Relazione Progetto per Laboratorio di Reti

Leonardo Brugnano

14 dicembre 2022

Contents

Part I

Panoramica generale

Chapter 1

Client

Il client prevede una dinamica di funzionamento prevalentemente sequenziale, con l'eventuale attivazione di un thread secondario per la ricezione e memorizzazione temporanea di messaggi di condivisione da parte di altri client.

1.0.1 Inizio

Come prima cosa, il client stabilisce la connessione con il server. In caso di fallimento, esso termina. Quindi, mostra all'utente le azioni possibili:

- registrare un nuovo account utente;
- accedere ad un account utente esistente;
- chiudere la connessione e terminare il processo.

Nota bene

Ogni input numerico da parte dell'utente viene controllato dal client prima di inoltrarlo al server, al fine di verificarne la validità. Invece, gli input testuale (username, password, etc.) viene interamente controllato dal server.

Nota bene

Per ogni dato inviato al server, il client aspetta di ricevere un **feedback** in risposta, al fine di verificare che il server abbia ricevuto correttamente il dato e quindi capire cosa fare successivamente.

Di seguito possiamo osservare il diagramma di flusso (figura 1.1) descrivente il comportamento iniziale del client. Alcune osservazioni preliminari si rendono necessarie:

- Per questi ed i futuri diagrammi di flusso, non è stata generalmente completamente rispettata la notazione standard. Comunque, la comprensione dei diagrammi dovrebbe risultare immediata.
- Nei diagrammi ogni output al server è etichettato con un tag della forma `X-Y`. Per ora questo fatto può essere ignorato.
- I diagrammi che vedremo sono da considerarsi puramente indicativi nei confronti dell'implementazione effettiva delle entità software.

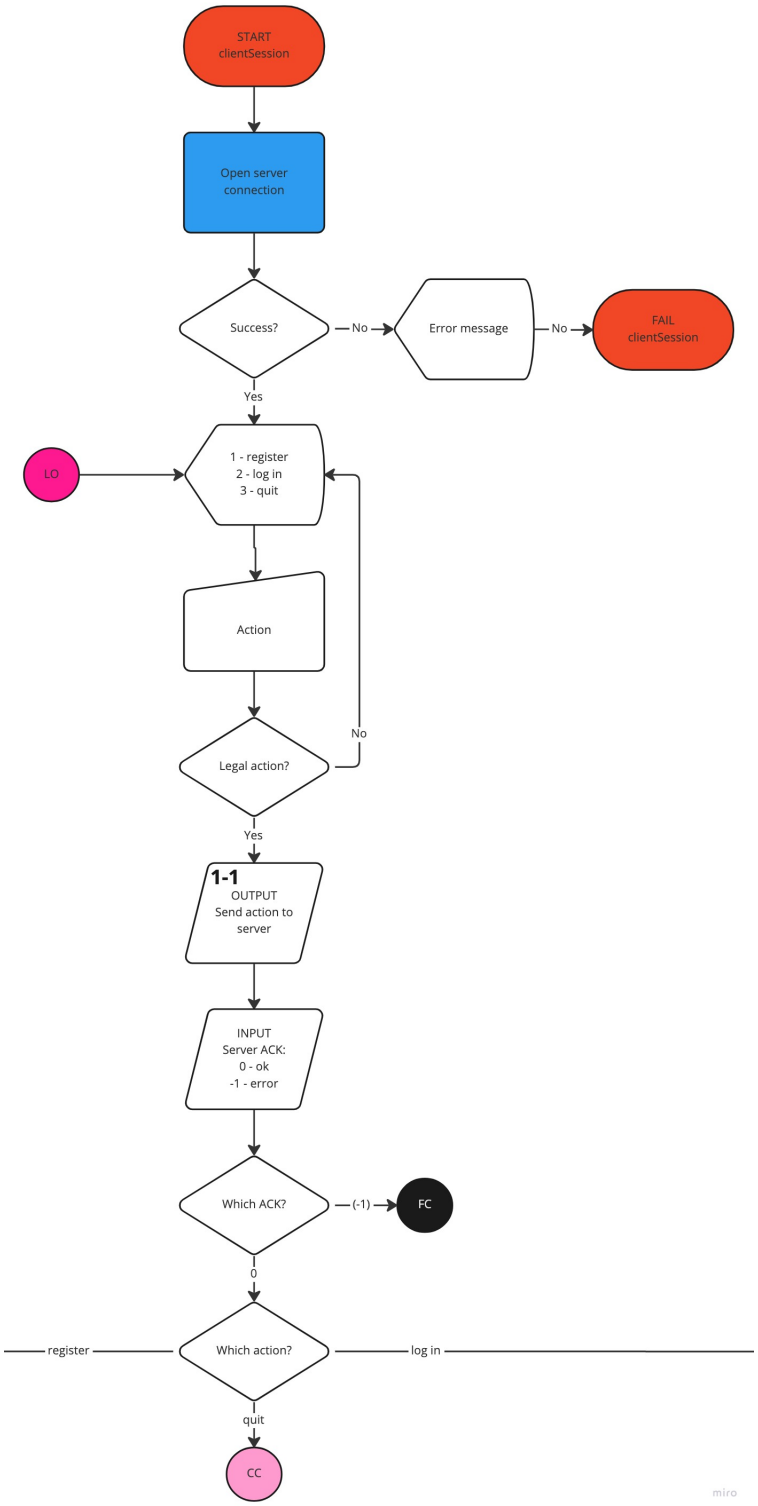

Figure 1.1: Comportamento iniziale del client.

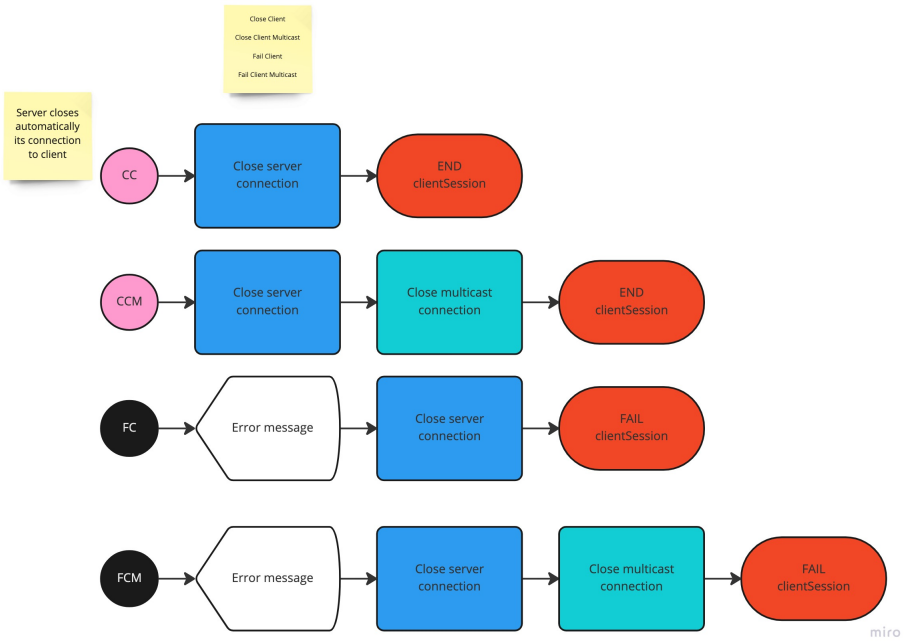

Figure 1.2: Sequenze di terminazione.

