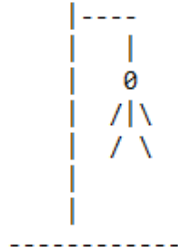


# Hangman Online



Filippo Venturini  
`filippo.venturini8@studio.unibo.it`

January 2023

L'obiettivo di questo progetto è la realizzazione di una versione multi-player online del gioco Hangman, meglio conosciuto come "L'Impiccato". Ad una partita partecipano due giocatori, il primo deve decidere una parola che verrà codificata utilizzando il carattere '+' per le vocali e '-' per le consonanti, il secondo ha l'obiettivo di indovinare la parola in un numero finito di tentativi, proponendo una lettera per turno. Il gioco termina con la vittoria del primo giocatore se i tentativi sono terminati o del secondo giocatore se è stata indovinata la parola.

# 1 Goal/Requirements

Di seguito sono riportate una serie di domande per raffinare le varie sfumature del progetto a livello di requisiti, con un orientamento verso le interazioni dell'utente con il sistema.

## 1.1 Questions/Answers

- *Come si accede al sistema?*
  - Per accedere al sistema è sufficiente inserire un nickname (che non risulti già in uso) che identifica univocamente il giocatore per l'intera sessione di gioco.
- *Una volta connessi cosa è possibile fare?*
  - Appena effettuata la connessione è possibile creare una lobby di gioco oppure unirsi ad una lobby già esistente (A patto che non sia già al completo).
- *Quanti giocatori possono partecipare contemporaneamente?*
  - Il sistema deve essere in grado di gestire molteplici partite effettuate da coppie di giocatori, ogni lobby permette l'inizio di una partita quando sono presenti esattamente due giocatori.
- *Che ruoli hanno gli utenti all'interno di una partita?*
  - Una volta iniziata la partita un utente viene identificato come *Chooser*, ovvero colui che sceglie la parola, mentre l'altro viene identificato come *Guesser* e ha l'obiettivo di indovinare la parola che è stata scelta con 5 possibilità di errore.
- *Com'è organizzata una partita?*
  - Una partita è articolata in al più tre round, vince il giocatore che riesce ad ottenere due vittorie. Durante il primo round vengono assegnati i ruoli di *Chooser* e *Guesser* casualmente, mentre per il secondo round i ruoli vengono invertiti. In caso si verificasse un pareggio, per il terzo e ultimo round i ruoli vengono nuovamente assegnati casualmente.

## 1.2 Scenarios

Considerando un tipico scenario di utilizzo di Hangman Online, un utente si connette al sistema per poter giocare all'impiccato con altri giocatori che sono collegati e per connettersi inserisce un nickname che lo identifica durante la sessione di gioco.

Dopo essersi connesso può decidere se creare una lobby di gioco e attendere che un altro giocatore si connetta, oppure partecipare a una lobby già esistente.

In entrambi i casi, quando la lobby contiene esattamente due giocatori, inizia la partita. Una partita è articolata in al più tre round, in ogni round le azioni richieste variano in base al ruolo che viene assegnato all'utente dal sistema:

- *Chooser*: L'utente sceglie una parola da far indovinare all'altro giocatore, essa viene codificata con i caratteri '+' e '-' (Come previsto dal gioco dell'impiccato). Successivamente l'utente attende e visualizza i tentativi effettuati dall'altro giocatore e vince se ha scelto una parola sufficientemente complicata per non essere indovinata con cinque tentativi.
- *Guesser*: L'utente attende che venga scelta la parola da indovinare e ne riceve la versione codificata (Con i caratteri '+' e '-'). Ha cinque possibilità di errore e ad ogni tentativo può inserire una lettera, che se contenuta nella parola verrà sostituita al carattere corrispondente, altrimenti verrà mostrata una parte della figura dell'impiccato (Che corrisponde quindi ad un errore). Inoltre è possibile durante un tentativo, provare a indovinare la parola per intero (In caso la parola sia errata viene consumato un tentativo). L'obiettivo del Guesser è indovinare la parola commettendo meno di cinque errori (Evitando quindi che la figura dell'impiccato venga mostrata completamente).

Al termine di ogni round i ruoli dei giocatori vengono invertiti, vince la partita chi riesce a vincere due round. Di conseguenza, in caso di pareggio viene svolto un terzo round per determinare il vincitore.

Terminata la partita l'utente può nuovamente accedere al menu principale e giocare con altri giocatori, oppure uscire dal gioco.

### 1.3 Self-assessment policy

Il progetto sarà dotato di una serie di test automatici, realizzati tramite il framework JUnit. I test saranno mirati alla verifica di tutte le funzionalità descritte durante l'analisi dei requisiti, a partire da semplici test di connettività fino alla simulazione di una vera e propria partita.

## 2 Requirements Analysis

Analizzando i requisiti descritti nella sezione precedente è possibile identificare una serie di funzionalità chiave che il sistema dovrà possedere per rispettare le specifiche. Le funzionalità fanno riferimento al diagramma dei Casi d'Uso per la connessione al sistema (fig.1) e a quello riferito alle partite (fig.2).

