UNIVERSITÀ DI PISA

WQ:WordQuizzle

Relazione del progetto - Laboratorio di Reti

Selene Gerali - matricola 546091

Marzo 2020

Contents

1 Introduzione

Il progetto richiede di sviluppare un sistema di sfide di traduzioni italiano - inglese tra gli utenti registrati al servizio. E' richiesta la gestione di una rete sociale, nella quale registrandosi al servizio, un utente può loggarsi, aggiungere o rimuovere amicizie, controllare la classifica generale, o visualizzare il proprio punteggio. Inoltre un utente ha la possibilità di sfidare un amico che necessariamente anch'esso deve essere registrato al servizio. L'obiettivo della sfida è la traduzione corretta del maggior numero di parole italiane proposte dal servizio. Tutte queste operazioni sfruttano il protocollo TCP, mentre i messaggi scambiati tra client e server per la richiesta di sfida seguono un protocollo UDP.

2 Architettura Complessiva e scelte implementative

La principale scelta di progetto da compiere, è stata quella relativa alla comunicazione sulla rete, tra l'uso di primitive di I/O bloccanti (e quindi il ricorso al multhithreading per gestire più connessioni) e il ricorso al multiplexing (Utilizzo quindi di channels e selectors forniti da NIO). Nell'implementazione presentata si è scelto di adottare un'architettura del primo tipo, ricorrendo alle chiamate del pacchetto java.io e a un pool di thread, ciascuno dei quali si occupa di eseguire solo le richieste provenienti da un particolare client. La ragione di questa scelta sta nella maggior semplicità d'uso di chiamate di funzione della libreria java.io, nella minor possibilità di commettere errori di programmazione e nella minor complessità del codice risultante, qualità che in questo caso sono state ritenute più importanti dell'efficienza e della scalabilità nell'uso delle risorse di memoria e di calcolo, che indubbiamente sarebbero state migliori se si fosse adottato il multiplexing.

2.1 Serializzazione

Per serializzare e trasformare i dati dell'intero sistema in e da JSON, è stata utilizzata la libreria Gson. Alla creazione del server viene controllata l'esistenza dei file di persistenza, nel caso in cui i file siano già presenti, con l'utilizzo della libreria Gson, vengono ricreate tutte le istanze delle strutture dati da mantenere consistenti. Questi file vengono aggiornati ogni volta che viene modificata la principale struttura dati contenente tutte le informazioni dei singoli utenti. Questa scelta può risultare computazionalmente costosa ma sicura nell'eventualità di crash da parte del server, poiché i dati rimarranno consistenti anche al termine dell'esecuzione della JVM.

2.2 Sincronizzazione

Per assicurare che sul lato server non si verifichino sovrapposizioni tra scritture o letture, è stato utilizzato il meccanismo ad alto livello offerto da Java del

metodo “synchronized”. Quando il metodo synchronized viene invocato, se nessun metodo synchronized della classe è in esecuzione, l’oggetto viene bloccato (lock acquisita sull’oggetto) ed il metodo viene eseguito. Se l’oggetto è bloccato, il thread viene sospeso nella coda associata all’oggetto fino a che il thread che detiene la lock la rilascia.

2.3 Interfaccia Utente

Per l’interazione con l’utente, si è scelto di adottare un’interfaccia grafica sviluppata usando la libreria JavaSwing con l’aiuto del plugin WindowBuilder, in modo da rendere la comunicazione tra client e server più intuitiva e User-friendly. WindowBuilder è un plugin creato per l’IDE Eclipse, con lo scopo di semplificare la creazione di interfacce grafiche Java (come SWING e AWT) mediante un’interfaccia visuale intuitiva e semplice da utilizzare.