1.0.2 Registrazione

La registrazione (figura 1.3) richiede di inserire uno username univoco ed una password. In particolare, il client si occupa solamente di ricevere l'input dall'utente ed inoltrarlo al server. Esso non effettua alcun tipo di controllo sull'input, dato che i parametri di validità sono decisi interamente dal server.

In particolare, la registrazione e divisa in tre fasi:

1. **Username:** lo username inserito dall'utente viene controllato dal server, al fine di verificare che non sia già esistente.

2. **Password:** la password viene anch'essa controllata dal server, il quale impone una lunghezza minima.

3. **Confirm password:** viene chiesta una conferma della password, quindi il server controlla che le due password fornite corrispondano.

1.0.3 Login

L'accesso (figura 1.4) richiede di inserire username e password. Come per la registrazione, ogni controllo sull'input viene effettuato dal server.

In questo caso, il server decide un numero massimo di volte in cui l'utente può provare ad indovinare la password. Se l'utente sbaglia password troppe volte, la sessione viene chiusa.

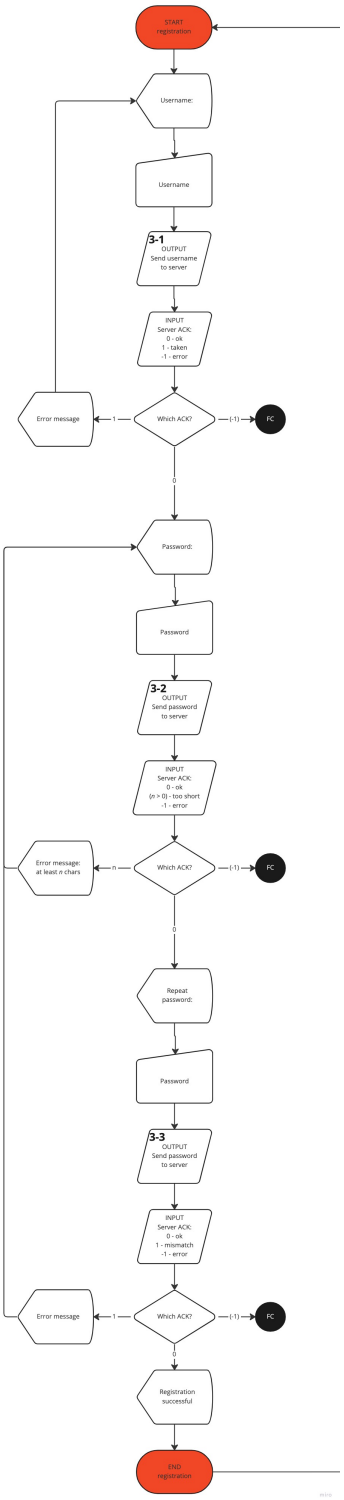

Figure 1.3: Registrazione.

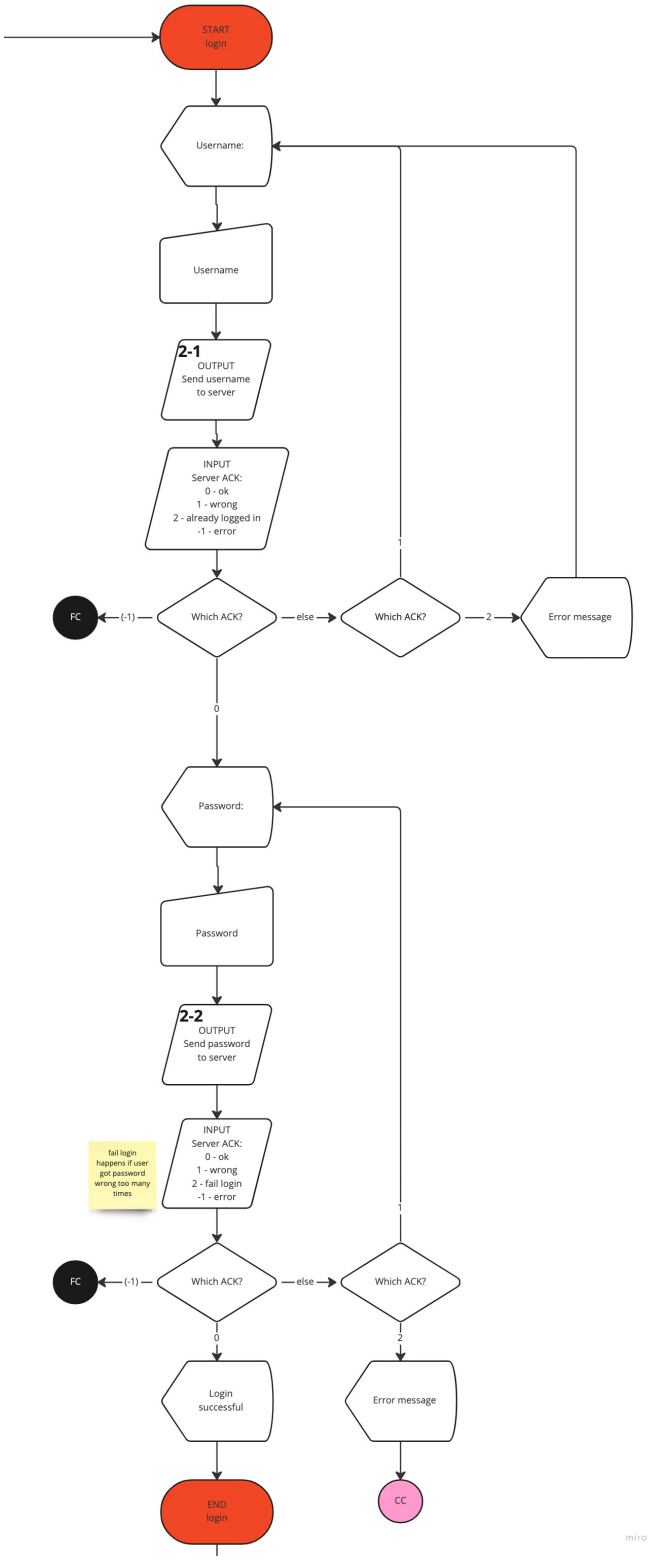

Figure 1.4: Login.

1.0.4 Menu centrale

Sia registrazione che accesso convergono in questa fase (figura 1.5), nella quale client mostra all'utente le azioni possibili, quali:

- fare una partita a WORDLE;
- ricevere le statistiche del giocatore;
- guardare i messaggi di condivisione ricevuti dagli utenti;
- effettuare il log out
- chiudere la connessione e terminare il processo.

Il client si occupa di soddisfare per intero solamente l'azione di guardare i messaggi di condivisione. Infatti, il client attiva un secondo thread, il quale apre una connessione multicast e si mette in ascolto di messaggi di condivisione inviati per conto di altri client. Quindi, ogni messaggi ricevuto viene memorizzato in un database local volatile.

Il processamento della altre azioni è nella pratica interamente delegato al server. Per questo motivo, il client inoltra l'azione al server.

1.0.5 Ricezione delle statistiche

Il client riceve dal server una stringa json, la quale viene processata e le informazioni estratte (figura). Quindi vengono mostrate le statistiche richieste dall'utente. Tuttavia, se il processamento della stringa fallisce, allora viene mostrato un messaggio d'errore.

1.0.6 Visione dei messaggi

I messaggi ricevuti sono in formato json. Di conseguenza, il client itera su tutti i vari messaggi memorizzati, processandoli e mostrandoli all'utente (figura 1.7).

