

# Wordle - Nannoni

Progetto Esame Lab. III - Wordle “un gioco di parole”.

Federico Nannoni MAT. **617376**.

## Scelte effettuate

Comunicazione

Caricamento da file

Server

Avvio server

Inizio partita

Cambio parola e salvataggio dati

Interruzione e chiusura server

Client

Strutture dati

Server

Client

Thread

Server:

Client:

Comandi

Compilazione

Client

Server

Eseguibile

Client

Server

Comandi gioco

## Scelte effettuate

### Comunicazione

Per l'interazione tra client e server ho utilizzato una connessione tra socket TCP.

Il client manda i vari comandi dell'utente al Server, il quale li interpreta e restituisce una risposta adeguata.

Sia il client che il server aprono un socket ad un indirizzo e una porta specificati (nei file di config.), utilizzano un `PrintWriter/Scanner` per rispettivamente scrivere nel socket/leggere dal socket. Il server ad ogni nuova connessione farà partire un nuovo thread (gestiti da un `ThreadPool`) che andrà a servire il client.

Il client si occupa di leggere i messaggi scritti dall'utente per mandarli all'utente.

solo in alcuni casi si intromette in questa comunicazione:

Quando vengono mandati i seguenti comandi:

- **exit**: per chiudere il client.
- **logout**: per cancellare i dati dell'utente (username, e messaggi ricevuti da gruppo multicast).
- **showMeSharing**: per mostrare in console i messaggi (ricevuti dal gruppo multicast).

Quando riceve le risposte dei seguenti comandi:

- **login**: se il login va a buon fine, il server restituisce l'indirizzo il e porta multicast, il client fa partire il thread che se ne occuperà
- **sendMeStatistics**: il client si occuperà di parsare il messaggio di risposta in modo da renderlo interpretabile dall'utente.

## Caricamento da file

La classe *FileManager.java* si occupa di caricare e salvare le strutture dati su file, e di caricare le parole. Per gli utenti semplicemente serializza e de-serializza un file *utenti.json*.

Mentre per i file di configurazione, essendo statici (ovvero che i nomi delle impostazioni non cambiano, ma possono cambiare i loro valori) vengono letti direttamente all'interno delle relative classi (*ClientMain.java* e *ServerMain.java*).

---

## Server

### Avvio server

Il server quando viene avviato:

1. carica dal file di configurazione le varie informazioni di cui ha bisogno (es. porta, numero massimo di tentativi...).
2. Apre il socket.

3. Crea un'istanza di `ServerMain` (cariando in memoria gli utenti e le parole)
4. Aggiunge un hook che gestisce il segnale di terminazione (`SIGINT`).

```
Runtime.getRuntime().addShutdownHook(new Thread() {<codice da eseguire>});
```

5. Manda in esecuzione il thread che si occupa di cambiare la parola.

```
pool.execute(new Thread(){
    ...
    while(true){
        try {
            Thread.sleep(sm.getTIME_TO_NEW_WORD()*1000);
            sm.changeWord();
        }
        ...
    }
});
```

6. Manda in esecuzione il thread che si occupa di leggere e inviare i messaggi degli utenti sul gruppo multicast UDP.

```
pool.execute(new MulticastUDPManager(sm.messaggiUDP, sm.UDP_INFO ));
```

7. Si mette in **accept**, per accettare le connessioni da nuovi client. Creando un thread per ogni nuova connessione.

```
while (true) {
    pool.execute(new ServerTask(listener.accept(), sm));
}
```

---

## Inizio partita

La partita inizia all'avvio del server, quando viene creata l'istanza di `ServerMain`. Resetando le statistiche e impostando la nuova parola.

---

## Cambio parola e salvataggio dati

Ad ogni cambio di parola, vengono aggiornate le statistiche di gioco, cancellate quelle della partita appena finita e imposta una nuova parola. In modo che ogni thread che gestisce un client possa riconoscere questo cambio e comunicarlo al

client (così poi il player potrà partecipare alla nuova partita). La variabile che contiene la parola è di tipo **volatile Integer** in modo che ogni cambiamento che riceve, venga effettuato in modo atomico e aggiornato per tutti i thread che provano a leggerla/modificarla.