3 Struttura del codice e delle classi

3.1 Server

Server
Il server appena avviato crea una nuova istanza di Databaseperation. Prima di mettersi in ascolto di client, crea ed esporta il registry per la registrazione attraverso RMI, successivamente crea una nuova istanza della classe DatabaseOperation e utilizza un ThreadPoolExecutor con LinkedBlockingQueue per limitare il numero di client connessi ed evitare il collasso dell’intera macchina a causa della creazione di infiniti thread. Appena il server accetta un client, si crea un nuovo thread: ThTask che si occupa dell’intera esecuzione del singolo client. Nel caso in cui la pool è al completo, il client attende il proprio turno all’interno della coda.

ThTask
Questo thread si occupa della gestione di tutte le richieste del singolo client. Ogni richiesta arrivata dal client, viene tokenizzata e viene chiamata la funzione di riferimento implementata all’interno della classe DatabaseOperation.

DatabaseOperation
Questa classe si occupa della gestione di tutti i dati dei client e della creazione e aggiornamento dei file Json di persistenza. All’interno di questa classe sono presenti due strutture dati:
private HashMap<String, String> passwords
private HashMap<String, User> users
Per quanto riguarda la prima struttura dati, la chiave è rappresentata dal nome dell’utente e il valore dalla password (per garantire maggiore sicurezza le password vengono crittografate attraverso l’hashing one-way SHA-256 concatenando nome e password). La seconda struttura dati invece ha come chiave sempre il

nome dell'utente e come valore un'istanza di User. La struttura dati users è necessaria per mantenere tutte le informazioni di ogni singolo utente registrato al sistema.

User
User è la classe dedicata ai dati del singolo utente. In essa, oltre a nome utente e password, sono contenute anche altre importanti informazioni, come: un array per memorizzare gli amici dell'utente, la porta UDP per i messaggi di richiesta di sfida, la socket sulla quale il client è in ascolto, una variabile dedicata all'indirizzo IP, un'altra variabile booleana che indica se l'utente è loggato o meno, e una che indica se l'utente è occupato in una sfida; inoltre sono contenute alcune informazioni necessarie per la sfida, come: i punteggio della singola partita, i punti totali ed infine una HashMap in cui vengono memorizzate le parole da tradurre con le relative traduzioni che viene inizializzata ogni volta che un utente viene coinvolto in una sfida.

3.2 Client

Gli stati del client possono descritti attraverso un automa a stati finiti come rappresentato di seguito:

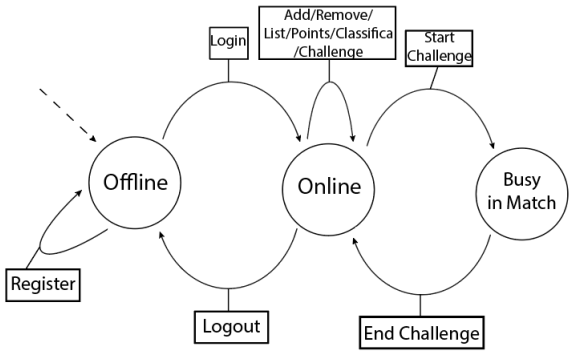

Figure 1: ASF che descrive gli stati del client

Client
Il client interagisce con il server attraverso un'interfaccia grafica, la classe Client avvia l'interfaccia iniziale definita nella classe: GuiRegisterLogin. Inoltre contiene tutti i metodi che si occupano di comunicare con il server tramite socket

TCP standard bidirezionale. La connessione viene aperta quando il client invoca il metodo login ed è persistente, ovvero rimane aperta fino all'invocazione del comando logout.

GUIRegisterLogin
Interfaccia che permette al client di effettuare la registrazione e il login al servizio WordQuizzle.

Figure 2: Interfaccia iniziale di WQ

Questa interfaccia è composta da due JTextField e 2 Jlabel che sono elementi che non generano alcun tipo di evento, a differenza dei due JBotton: Register e Login che provocano l'esecuzione di due eventi, i cui compiti sono la gestione della registrazione e del login dell'utente chiamando i metodi di riferimento: "user-registration", metodo dell'oggetto remoto e "login", metodo implementato nella classe Client. In particolare il metodo "login" si occupa di creare la connessione TCP con il server; nel caso un cui il login termina correttamente, viene creato e attivato un Thread: UDPcontroller che si occupa della comunicazione tra client e server per quanto riguarda la richiesta di sfida e la relativa risposta. Attraverso il metodo "showMessageDialog" in entrambi gli eventi faccio visualizzare l'esito delle operazioni effettuate.