### Connectivity functionalities

- Registrazione
- Creazione di una lobby
- Connessione ad una lobby

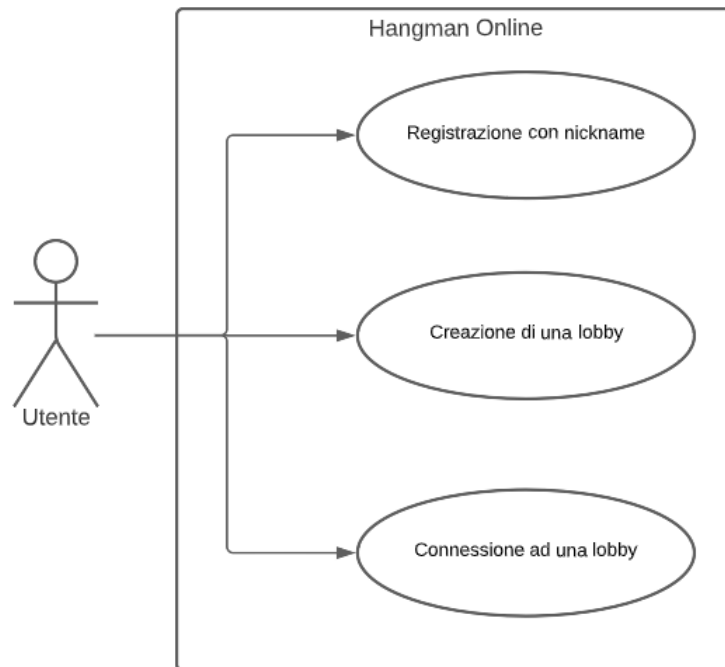


Figure 1: Connection Use Case Diagram.

## Game functionalities

- Scelta di una parola
- Tentativo di indovinare una lettera
- Tentativo di indovinare la parola

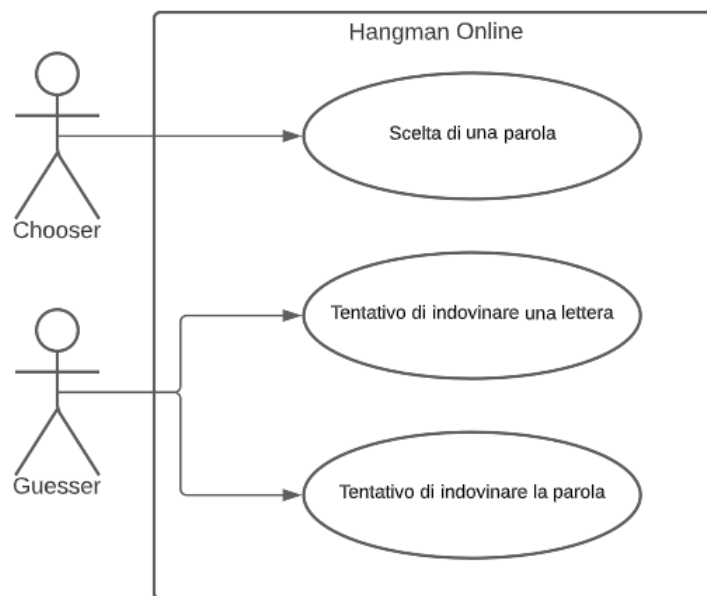


Figure 2: Game Use Case Diagram.

Alla luce dell'analisi effettuata è stato deciso di implementare il sistema utilizzando un'architettura client-server, implementando il server come un Web Service, dotato di ReST API e che utilizza quindi HTTP come protocollo di comunicazione.

### 3 Design

Successivamente all'analisi dei requisiti si procede ad illustrare il design adottato per la realizzazione di Hangman Online.

Essendo pensato come un gioco multiplayer online è stato deciso di mantenere un alto livello per quanto riguarda i protocolli e le tecnologie di comunicazione tra i vari componenti del sistema, ciò rende l'applicativo più facilmente collocabile in un contesto reale.