1.0.7 Partita a WORDLE

Iterativamente, il cliente riceve una parola dall'utente e la inoltra al server (figura 1.8). Se la prola non è quella da indovinare, allora il client riceve dal servere un indizio e lo mostra all'utente. Tale indizio ha struttura analoga a quelli del gioco originale, quindi non è qui spiegato nuovamente. Il client esce dal ciclo di ricezione ed inoltro in due casi standard:

- **Vincità:** l'utente indovina la parola segreta.
- **Perdita:** l'utente ha tentato troppe volte, fallendo, di indovinare la parola segreta. Quante volte egli possa provare viene deciso dal server.

Condivisione dei risultati

Alla fine di ogni partita, viene chiesto all'utente se condividere i risultati. In caso affermativo, generazione ed invio del messaggio di condivisione vengono interamente gestiti dal server (figura 1.9).

Osservazione: In questa versione semplificata di WORDLE, il messaggio di condivisione viene ricevuto anche dal client che lo ha commissionato.

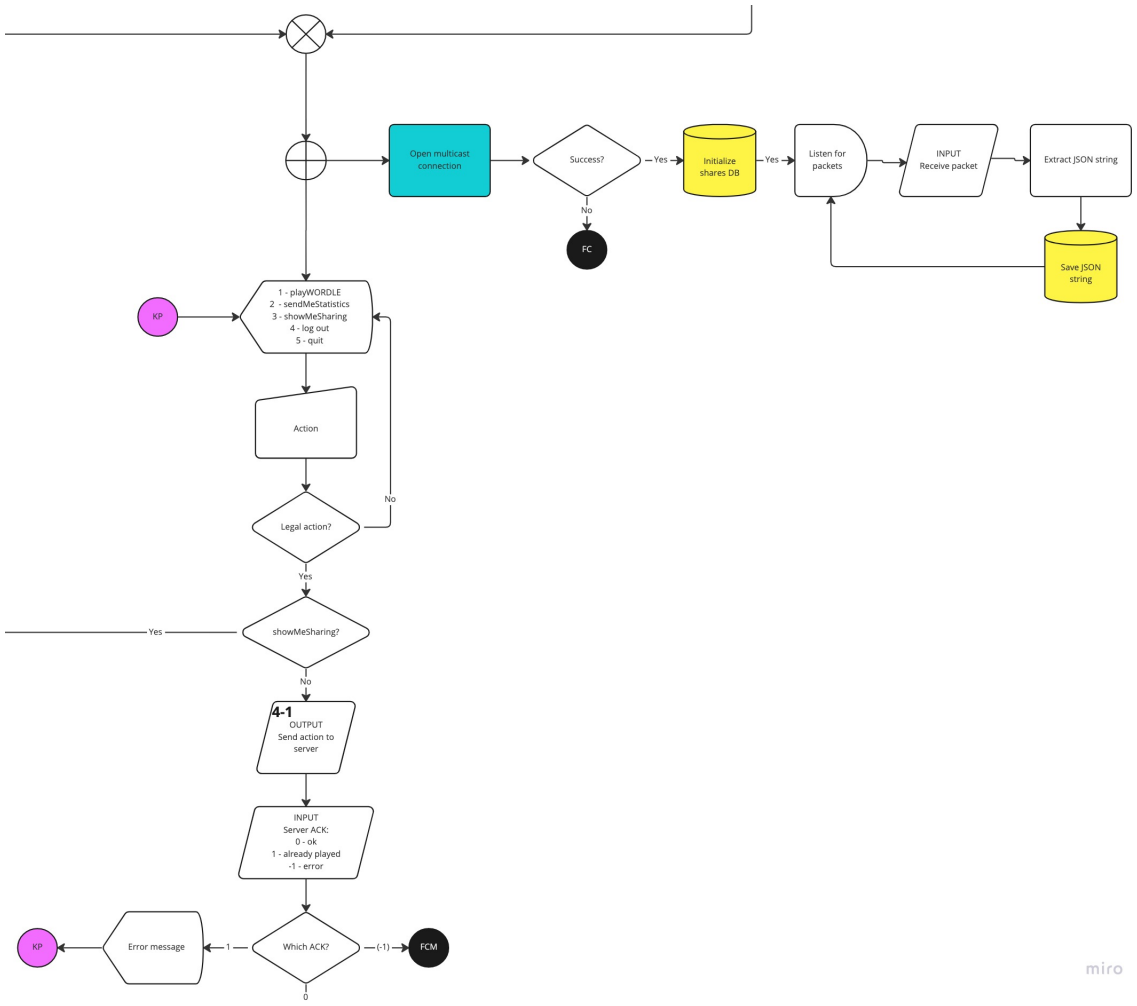

Figure 1.5: Menu principale.

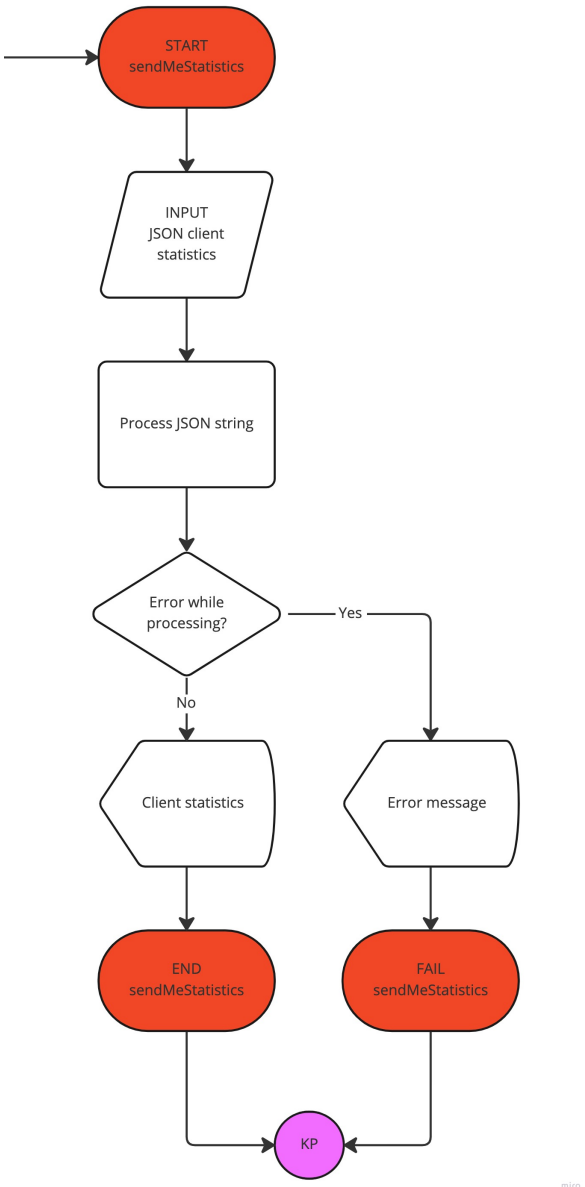

Figure 1.6: Ricezione delle statistiche.