GUIOperation
Interfaccia, attivata solo se il login dell'utente è andato a buon fine, contiene i bottoni per le principali richieste da effettuare al server. Attraverso questa interfaccia, l'utente può aggiungere o rimuovere un amico semplicemente inserendo in nome interessato nella Jlabel corrispondente. E' possibile inoltre richiedere la visualizzazione della classifica e del proprio punteggio, oppure effettuare il logout dal servizio di WQ. Tutte queste operazioni vengono effettuate attraverso i bottoni di riferimento che invocano a loro volta gli eventi relativi, i quali hanno il compito di invocare i metodi dedicati definiti anch'essi nella classe Client. Nella parte destra dell'interfaccia, è possibile caricare e visualizzare nella JList la lista degli amici di un utente semplicemente cliccando il bottone di riferimento; selezionando un nome-utente presente all'interno della Jlist è possibile inviare una richiesta di sfida a quest'ultimo attraverso l'apposito bottone: Challenge, che come tutti gli altri invocherà un evento che andrà a chiamare il metodo challenge implementato nella classe Client.

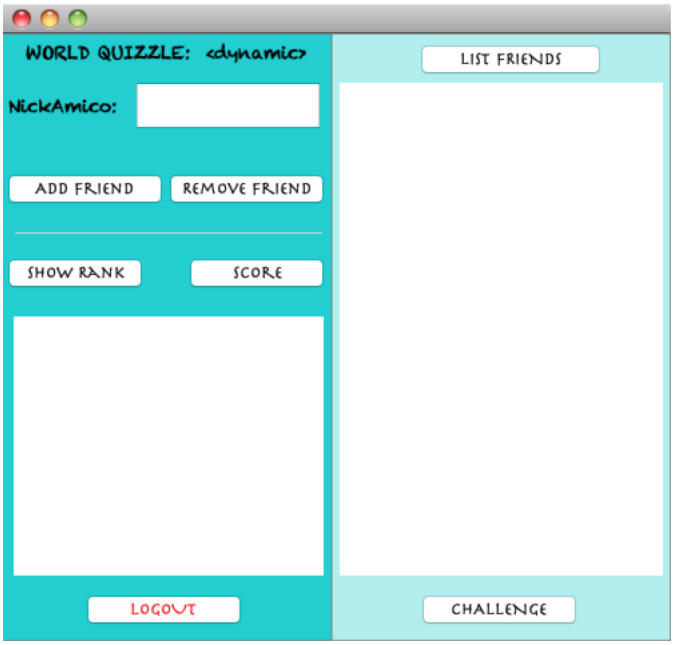

Figure 3: Interfaccia principale di WQ

4 Sfida

Come richiesto, la sfida viene implementata inviando la richiesta tramite UDP da parte del server allo sfidato. Lato client le richieste di sfida sono gestite da un thread apposito: UDPcontroller che si occupa della gestione di tutte le richieste in arrivo dal server. Come già anticipato, questo thread viene messo in esecuzione una volta che l'utente ha effettuato il login, passandogli come parametro la datagramsocket. Il thread è bloccante sulla receive della socket in attesa di un nuovo datagramma. Non appena arriva un messaggio di tipo "CHALLENGE", controlla che l'utente non sia già occupato in un'altra sfida, nel caso rifiuta la richiesta automaticamente, altrimenti avvia l'interfaccia di richiesta di sfida.