#### 3.1 Structure

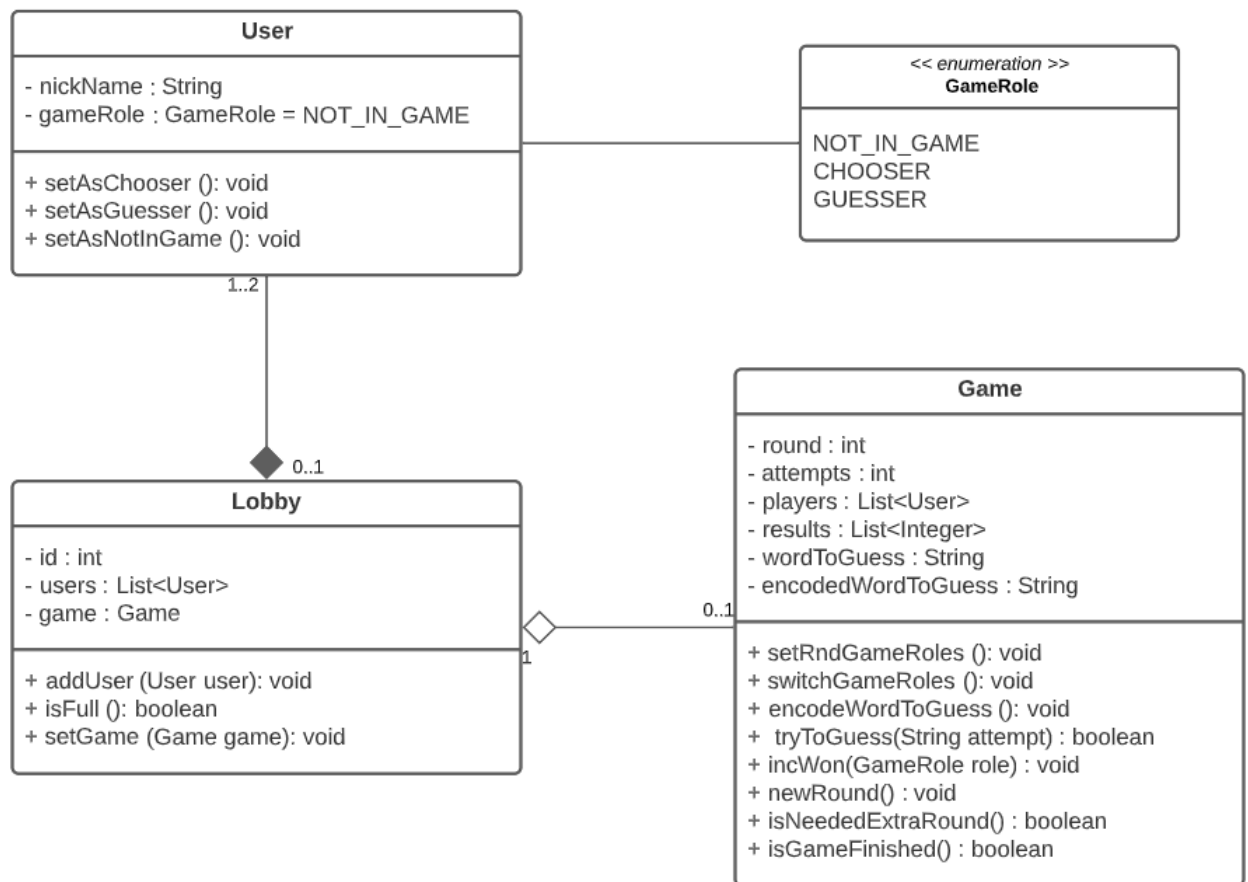


Figure 3: Class Diagram.

In figura 3 è viene mostrata la struttura dei componenti del Model tramite un diagramma delle classi UML, nel quale per ogni entità sono riportati solamente i campi e i metodi più significativi.

Considerando l'entità **User**, essa incapsula il concetto di utente che viene identificato da un `nickname` e il suo attributo `gameRole` ne identifica il ruolo all'interno del sistema. L'attributo `gameRole` fa riferimento ad un'enumerazione che individua come possibili ruoli: `NOT_IN_GAME` per gli utenti che sono semplicemente connessi al sistema ma non stanno giocando, `CHOOSER` per un utente che ha il compito di scegliere la parola e `GUESSER` per un utente che ha l'obiettivo di indovinarla. L'entità `User` possiede inoltre dei metodi per poter settare correttamente il suo ruolo all'interno del sistema.

Per quanto riguarda l'entità **Lobby** essa (come si evince dall'analisi dei requisiti), contiene almeno un utente (colui che la crea) e viene considerata piena quando è popolata da esattamente due utenti. La lobby possiede un `id` che la identifica univocamente, la lista di utenti ad essa connessi e il riferimento all'oggetto `game` che le corrisponde. Possiede inoltre il metodo `addUser(User user)` per effettuare l'aggiunta degli utenti, il metodo `isFull()` per verificare se è piena e il metodo `setGame(Game game)` per settare la partita.