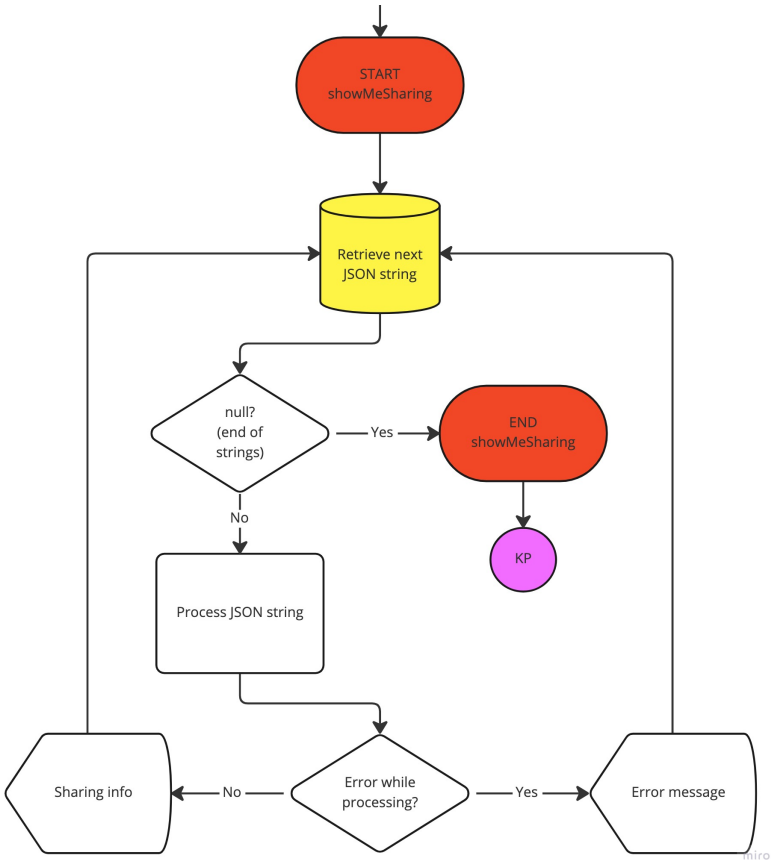

Figure 1.7: Visione dei messaggi.

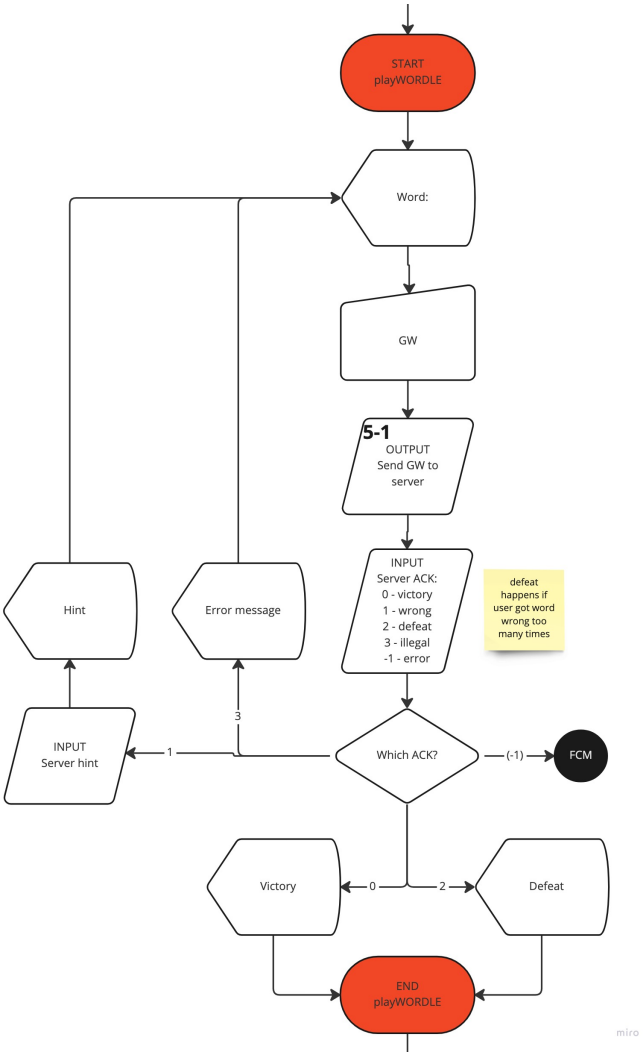

Figure 1.8: Partita a WORDLE.

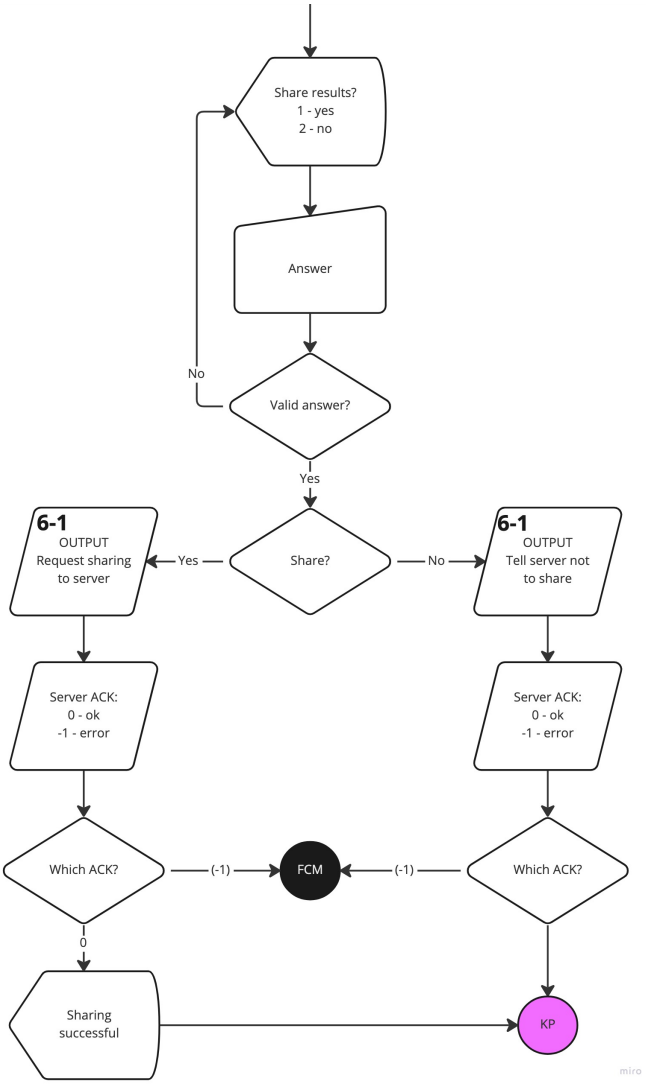

Figure 1.9: Condivisione dei risultati.

Chapter 2

Server

Il server prevede una gestione sia concorrente che parallela dei client ad esso connessi. Tuttavia, questi aspetti implementativi vengono approfonditi dopo. Adesso ci limitiamo a fornire una descrizione generale, come nel caso del client.

2.0.1 Setup

Una volta avviato, il server prevede una fase inziale iniziale di setup, in cui cerca il file delle parole ed il file json di backup del database degli utenti (figura 2.1). Se non trova il file di backup, chiede all'amministratore (utente lato server) se ciò è previsto. Se invece il file di backup è disponibile, processa ed importa il contenuto all'interno del suo database locale volatile.

2.0.2 Apertura delle connessioni

Terminata la fase iniziale di setup, il server apre una connessione multicast preliminare sulla quale invire in futuro eventuali messaggi di condivisione da parte dei client.

Inoltre, il server attiva un thread secondario per la scelta ciclica di una nuova parola segreta.