Figure 4: Interfaccia di richiesta di sfida

Una volta avviata l'interfaccia, il client dello sfidato può decidere se accettare o rifiutare la sfida attraverso i bottoni appositi, se la risposta non viene data entro 10 secondi (T1), scatta il timeout e la sfida viene rifiutata automaticamente. Il server appena riceve la risposta alla richiesta di sfida, avvisa i due client inviando l'esito del messaggio attraverso un pacchetto UDP; nel caso la sfida fosse stata accettata, il client si preoccupa di avviare l'interfaccia grafica della partita: GUIChallenge, mentre il server seleziona K parole casuali dal file "Words", che contiene (N) 100 parole, e richiede le traduzioni al servizio esterno mymemory.translated.net la cui chiamata al servizio REST avviene sfruttando la classe HTTPsURLconnection e settando il metodo di richiesta a "GET". Per ogni parola italiana viene memorizzata la lista delle traduzioni fornite dal servizio. Al termine della memorizzazione delle traduzioni di tutte le K parole, il server avvia un thread: ThChallenge che gestisce la sfida, il quale si occupa di fornire ai due utenti le parole da tradurre e attende che entrambi gli utenti

abbiano finito la sfida.

Figure 5: Interfaccia di gioco di WQ

Dopo di che, appena il server riceve dai due utenti la stringa "STARTCH", il server da inizio alla partita vera e propria invocando il metodo "startchallenge". Il metodo "startchallenge" si occupa di inviare all'utente le parole da tradurre e attende un messaggio di risposta. Il client concede all'utente 10 secondi per inviare la traduzione, la sfida dunque può durare al massimo 10*K secondi (T2). Se il messaggio di risposta contiene una traduzione corretta, assegna all'utente (X) 3 punti, se la risposta è sbagliata toglie (Y) 1 punto, mentre se scade il timer, il client risponde automaticamente con "TIMEOUT" e non viene assegnato nessun punto.

Non appena entrambi gli utenti concludono la sfida, il thread (ThChallenge) si occupa di decretare il vincitore in base a chi dei due utenti ha ottenuto il punteggio più alto invocando il metodo "challengeend". Il vincitore otterrà (Z) 5 punti bonus. L'arbitro invia il risultato ai due utenti, i quali nel frattempo hanno ricevuto il payload "CHALLENGEEND" per mettersi in attesa dei risultati. Appena ricevono l'esito, li mostrano nell'interfaccia grafica.
Nel caso in cui l'utente abbandona la sfida attraverso il bottone "X" (posizionato in alto a destra dell'interfaccia), viene impostata la scena della schermata principale e viene valutata la partita conteggiando il punteggio fino al momento in cui l'utente è arrivato prima dell'uscita.

5 Protocollo di comunicazione

Il servizio di registrazione viene implementato tramite RMI, ed è l'unico tipo di comunicazione che non sfrutta le socket TCP/UDP. Il server crea un'istanza dell'oggetto remoto, esporta l'oggetto remoto e successivamente associa all'oggetto un nome simbolico e crea un registry sulla porta 30000. Il client a sua volta effettua la lookup per ottenere il riferimento all'oggetto remoto in modo da poter chiamare il metodo dell'oggetto come fosse un metodo locale.

Per il resto del progetto, sono state utilizzate:

- socket TCP per l'invio e la ricezione di richieste e risposte tra client e server
- socket UDP per la comunicazione tra client e server riguardo alla richiesta di sfida

5.1 Comunicazione TCP

Su questa connessione TCP, dopo previa login effettuata con successo, avvengono le interazioni richieste-risposte tra client-server. Di seguito si descrive lo schema del protocollo di rete e la sintassi dei messaggi inviati attraverso questa connessione. (Lo schema differisce dalla comunicazione dedicata alla sfida).