---

## Interruzione e chiusura server

Quando il server riceve un segnale di terminazione SIGINT, il thread che si occupa della terminazione viene eseguito disconnettendo gli utenti dalla sessione attuale, salvando i dati fino all'ultima partita terminata (se la partita è ancora in corso, non viene salvata. In modo da non avvantaggiare/svantaggiare nessun utente)

---

## Client

Il client si occupa in generale solo di fare da tramite tra utente e server, solo per alcuni casi si intromette. (Questi casi sono spiegati in "[Comunicazione](#)").

All'avvio, carica le informazioni necessario dal file *clientConfig.json*.

Una volta che il login ha avuto successo, riceve dal server indirizzo e porta UDP e fa partire il thread che si occupa di ricevere e salvare i messaggi dal gruppo multicast. Per poi stamparli a video quando l'utente lo richiede (tramite il comando "*showMeSharing*").

In caso di logout stoppa il thread e cancella i messaggi in modo da eliminare le informazioni del precedente utente

---

## Strutture dati

### Server

Il server usa varie struttura dati:

```
private ConcurrentLinkedQueue<String> messaggiUDP;  
private ConcurrentHashMap<String, User> users;  
private ConcurrentHashMap<String, String> words;  
private ConcurrentHashMap<String, gameStat> gameStats;
```

- **messaggiUDP:**

Lista concorrente che server per salvare i messaggi da mandare nel gruppo multicast.

- **users:**

HashMap concorrente per la gestione degli utenti registrati presso il servizio.

- **words:**

HashMap concorrente per la gestione delle parole. In modo che la ricerca sia immediata.

- **gameStats:**

HashMap concorrente utilizzata per gestire le partite attualmente in corso.

- **gameStat:**

Struttura dati utilizzata per salvare i dati di una singola partita in corso.

Tutte le strutture dati del server sono Thread Safe, perché vengono accedute in modo concorrente da più thread alla volta.

## Client

Il client usa una sola struttura dati:

```
private LinkedList<String> messages;
```

Viene utilizzata per salvare tutti i messaggi che riceve tramite il gruppo multicast a cui viene iscritto in fase di login.

## Thread

Durante l'esecuzione vengono fatti partire varie thread:

### Server:

- Per ogni client che si connette viene fatto partire un thread che si occupa di gestire i comandi che riceve
- Inoltre vengono fatti partire altri 3 thread che si occupano:
  - Gestione segnale SIGINT: quando viene ricevuto questo segnale, il thread viene messo in esecuzione e fa terminare il server in modo pulito,

disconnettendo tutti gli utenti e salvando le varie strutture dati su file.

- Gestione cambio parola: questo thread cambia la parola ogni X secondi, salva le varie informazioni della partita e resetta i dati di gioco attuali ad ogni cambio di parola.
- Gestore gruppo multicast: si occupa di registrare il server al gruppo multicast e di leggere i vari messaggi dalla struttura dati messaggiUDP per poi effettivamente mandarli al gruppo.

## Client:

Il client fa partire un thread quando il login ha successo, questo thread si occupa di unirsi al gruppo multicast, di ricevere e salvare tutti i messaggi.

---

# Comandi

## Compilazione

Per poter compilare ed eseguire il progetto, la directory deve essere così strutturata:

```
✓ PROGETTO WORDLE
  ✓ codice_sorgente
    > client
    > lib
    > server
  ✓ jar_eseguibile
    > client
    > server
```

Invece i file devono essere disposti in questo modo:

```

✓ PROGETTO WORDLE
  ✓ client
    {} clientConfig.json
    J ClientMain.java
    J MulticastReceiverWorker.java
  ✓ lib
    ■ gson-2.2.2.jar
  ✓ server
    J FileManager.java
    J gameStat.java
    J MulticastUDPManager.java
    J prova.java
    {} serverConfig.json
    J ServerMain.java
    J ServerTask.java
    J ServerWordChange.java
    J User.java
    {} user.json
    ≡ words.txt
```