Infine viene presentata l'entità più complessa, nonché centrale nella struttura del Model, ovvero l'entità **Game**. Essa possiede tutti gli elementi che sono stati descritti nella fase di analisi dei requisiti. Contiene infatti al suo interno: il riferimento ai players, i risultati, il round corrente, il numero di tentativi rimasti, la parola da indovinare e la sua codifica. L'entità `Game` possiede inoltre tutti i metodi necessari per la progressione del gioco:

- `setRndGameRoles()`: viene utilizzato per il setting casuale dei ruoli
- `switchGameRoles()`: scambia i ruoli dei due giocatori in partita
- `encodeWordToGuess()`: codifica la parola da indovinare
- `tryToGuess(String attempts)`: verifica se il tentativo è andato a buon fine o meno
- `incWon(GameRole role)`: incrementa le vittorie del giocatore con il ruolo indicato
- `newRound()`: predispose gli attributi per l'inizio di un nuovo round
- `isNeededExtraRound()`: verifica se è necessario uno spareggio
- `isGameFinished()`: verifica se la partita è conclusa

### 3.2 Behaviour

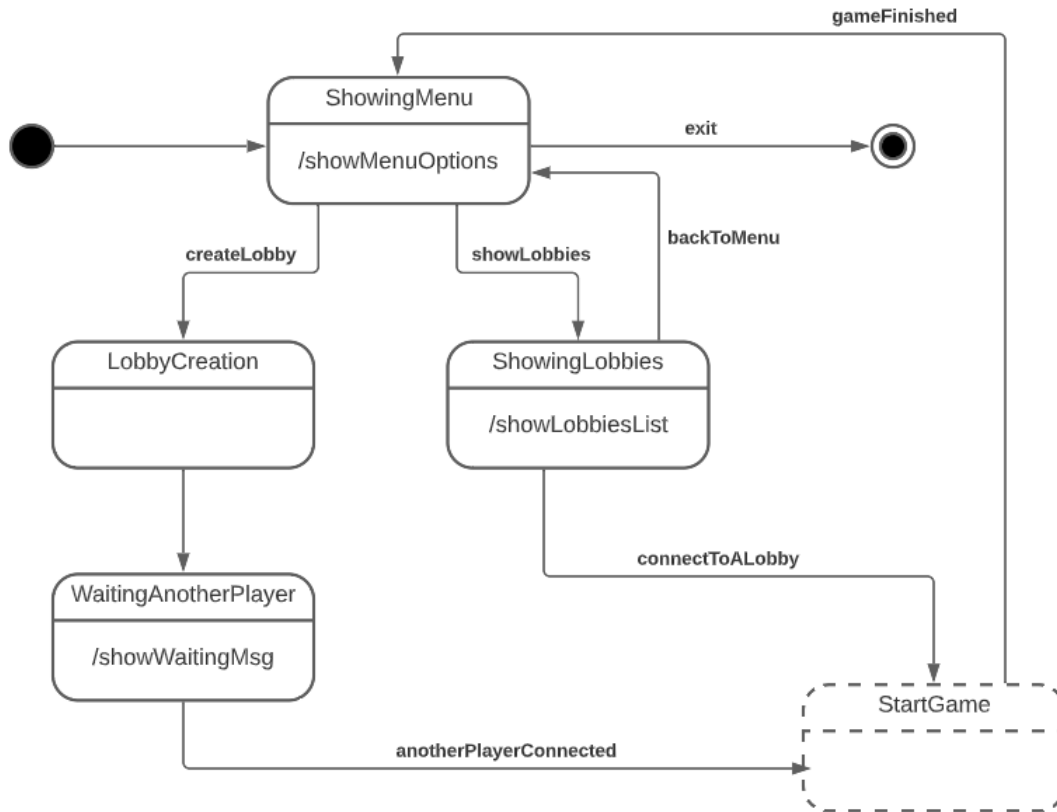


Figure 4: Lobby State Diagram.

In figura 4 viene mostrato un diagramma degli stati che esemplifica il comportamento dell'entità Lobby in seguito all'interazione con l'utente. Inizialmente viene mostrato il menu che permette all'utente di scegliere l'operazione desiderata e come descritto nei requisiti, è possibile creare una nuova lobby oppure connettersi ad una lobby già esistente.

Nel caso di una creazione, l'utente rimane in attesa della connessione di un altro giocatore alla propria lobby, quando ciò avviene si inizia la partita.

Nel caso invece l'utente volesse collegarsi ad una lobby già esistente, vengono prima mostrate tutte le lobbies esistenti e una volta che una di esse viene selezionata la partita ha inizio.

In entrambe le situazioni una volta terminata la partita viene nuovamente mostrato il menu principale, da cui è sempre possibile uscire.