Le possibili richieste effettuate dal client, con le rispettive risposte da parte del server possono essere:

- **LOGIN NomeUtente Password Host-address UdpPort**. Restituisce un codice corrispondente all'esito dell'operazione.
- **ADDFRIEND NomeUtente NomeAmico**. Restituisce un codice corrispondente all'esito dell'operazione.
- **REMOVEFRIEND NomeUtente NomeAmico**. Restituisce un codice corrispondente all'esito dell'operazione.
- **LISTFRIEND NomeUtente**. Restituisce una lista contenente tutti i nomi degli amici dell'utente.
- **CHALLENGE NomeUtente NomeAmico**. Restituisce un codice corrispondente all'esito dell'operazione.
- **SCORE NomeUtente**. Restituisce il punteggio totale dell'utente.
- **RANK NomeUtente**. Restituisce la classifica (composta da nome utente e punteggio di tutti gli amici) aggiornata in ordine decrescente.
- **LOGOUT NomeUtente**. Restituisce un codice corrispondente all'esito dell'operazione.

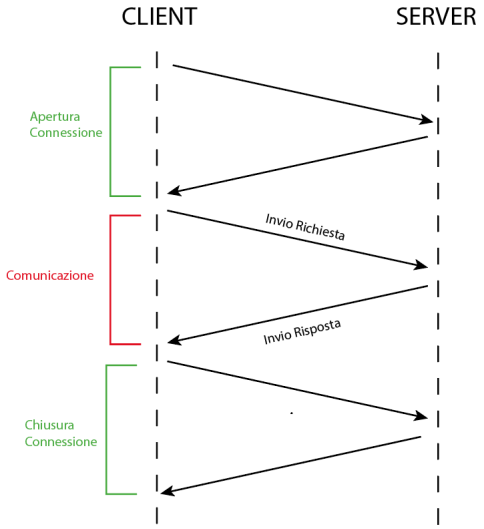

Figure 6: Comunicazione client-server tramite protocollo TCP

La comunicazione tra client e server durante la sfida può essere descritta in questo modo:

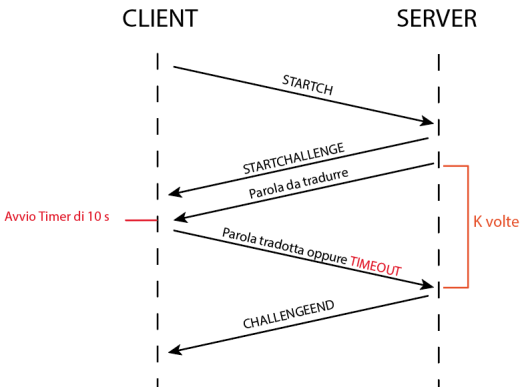

Figure 7: Comunicazione sfida client-server tramite protocollo TCP

Il server appena riceve “STARTCH” dal client si occupa di dare inizio alla sfida inviando a sua volta “STARTCHALLENGE”, dopo di che inizia lo scambio

Request-Response: inizia un ciclo ripetuto K volte (pari al numero delle parole da tradurre), nel quale il server invia la parola italiana e di conseguenza il client risponde con una stringa contenente la parola tradotta in inglese e il proprio nome utente, oppure nel caso in cui scade il timer di 10 secondi viene inviata automaticamente la stringa “TIMEOUT” al server. Terminato il ciclo, il server avvisa il client della fine della sfida con il messaggio “CHALLENGEEND”, che a sua volta si mette in attesa dei risultati ottenuti.

5.2 Comunicazione UDP

Il protocollo di richiesta di sfida tramite UDP, ha inizio quando lo sfidante invia al server attraverso connessione TCP standard, il messaggio "CHALLENGE sfidante sfidato”. Successivamente il server invia tramite UDP allo sfidato il messaggio ”CHALLENGE sfidante”. Il client dello sfidato, ricevuto il messaggio di sfida, può decidere se accettare, rifiutare o ignorare la richiesta di sfida. Di seguito si descrivono le 3 possibili situazioni:

