volta la funzione "start" che mostrerà a video il primo "stage", ovvero la StartView.

Il client gestirà quindi il caricamento delle varie interfacce (realizzato dalla **FXMLLoader**, che carica un file .fxml, creato mediante il programma SceneBuilder, contenente tutti i containers e i controlli della singola interfaccia).

Alla richiesta di login e alla sua risoluzione, il client creerà una **socket TCP** che si collegherà a quella del server al fine di mandare le richieste dell'utente. Sarà pertanto la classe Client a inviare al server tutti i codici che fanno riferimento a una richiesta dell'utente e a ricevere quelli di successo o fallimento dell'azione.

Inoltre, sempre in seguito al successo del login dell'utente, il client creerà anche una **socket UDP** (compresa la porta) e avvierà un nuovo thread (**Notify**), tutto al fine di gestire la ricezione delle notifiche.

3.1 Notify

Appena avviato, il thread Notify si mette in attesa di possibili sfide ricevute mediante la "receive".

Se ricevuta la stringa "CHALLENGE", allora il thread si occuperà di avvisare l'interfaccia di mostrare a schermo il nome dell'amico sfidante e di poter scegliere di accettare la sfida oppure rifiutarla. In entrambi i casi, sarà sempre questo thread ad avvisare il server della risposta mediante l'interazione con i pulsanti dell'interfaccia.

Il thread si occuperà anche di ricevere la risposta che l'amico sfidato darà di fronte alla possibilità di giocare una nuova partita:

- in caso di risposta positiva, il thread ci condurrà direttamente all'interfaccia di gioco;
- in caso di risposta negativa o di time-out scaduto, stamperà a schermo la notifica di rifiuto.

3.2 GUI

3.2.1 StartController

È la schermata iniziale del nostro gioco, presenta solo il pulsante START che ci condurrà alla pagina di Login.

3.2.2 LoginController

È la classica schermata per accedere al proprio profilo: presenta due TextField per l'immissione di username e password, il pulsante LOGIN che avvierà le procedure spiegate sopra e una scritta interattiva che ci permette di passare alla schermata di registrazione in presenza di un nuovo utente. Sopra la "field" dell'username, saranno stampati eventuali errori.

3.2.3 RegisterController

Simile alla schermata di Login, essa permetterà all'utente di registrare un nuovo username all'interno del database mediante il pulsante SIGN IN. A registrazione effettuata, è possibile tornare alla pagina di Login per accedere al servizio.

3.2.4 MainController

È la schermata principale del gioco. Qui si può controllare il punteggio personale, la propria lista di amici, ammirare una classifica decrescente con i punteggi dei propri amici, aggiungere un amico o sfidarlo mediante i rispettivi pulsanti SCORE, MY FRIENDS, RANKING, ADD, CHALLENGE. In questi due ultimi casi, è presente una TextField dove inserire l'username dell'amico. Sopra la TextField per l'invio della sfida, è presente uno spazio delle notifiche che risulterò visibile non appena un amico avrà invitato la sua richiesta. A quel punto si sbloccheranno anche i pulsanti ACCEPT e DECLINE, utili per accettare o rifiutare la sfida. Infine in basso a destra, è presente il tasto di LOGOUT, da cliccare sempre se si vuole poi accedere di nuovo al profilo senza ricompilare il server.

3.2.5 GameController

Si accede a questa schermata in seguito alla richiesta o all'accettazione di una sfida. Presente una TextField dove inserire la parola inglese, una ProgressBAR che comunicherà a che punto della sfida si è arrivati, quindi uno spazio relativo ai risultati finali. Il tasto EXIT ci permetterà di tornare alla schermata principale anche a sfida in corso, ma in tal caso non avremo alcun responso finale.

4 Compilazione ed Esecuzione

Per compilare ed eseguire il programma tramite terminale seguire i seguenti passaggi:

- Aprire il terminale nella cartella “src” di Word Quizzle

- Compilazione del server:
  javac -cp “../jars/ json-simple-1.1.1:../jars/gson-2.8.2:” Server.java

- Esecuzione del server:
  java -cp “../jars/ json-simple-1.1.1:../jars/gson-2.8.2:” Server

- Compilazione del client:
  javac -cp “../jars/ json-simple-1.1.1:../jars/gson-2.8.2:”
  – module-path ../javafx-sdk-11.0.2/lib/ --add-module ADD-MODULE-PATH Client.java

- Esecuzione del client:
  java -cp “../jars/ json-simple-1.1.1:../jars/gson-2.8.2:”
  – module-path ../javafx-sdk-11.0.2/lib/ --add-module ADD-MODULE-PATH Client