A questo punto, il server si mette in ascolto di nuove richieste di connessione da parte dei client. Quando un client si connette, il server lo inizia a servire ritornando al contempo in ascolto di nuove richieste. Come prima cosa, il server si aspetta di ricevera un azione da parte del client, al fine di capire cosa fare (figura 2.3).

2.0.3 Registrazione

La fase di registrazione (figura 2.4) è divisa in tre fasi:

1. **Username:** il server riceve uno username, quindi controlla il database degli utenti per capire se esiste di già.
2. **Password:** il server riceve una password e verifica che non sia troppo corta.
3. **Confirm passowrd:** il server riceve una seconda password e verifica che corrisponda alla precedente. Se le password coincidono, il server crea un nuovo utente con username quello fornito. Quindi la registrazione termina con successo.

Il server memorizza nel database degli utenti **non** le password, ma il loro valore hash calcolato con una funziona hash crittografica. Ciò viene fatto al fine di garantire la confidenzialità delle password, anche nel caso in cui il file di backup venga esfiltrato.

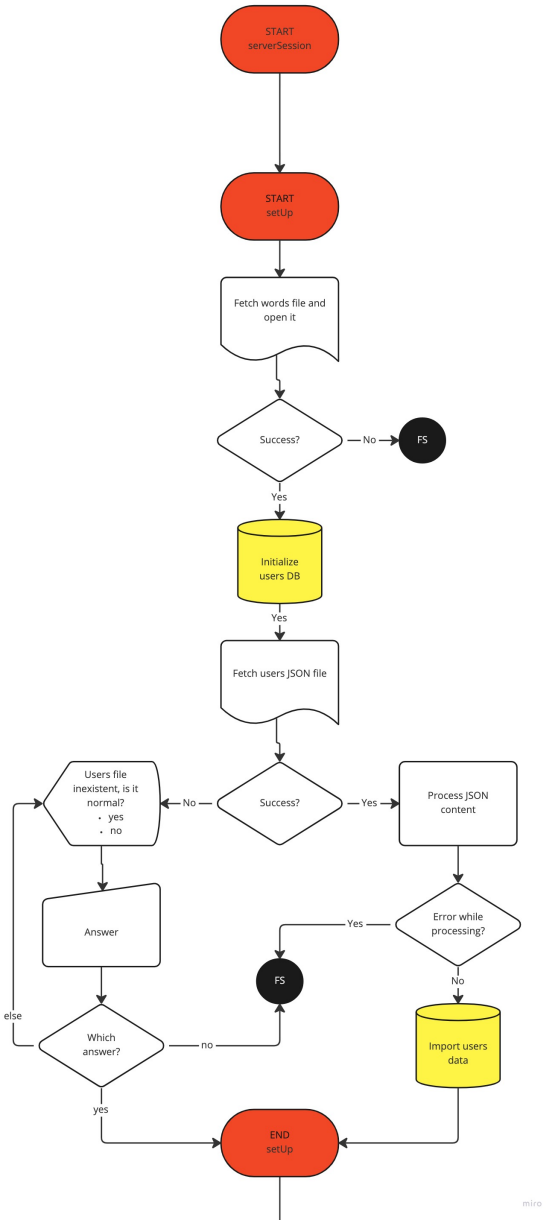

Figure 2.1: Setup iniziale.

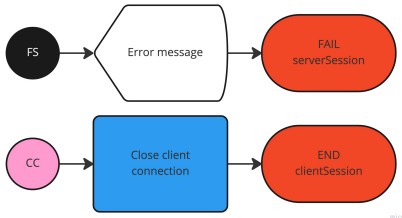

Figure 2.2: Sequenze di terminazione.

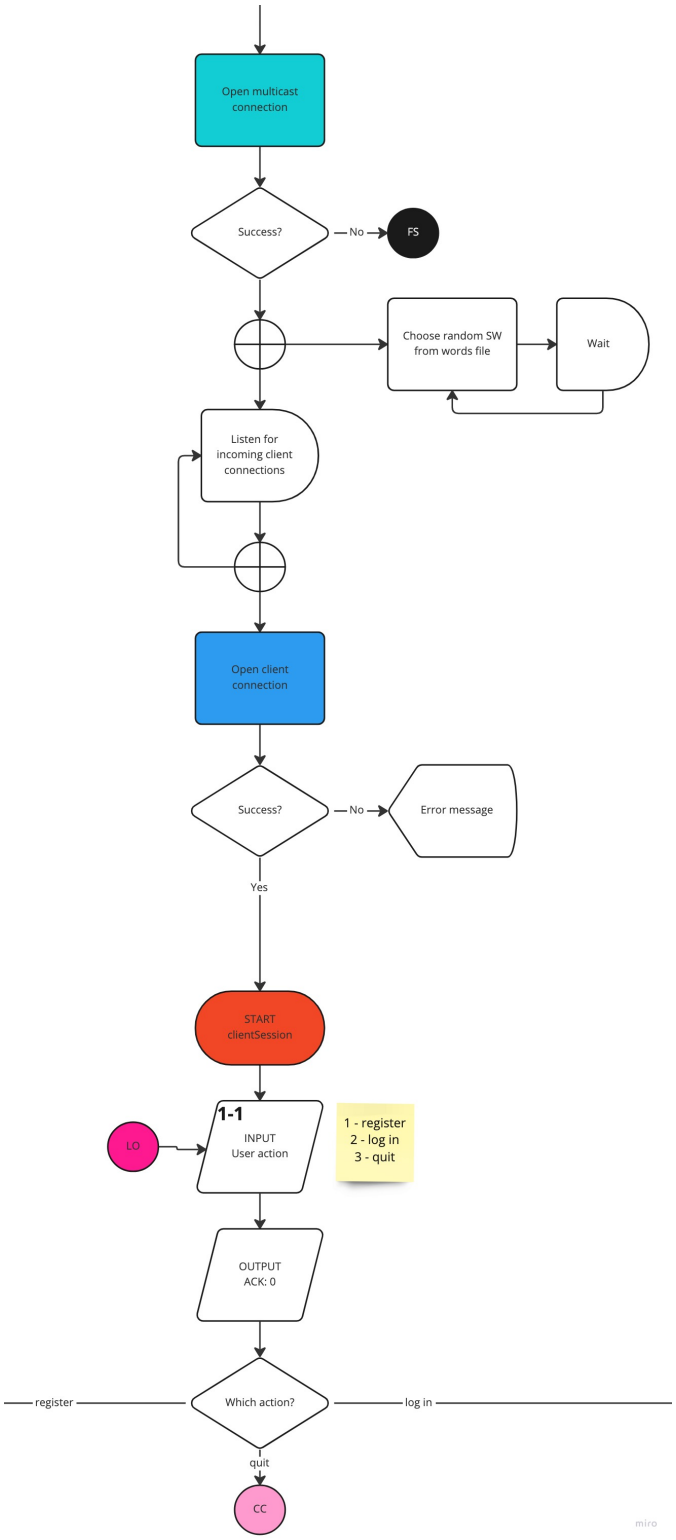

Figure 2.3: Apertura delle connessioni.

Osservazione: Un livello di sicurezza maggiore si potrebbe ottenere concatenando alle password di cui si calcola l'hash una sequenza pseudocasuale, al fine di ottenere hash diversi per utenti con uguali password.