1. Lo sfidato accetta la richiesta di sfida:

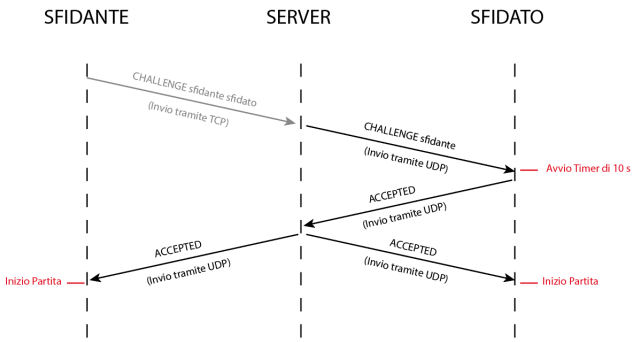

Figure 8: Comunicazione client-server tramite protocollo UDP

Lo sfidato invia al server ACCEPTED e successivamente il server notifica ad entrambi gli utenti (sempre attraverso comunicazione UDP) l’accettazione della sfida in modo da prepararsi per l’inizio della partita.

2. Lo sfidato rifiuta la richiesta di sfida:

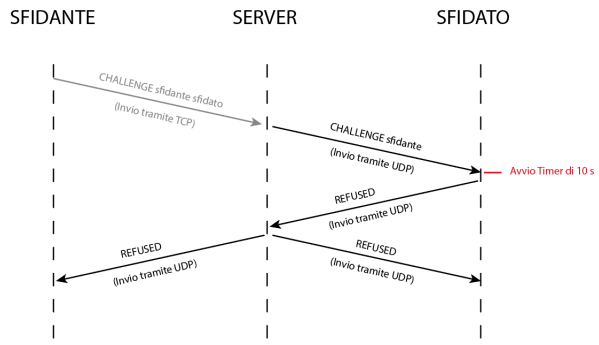

Figure 9: Comunicazione client-server tramite protocollo UDP

Lo sfidato invia al server REFUSED e successivamente il server notifica ad entrambi (sempre attraverso comunicazione UDP) gli utenti il rifiuto della sfida.

3. Scade il timer:

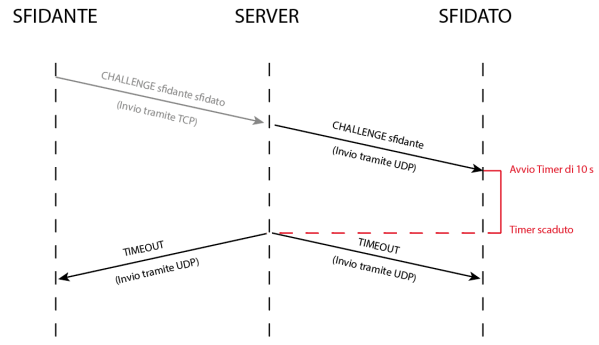

Figure 10: Comunicazione client-server tramite protocollo UDP

Lo sfidato non risponde alla richiesta di sfida: la socket UDP del server

non riceve risposta entro il tempo T1 (10 s) e scatta il timeout. Il server invia "TIMEOUT" in un datagramma a entrambi gli utenti.

6 Istruzioni per la compilazione e l'esecuzione

Il progetto è stato sviluppato attraverso l'IDE Eclipse utilizzando come librerie esterne la libreria GSON (com.google.code.gson) in versione 2.8.2. e json-simple in versione 1.1.1 (sarà necessario specificare tali librerie nel build path per poter compilare il progetto), e testato su ambiente MacOS. Per compilare ed eseguire il server è necessario spostarsi nella directory WordQuizzle/src.

Compilazione del server:

javac -cp ".:../lib/json-simple-1.1.1.jar:gson-2.8.2.jar" Server.java

Esecuzione del server:

java -cp ".:../lib/json-simple-1.1.1.jar:gson-2.8.2.jar" Server

Per compilare ed eseguire il client è necessario spostarsi nella directory WordQuizzle/src.

Compilazione del client: (-Xlint necessario per l'avvio dell'interfaccia grafica):

javac -Xlint -cp ".:../lib/json-simple-1.1.1.jar:gson-2.8.2.jar" Client.java

Esecuzione del client:

java -cp ".:../lib/json-simple-1.1.1.jar:gson-2.8.2.jar" Client