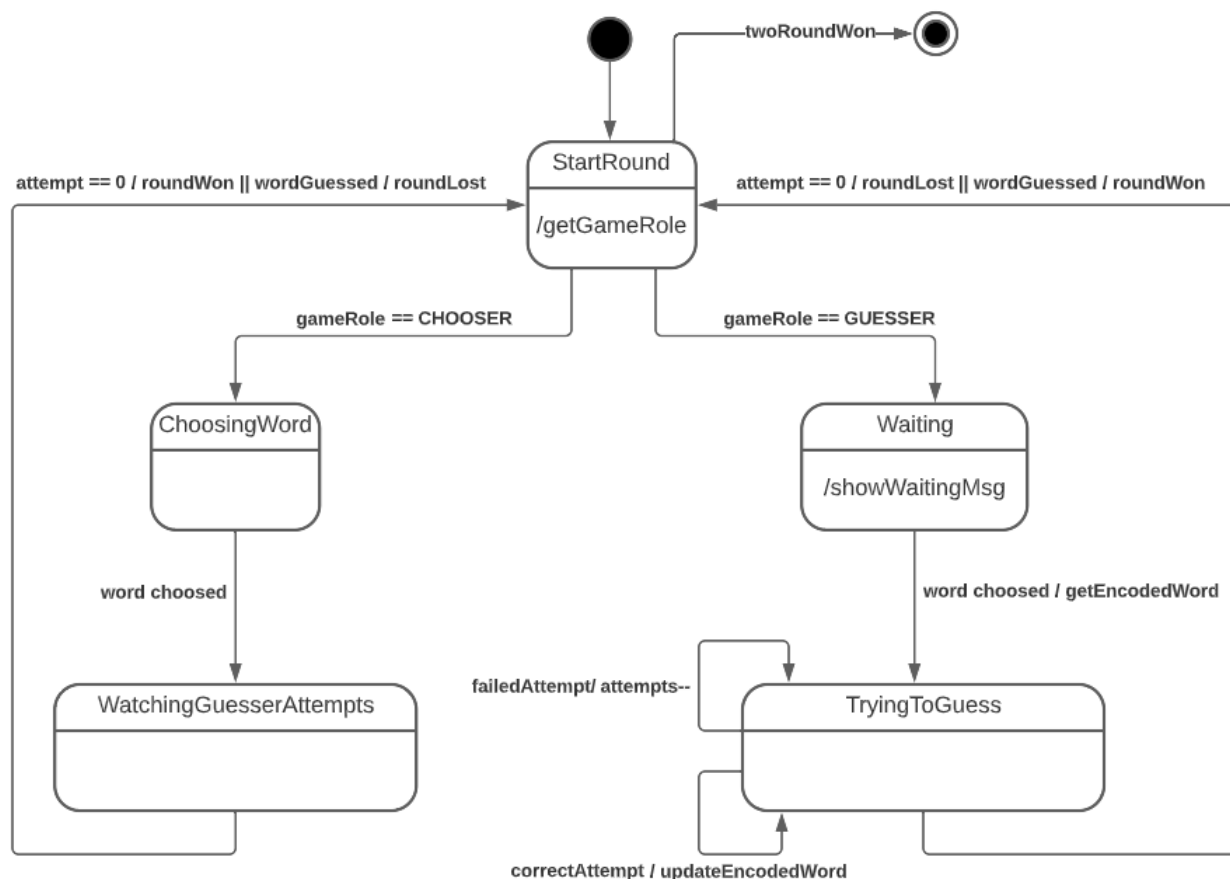


Figure 5: Game State Diagram.

In figura 5 viene mostrato il diagramma degli stati che illustra il comportamento dell'entità Game, durante l'evoluzione di una partita. Essendo la partita strutturata in round, le dinamiche dipendono dal ruolo assunto dal giocatore.

In caso all'utente venga assegnato il ruolo di **CHOOSER**, gli viene richiesto di scegliere una parola e poi viene collocato in uno stato di attesa in cui visualizza i tentativi dell'altro giocatore e al termine del round gli viene comunicata la vittoria o la sconfitta.

Nel caso invece in cui l'utente venga identificato come **GUESSER**, si entra in uno stato di attesa della parola da indovinare, una volta che essa viene scelta dal **CHOOSER** al **GUESSER** ne viene mostrata la codifica con i caratteri '+' e '-'. Partendo da questa codifica l'utente deve provare a indovinare: in caso di tentativo corretto viene effettuato un aggiornamento della parola codificata (mostrando il carattere indovinato) altrimenti vengono decrementati i tentativi a disposizione. Al termine dei tentativi oppure quando la parola viene indovinata, viene comunicata la vittoria o la sconfitta. Successivamente ad un round, vengono riassegnati i ruoli e si procede con le stesse dinamiche ai round successivi. Quando uno dei due utenti totalizza due round vinti la partita termina.