2.0.4 Login

La fase di login (figura 2.5) è divisa in due fasi:

1. **Username:** il server riceve uno username, quindi controlla il database degli utenti per verificare che esista. In caso affermativo, controlla anche che l'utente associato allo username non sia già loggato.
2. **Password:** il server riceve una password, quindi ne calcola l'hash per verificare che coincida con quello associato all'utente specificato. In caso affermativo, l'utente viene loggato.

2.0.5 Menu centrale

In questa fase (figura 2.6), il server riceve un azione dal client per capire cosa fare. Se l'utente vuole effettuare il logout o chiudere la connessione, il server slogga l'utente, quindi fa quanto richiesto.

Se il client vuole giocare una partita a WORDLE, il server verifica che egli non abbia già giocato la parola segreta corrente, informazione mantenuta all'interno del database degli utenti.

2.0.6 Invio delle statistiche

Se il client ha richiesto di ricevere le proprie statistiche, allora il server recupera tali informazioni dal database degli utenti, crea una stringa json che le racchiude e quindi la invia al client (figura 2.7).

2.0.7 Partita a WORDLE

Durante la partita, il server ciclicamente una parola dal client, che controlla se corrisponde alla parola segreta corrente. Inoltre, il server costruisce a prescindere un indizio, in base alle informazioni ottenute dal confronto con la parola segreta. Quando l'utente vince o perde, il server aggiorna le statistiche relative ad esso (figura 2.8).

Condivisione dei risultati

Sia che il client vinca o che perda, il server riceve una decisione sul condividere o meno i risultati. Nel caso il client voglia condividere, il server costruisce un messaggio json contenente i risultati della partita ed i vari indizzi generati; quindi invia il messaggio a tutti i clienti ad esso connessi (figura 2.9).

2.0.8 Chiusura del server

Se il server riceve un segnale di shutdown da parte dell'amministratore, avvia un graduale shutdown del sistema (figura 2.10) diviso nelle seguenti fasi:

1. Smette di accettare nuove richieste di connessione e si mette un certo periodo di tempo in attesa che i client connessi si scolleghino autonomamente.
2. Invia dei messaggi di errore a tutti i client connessi, al fine di costringerli a sconnettersi. Quindi mette un alro po' in attesa, per assicurarsi che effettivamente tutti i client si siano sconnessi.

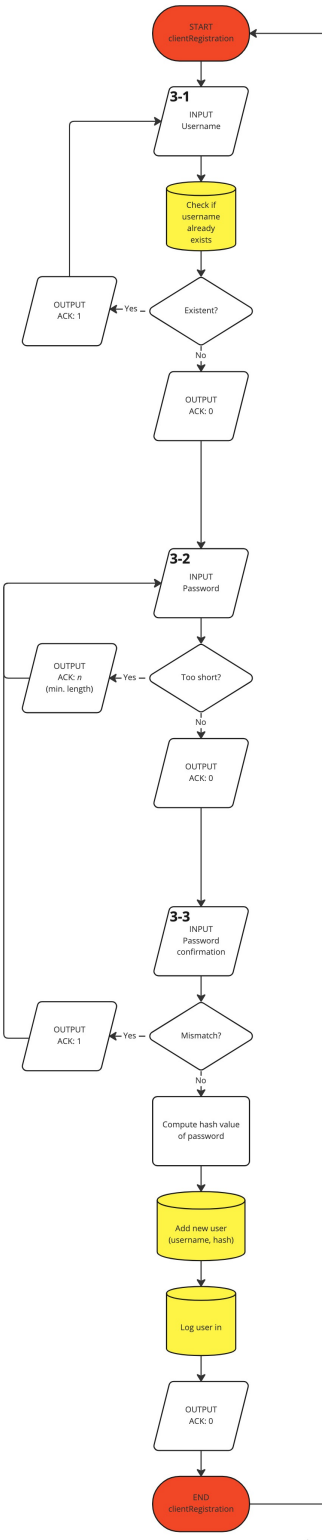

Figure 2.4: Registrazione.

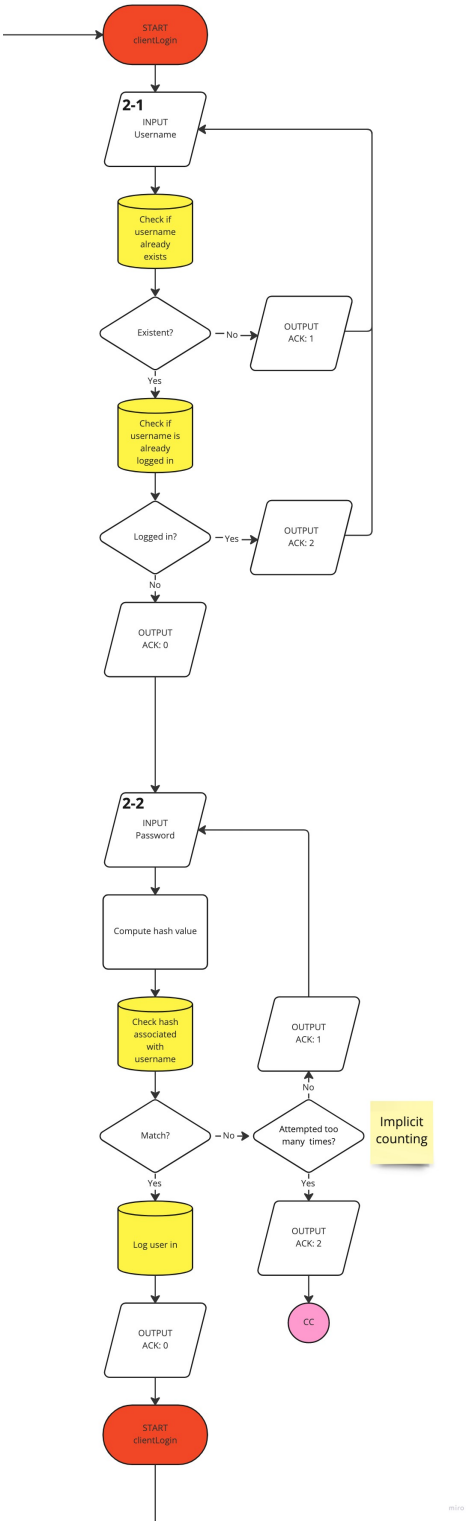

Figure 2.5: Login.

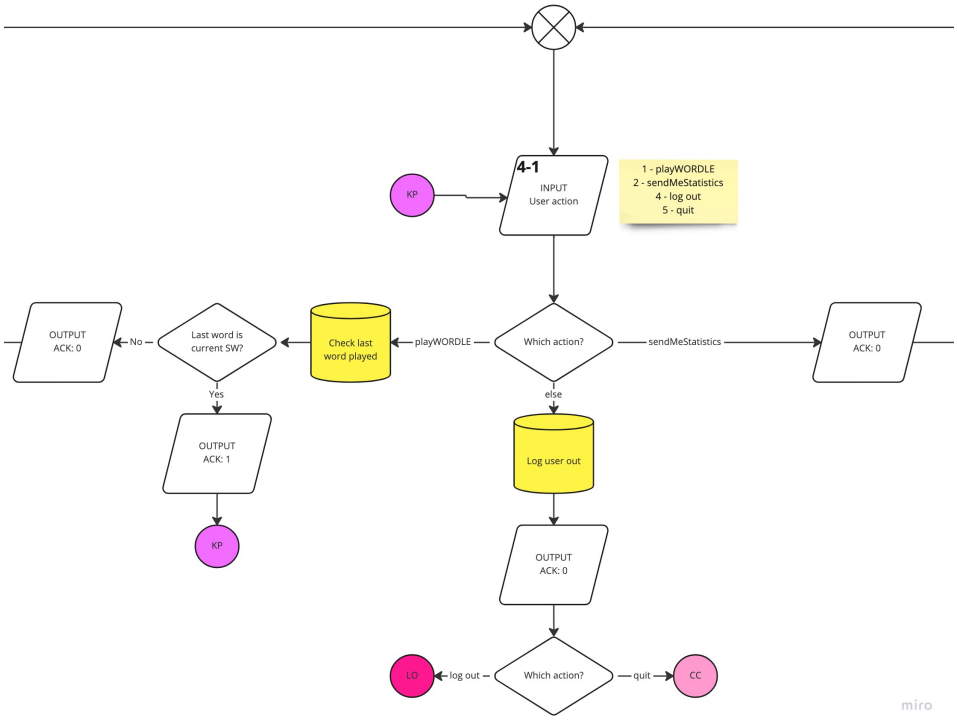

Figure 2.6: Menu centrale.