(é possibile spostare i file di config e i file delle strutture dati. Bisogna però modificare i path all'interno di ClientMain.java e ServerMain.java)

## Client

Dalla directory principale:

```
cd codice_sorgente/client/
```

- compilazione:

```
javac -cp ../lib/gson-2.2.2.jar:. ClientMain.java MulticastReceiverWorker.java
```

- esecuzione:

```
java -cp ../lib/gson-2.2.2.jar:. ClientMain
```

## Server

Dalla directory principale:

```
cd codice_sorgente/server/
```

- compilazione:

```
javac -cp ../lib/gson-2.2.2.jar ServerMain.java FileManager.java ServerTask.java \
gameStat.java MulticastUDPManager.java ServerWordChange.java User.java
```

- esecuzione:

```
java -cp ../lib/gson-2.2.2.jar:. ServerMain
```

## Eseguibile

### Client

```
cd jar_eseguibile/client
```

- Esecuzione:

```
java -jar Wordle_server.jar
```

### Server

```
cd jar_eseguibile/server
```










- Esecuzione:

```
java -jar Wordle_client.jar
```

## Comandi gioco

I comandi sono i seguenti (sono state aggiunte delle emoji per aiutare la memoria visiva):

- ->  login <username> <password>
- ->  register <username> <password>
- >  playWORDLE (inizia una nuova partita)
- ->  sendWord <parola> (manda una parola)
- ->  sendMeStatistics (ricevi le tue statistiche)
- ->  share (condividi i risultati delle tue partite)
- ->  showMeSharing (guarda le statistiche condivise dagli altri utenti)

I comandi sono già spiegati dal client, però in generale per poter giocare abbiamo bisogno di un account che può essere creato con:

- register <username> <password>

Una volta creato l'account dovremo eseguire il login tramite:

- login <username> <password>

Se tutto va a buon fine (password giusta, utente creato in precedenza) verremo autenticati e potremo iniziare a giocare tramite:

- playWORDLE

Questo comando va eseguito ogni volta che vogliamo partecipare ad una nuova partita (es. se stiamo partecipando ad una partita, chiudiamo il client e ci ricollegiamo mentre la partita è ancora in corso, non sarà necessario ri-mandarlo, perché staremo già partecipando alla partita. Se invece cambia la parola sarà necessario partecipare alla nuova partita e quindi dovremo mandare il comando).

Una volta entrati in partita potremo mandare le nostre parole attraverso il comando:

- sendWord <parola>

Il server ci restituirà le informazioni sulla correttezza della parola:

- non presente nel vocabolario: la parola non è presente nel vocabolario di WORDLE, il tentativo non verrà conteggiato
- parola presenta ma sbagliata: verranno restituiti degli indizi sulla correttezza della singole lettere
- Vittoria: hai indovinato la parola.

Se vogliamo consultare le nostre statistiche, per sapere a quante partite abbiamo partecipato ecc. Possiamo usare il comando:

- sendMeStatistics

Ci restituirà le varie informazioni e una piccola tabella che ci mostra la distribuzione delle nostre vittorie

```
+---Statistic---+
Partite giocate: 20
win: 12
streak: 0
best streak: 4
1: ** , 6
2:  , 0
3:  , 0
4:  , 1
...
12:  , 0
```

Se invece vogliamo condividere queste statistiche con tutti gli utenti collegati, possiamo utilizzare il comando:

- share

Se invece vogliamo visualizzare tutte le statistiche condivise dagli altri utenti possiamo utilizzare il comando:

- showMeSharing

Se qualcuno ha condiviso le proprie statistiche verranno visualizzate, altrimenti avremo un messaggio *“Nessun messaggio da leggere”*. Attenzione, dopo aver visualizzato le statistiche condivise, queste verranno cancellate. In modo da fare spazio a nuovi messaggi più recenti.

- Exit

Esegue il log-out e chiude il client.