### 3.3 Interaction

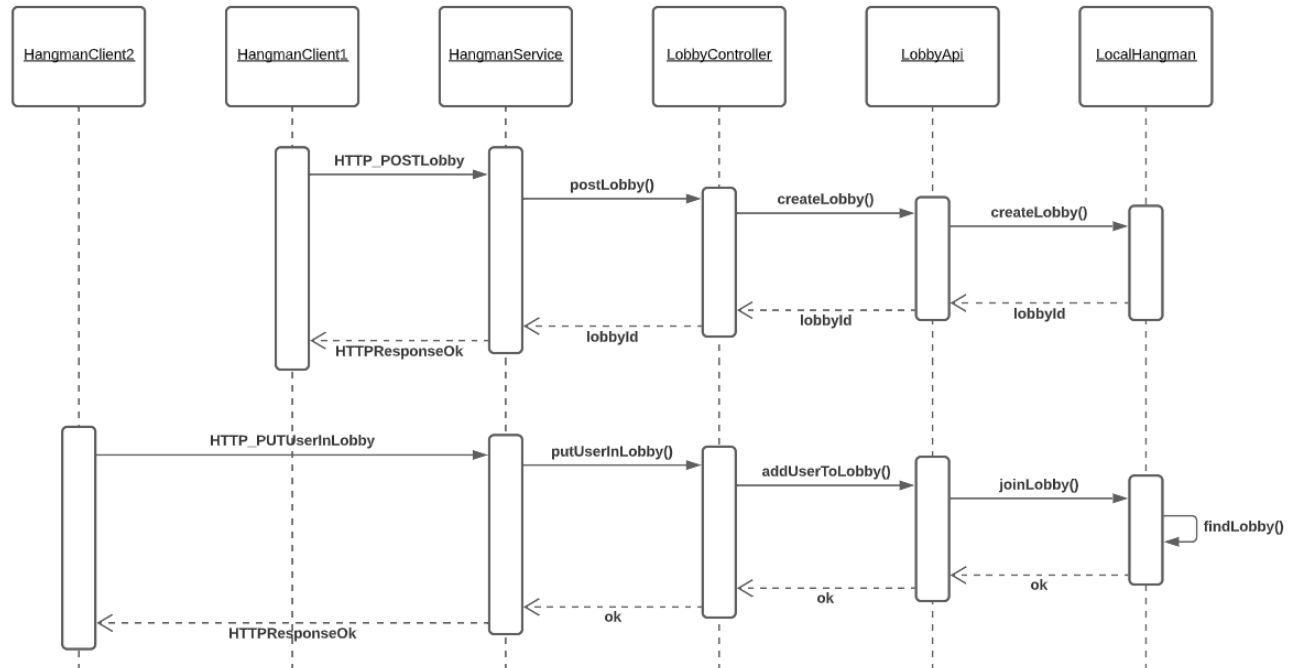


Figure 6: Lobby Sequence Diagram.

Ora è necessario analizzare la struttura delle interazioni presenti all'interno del sistema. In figura 6 viene mostrato un diagramma di sequenza che esemplifica la creazione di una lobby e la connessione ad essa.

Come ogni risorsa presente nel sistema, la Lobby lato server possiede un Controller e un insieme di Api che si occupano di rispondere alle richieste dei client.

In particolare quando il **Client1** effettua una richiesta di tipo POST al server per creare la lobby, essa viene indirizzata al **LobbyController** che richiama a sua volta la corretta API contenuta all'interno di **LobbyApi** che infine agisce sull'istanza locale di Hangman presente sul server. Le risposte alle varie richieste si propagano seguendo la stessa catena di interazioni tra i vari componenti del sistema.

Anche le entità **User** e **Game** possiedono i relativi Controller e le corrispondenti API, di conseguenza tutte le richieste effettuate dai client con le relative risposte, vengono indirizzate ai Controller di competenza e la sequenza di interazioni che si verificano è la medesima illustrata nello schema presentato.