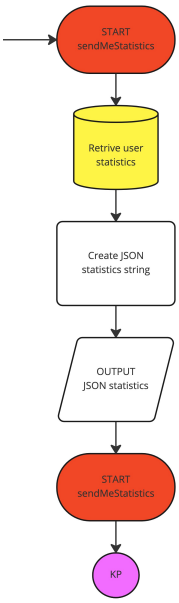

Figure 2.7: Invio delle statistiche.

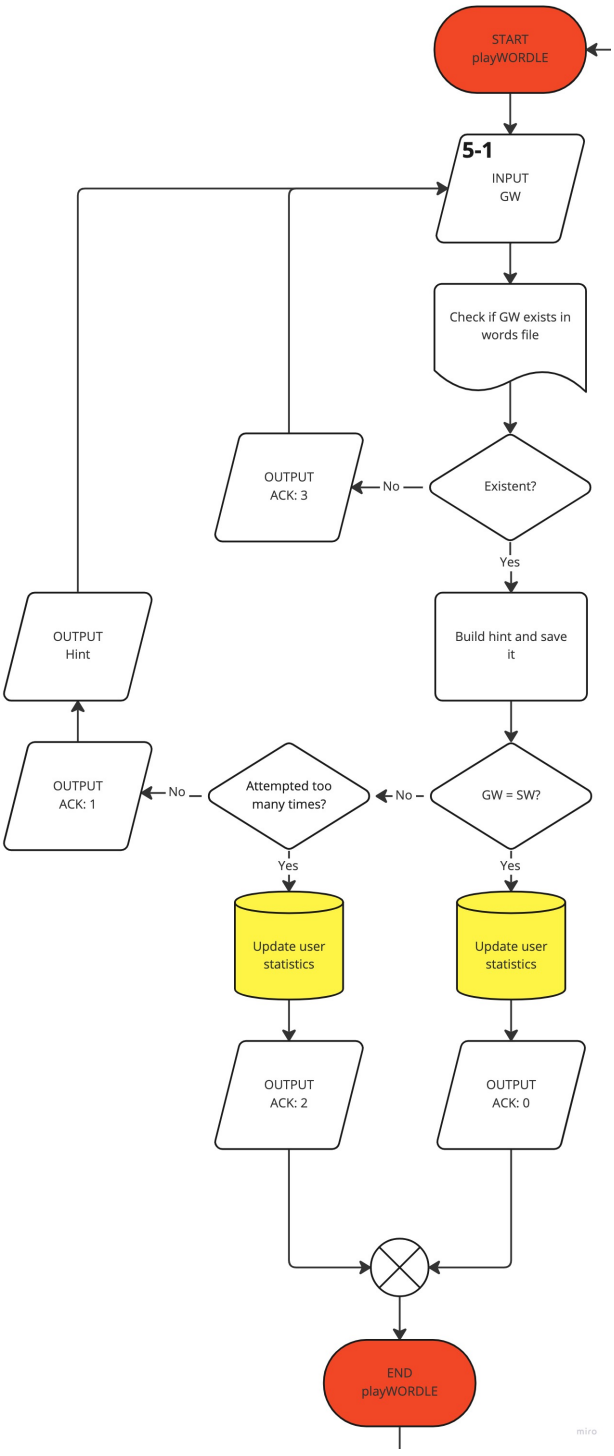

Figure 2.8: Partita a WORDLE.

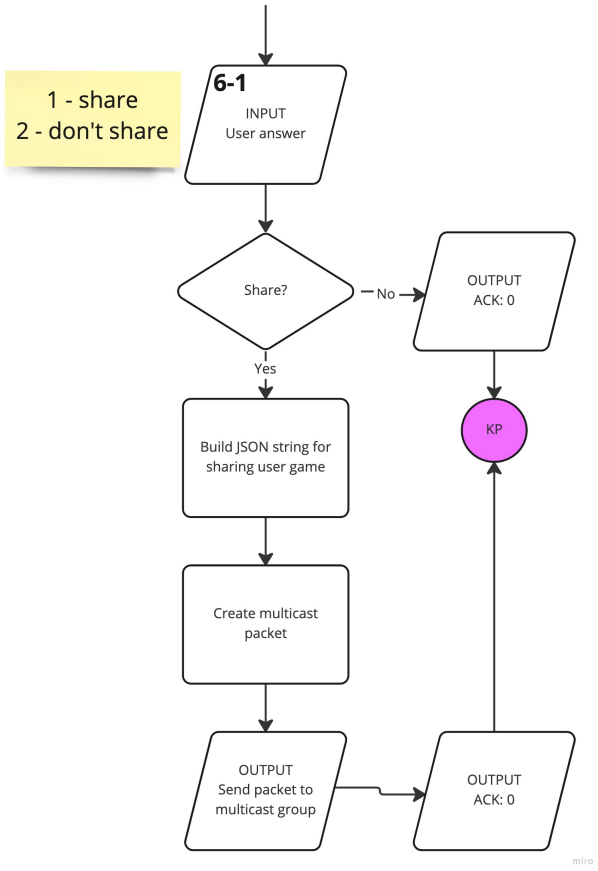

Figure 2.9: Condivisione dei risultati.

3. A questo punto chiude tutte le connessioni ancora aperte e slogga tutti gli utenti registrati.
4. Esporta un file json contenente le informazioni sul database degli utenti.
5. Termina.

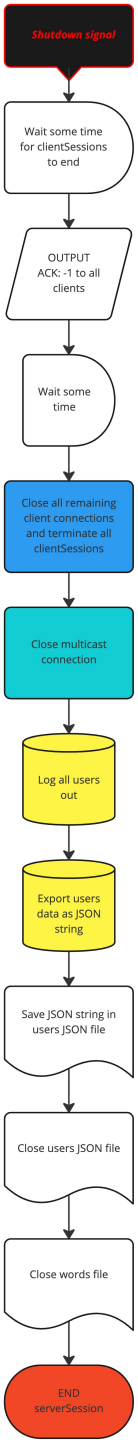

Figure 2.10: Shutdown.

Part II

Dettagli implementativi

Chapter 3

Comunicazione

Client e server comunicano per mezzo di socket channels, quindi mediante NIO. Questa modalità di comunicazione è preferibile rispetto alla JAVA I/O al fine ottenere maggiore scalabilità sul numero di client efficientemente gestibili dal server (more on this later).

Per agevolare la comunicazione tra le due entità, sono state definite due classi di servizio, `ChannelReader` e `ChannelWriter`, che definiscono metodi per leggere da e scrivere su socket channels, rispettivamente.

3.1 Regole

Come già detto, il client si aspetta di ricevere un feedback (**ack**) dal server dopo ogni dato inviato ad esso. La convenzione seguita qui è la seguente:

- `ACK = 0`: tutto nella norma.
- `ACK > 0`: errore dovuto ai dati inviati dal client.
- `ACK = -1`: errore fatale dovuto ad un problema del server o della comunicazione tra le due entità.

Nel caso di errore fatale, il client si disconnette e termina, mostrando un messaggio di errore appropriato all'utente.

Chapter 4

Server

4.1 Classi

Il server è composto dalle seguenti classi:

- `ServerMain`: classe principale del server, il quale si occupa di importare tutti i parametri di configurazione, effettuare setup iniziale, inizializzare il database degi utenti ed eventualmente importarlo da un file di backup, stare in ascolto di nuove richieste di connessione.
- `ClientHandler`: classe istanziata da ServerMain che si occupa di servire effettivamente i client connessi. Tutte le funzionalità messe a disposizione del client vengono implementate qui dentro. Tale classe implementa l'interfaccia `Runnable`, infatti ogni sua istanza viene passata come task ad un threadpool in ServerMain.
- `SWHandler`: classe istanziata da ServerMain che si occupa di generare ciclicamente una nuova secret word. L'attesa tra due generazioni consecutive è fornita come parametro di configurazione.

Inoltre, le ultime due classi si appoggiano sulla classe di servizio `WordsFileHandler`, la quale offre metodi statici per restituire una parola casuale appartenente al file di parole e dire se un certa parola datagli in input appartiene a tale file.

4.2 Gestione dei client

È di immediata comprensione il fatto che ogni richiesta da parte di un client è computazionalmente facile da soddisfare. Inoltre, le richieste da parte dei client connessi arrivano con una frequenza *umana*. Infatti, ogni richiesta viene generata in conseguenza di un input da parte di un utente lato client che impiega solitamente centinaia di millisecondi (se non secondi) per fornirlo.

Tutto ciò porta a concludere che l'opzione di allocare un thread per client è altamente sconsigliata, dato che ogni thread rimarrebbe *idle* per la maggior parte del tempo.

Altra opzione è quella di utilizzare un selector, quindi NIO, per servire le singole richieste dei client. Ciò implica che ad ogni client connesso debba essere associato uno **stato** univoco che mantenga tutte le informazioni relative alla sessione. Pur se in assoluto preferibile questa soluzione rispetto a quella precedente, si ha comunque scalabilità limitata, dato che le richieste vengono soddisfatte sequenzialmente ed un numero elevato di client potrebbe introdurre sensibili ritardi.

4.2.1 Approccio usato

In questo progetto, è stata usata una **soluzione ibrida** rispetto alle due sopra menzionate. In particolare, essa prevede la seguente dinamica di funzionamento:

- Ogni richiesta da parte dei client viene *ascoltata* da un selector in ServerMain.
- Se la richiesta è di connessione, allora viene gestita direttamente in ServerMain. Questo in quanto si suppone che le richieste di connessione siano rare rispetto a quelle di operazione da parte degli utenti già connessi.
- Se la richiesta è di operazione (un utente già conesso interagisce col server) allora viene creato un nuovo task (`ClientHandler`) di gestione della richiesta, il quale è quindi sottomesso ad un cached thread pool.

Osservazione: È stato scelto di usare un cached thread pool proprio per il fatto che ogni richiesta viene soddisfatta in tempi molto brevi.

Vantaggi

I vantaggi di applicare questa soluzione sono princiapalmente due:

- Siccome i tempi di soddisfacimenti di una richiesta sono molto bassi, è difficile che il thread pool attivi molti thread in contemporanea, impattando così poco sulle risorse di sistema.
- L'utilizzo di un thread pool, permette di soddisfare più richieste (da client diversi) parallelamente, rendendo il il servizio maggiormente scalabile.

4.3 Strutture dati

Il database locale degli utenti è implementato medianto una **HashMap** con coppie $\langle Username, User \rangle$. **User** è un oggetto contenente tutte le informazioni relative all'utente, quali:

- nome;
- hash della password;
- stato di log;
- ultima secret word giocata (non necessariamente indovinata);
- insieme di statistiche contenute in un oggetto $UserStats$.

Ogni oggetto **UserStats** contiene tutte le statistiche di un utente, quali:

- numero di partite giocate;
- numero di vittorie;
- numero di sconfitte;
- percentuale di vittorie;
- last streak;
- maximum streak;

- guess distribution.

Infine, `ClientHandler` gestisce a livello di classe un **HashSet** all'interno del quale memorizza lo username al momento scelto da un client in fase di registrazione. Le operazioni su tale struttura dati vengkno effettuate all'interno dun un blocco sincronizzato. Lo scopo di tale struttura è quello di evitare che due client possano possano in contemporanea scegliere lo stesso username in fase di registrazione. Quindi l'esistenza di uno username, in fase di registrazione, viene controllata cercandolo sia nel database degli utenti sia in questa struttura dati.

4.4 Sincronizzazione

I blocchi sincronizzati appena menzionati sono le uniche strutture di sincronizzazione adoperate. Infatti, il databse degli utenti non necessita di essere una struttura concorrente. Si noti infatti come l'unico momento in cui si modifica il database è proprio la fase di registrazione.

Chapter 5

Client

5.1 Classi

Il client è composto da due classi:

- `ClientMain:` corpo del client, il quale si occupa di implementare tutte le comunicazioni con il server e mantenere i messaggi di condivisione.
- `MCHandler:` classe secondaria che ha il compito di stare in ascolto sul gruppo multicast e ricevere messaggi di condivisione inviati dal server per conto di altri client. Essa è sottoclasse di `Thread`, quindi la sua istanza opera su un thread secondario.

Si comprende quindi la natura sequenziale di tale entità.

5.2 Strutture dati

L'unica struttura dati utilizzata nel client è l'`ArrayList` di stringhe, usata per implementare il database dei messaggi di condivisione ricevuti. Tale database viene acceduto solo quando l'utente vuole visionare i messaggi di condivisione ricevuti. Tali messaggi vengono mostrati per ordine di ricezione, quindi l'ArryList si presta particolarmente bene alla realizzazione di questo database.

Part III

Istruzioni per l'uso

NOTA: Tutte le istruzioni sotto riportate sono da considerarsi valide in ambienti Unix (Linux, MacOS, etc.). Non si garantisce la loro validita in ambienti Windows.

Materiale

All'interno del file ZIP relativo progetto, vi è una directory `Progetto`, la quale contiene:

- Tutte le classi `*.java` del caso.
- Una directory `configuration` contenente i file di configurazione di server e client.
- Un file `words.txt` contenente le parole per il gioco.
- Un file `gson-2.10.jar` contenente la libreria gson.

Compilazione

Per compilare il progetto muoversi, da linea di comando, nella directory `Progetto`, quindi digitare:

```
javac -cp .:./gson-2.10.jar *.java
```

Esecuzione

Per eseguire il server, digitare:

```
java -cp .:./gson-2.10.jar ServerMain
```

Mentre per eseguire il client, digitare:

```
java -cp .:./gson-2.10.jar ClientMain
```

Pulizia

Per rimuovere tutti file generati dalla compilazione, digitare:

```
rm *.class
```