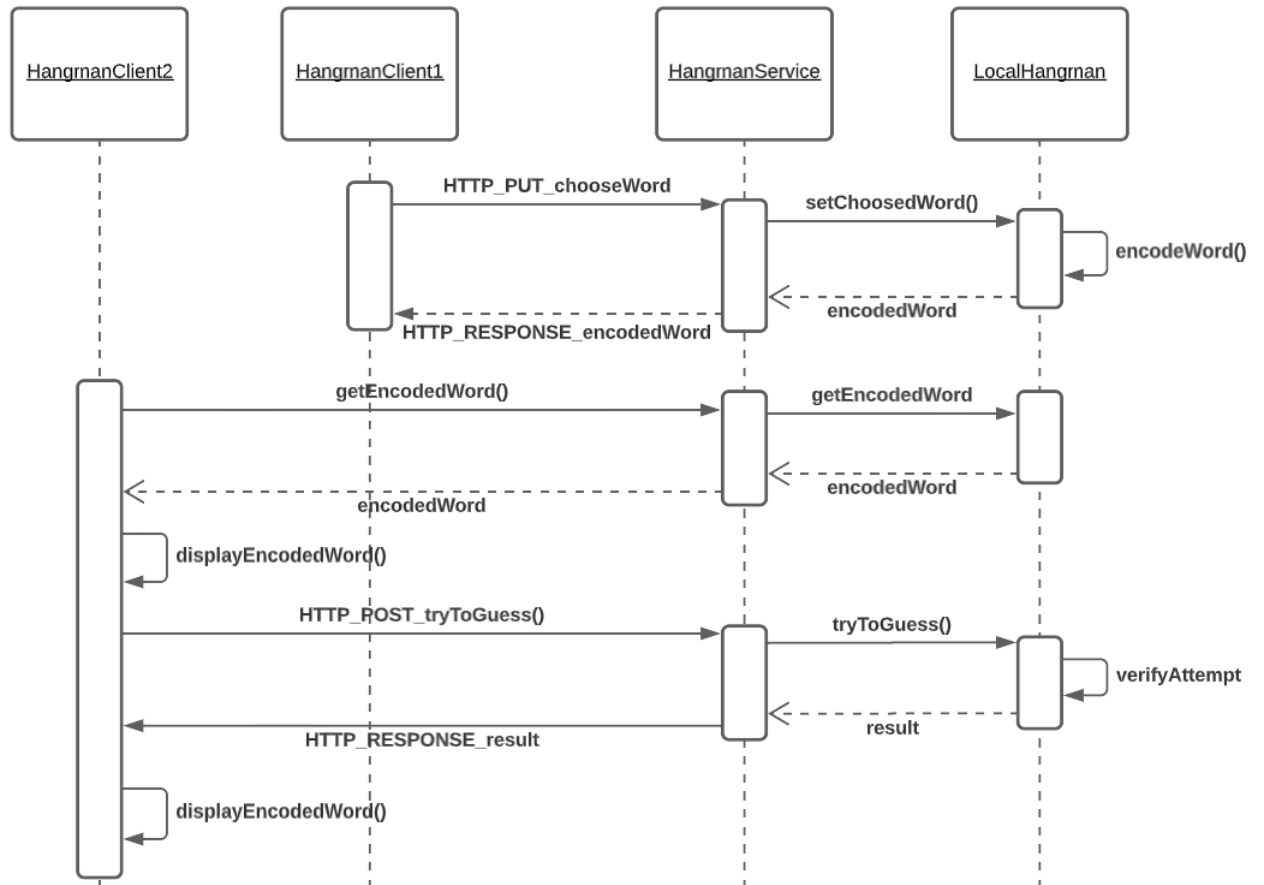


Figure 7: Game attempt Sequence Diagram.

Per quanto riguarda una partita, in figura 7 viene mostrata la sequenza di interazioni che avvengono quando un client sceglie la parola da far indovinare e l'altro client effettua un tentativo. Per non appesantire lo schema non sono stati rappresentati il `GameController` e il `GameApi` che si suppone siano collocati tra l'`HangmanService` e il `LocalHangman` ed effettuino la medesima sequenza di interazioni illustrata nel paragrafo precedente.

## 4 Implementation Details

In questa sezione verranno riassunti i meccanismi e le strategie adottate nella fase di implementazione, citando brevemente le tecnologie utilizzate. L'intero progetto è stato sviluppato in Java e si articola principalmente in due sottoparti: `HangmanClient` e `HangmanService`.

### 4.1 HangmanClient

Per quanto riguarda il Client esso contiene tutti i metodi che sono necessari per effettuare le richieste HTTP verso il server che fanno riferimento ai seguenti URL:

- `/users`: fa riferimento a tutte le richieste effettuate per la gestione degli utenti (ricerca utenti, connessione e disconnessione)
- `/lobbies`: si riferisce a tutte le richieste per la gestione delle lobbies (creazione, eliminazione, connessione ecc.)
- `/games`: è il riferimento per tutte le richieste inerenti alle partite (inizio di una partita, scelta della parola, effettuazione tentativi ecc.)

Ognuno dei percorsi indicati viene utilizzato per fare richieste coerenti con le necessità del client, che possono essere quindi di tipo GET, POST, PUT o DELETE. In aggiunta dove è necessario viene inserita come *pathParam* qualche informazione indispensabile per la richiesta (ad esempio un id).

Il client possiede un `main` nel quale viene istanziato e viene gestito l'invio delle richieste HTTP al server in base all'interazione dell'utente con l'interfaccia grafica a linea di comando.

### 4.2 HangmanService

Il server è stato sviluppato utilizzando il framework **Javalin** [2]. Esso è strutturato come un web service, possiede infatti varie risorse ognuna delle quali è dotata di un **Controller** e di un insieme di **Api**.

Le risorse sono quelle identificate nelle precedenti sezioni ovvero: User, Lobby e Game. Per ogni risorsa viene registrato il relativo controller all'indirizzo corrispondente, in modo da ricevere correttamente le richieste dei client e indirizzarle al componente di competenza. Mentre le API interagiscono con un'istanza locale di Hangman che realizza la business logic del sistema.

| games   |   |
|---------|---|
| POST    | /hangman/v0.1.0/games/{lobbyId}         |
| PUT     | /hangman/v0.1.0/games/{lobbyId}         |
| PUT     | /hangman/v0.1.0/games/attempt/{lobbyId} |
| lobbies |   |
| GET     | /hangman/v0.1.0/lobbies/{lobbyId}       |
| DELETE  | /hangman/v0.1.0/lobbies/{lobbyId}       |
| POST    | /hangman/v0.1.0/lobbies                 |
| GET     | /hangman/v0.1.0/lobbies                 |
| users   |   |
| POST    | /hangman/v0.1.0/users                   |
| GET     | /hangman/v0.1.0/users/{userId}          |
| DELETE  | /hangman/v0.1.0/users/{userId}          |

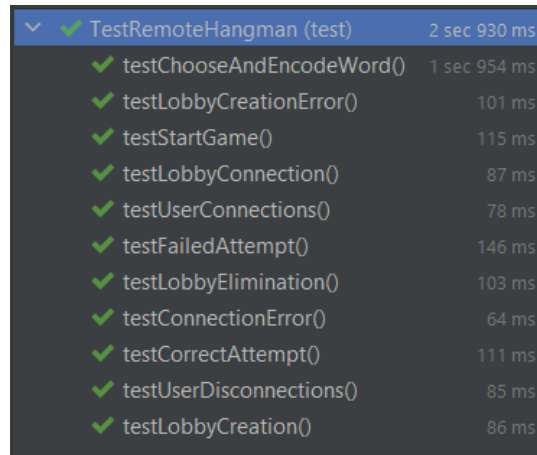
Figure 8: Swagger API docs.

Tutte le rotte registrate nel web server sono state documentate con l'ausilio di **Swagger** [4] (figura 8) e sono consultabili seguendo il percorso `/doc/ui`.

Mentre per quanto riguarda il livello di *presentation* è stata utilizzata la libreria **Gson** [1] per la serializzazione e deserializzazione degli oggetti in formato JSON.

## 5 Self-assessment / Validation

Per quanto riguarda il testing, oltre a test manuali sono stati realizzati dei test automatici con l'ausilio del framework **JUnit** [3]. Nel progetto è presente un modulo chiamato `test` che al suo interno contiene le tre classi sviluppate per il testing.



|                             |              |
|-----------------------------|--------------|
| ✓ TestRemoteHangman (test)  | 2 sec 930 ms |
| ✓ testChooseAndEncodeWord() | 1 sec 954 ms |
| ✓ testLobbyCreationError()  | 101 ms       |
| ✓ testStartGame()           | 115 ms       |
| ✓ testLobbyConnection()     | 87 ms        |
| ✓ testUserConnections()     | 78 ms        |
| ✓ testFailedAttempt()       | 146 ms       |
| ✓ testLobbyElimination()    | 103 ms       |
| ✓ testConnectionError()     | 64 ms        |
| ✓ testCorrectAttempt()      | 111 ms       |
| ✓ testUserDisconnections()  | 85 ms        |
| ✓ testLobbyCreation()       | 86 ms        |

Figure 9: JUnit tests.

La classe `AbstractTestHangman` contiene tutti i metodi di testing previsti che coprono tutte le funzionalità che il sistema deve necessariamente possedere (figura 9). Sono presenti test basici che verificano la corretta connessione e disconnessione di utenti al sistema, la corretta creazione e eliminazione delle lobby e il controllo della gestione delle eccezioni in caso di errore. Sono inoltre presenti test avanzati che si occupano di simulare un caso d'uso del sistema da parte di differenti utenti e testano quindi: la connessione ad una lobby, l'inizio di una partita con conseguente scelta della parola da indovinare, i tentativi sia corretti che errati da parte dell'utente e la progressione dei round.

Nello specifico tutti i test vengono eseguiti tramite due classi: `TestLocalHangman` che si occupa di verificare il funzionamento dell'istanza locale che verrà creata lato server, `TestRemoteHangman` che testa le stesse funzionalità sull'intero sistema distribuito, istanziando server e client.

## 6 Deployment Instructions

Essendo il progetto costituito da applicativi Java, è necessario eseguirlo su una macchina dotata di JVM, non sono richieste ulteriori configurazioni.

Inizialmente è necessario avviare il server, collocandosi con un terminale nella directory iniziale del progetto ed eseguendo il comando:

```
./gradlew --console plain server:run
```

Successivamente (solamente quando il server è stato avviato) per simulare i giocatori è sufficiente lanciare il seguente comando (uno per ogni giocatore desiderato):

```
./gradlew --console plain client:run
```

(L'opzione `--console plain` permette di non visualizzare i messaggi di gradle che appesantirebbero l'interfaccia di gioco)

## 7 Usage Examples

Avviando un client il sistema richiede di inserire un nickname e se il nickname non genera conflitti viene mostrato il menu.

```
Inserire un nickname: filippo00

##### MENU #####
[1] Crea una lobby
[2] Visualizza la lista delle lobby
[3] Esci
#####

Seleziona un'operazione:
```

Figure 10: Game menu.

Scegliendo l'operazione numero [1] viene creata una nuova lobby e il giocatore viene messo in attesa della connessione di un altro utente.

```
#####
Lobby: 1

In attesa di un altro giocatore...
#####
```

Figure 11: Lobby creation.

Se in alternativa viene selezionata l'operazione [2] nel menu, viene mostrata la lista delle lobbies disponibili (con il numero di giocatori già presenti al suo interno) e viene richiesto il codice della lobby a cui ci si vuole connettere.

```
#####LOBBIES#####  
Lobby 1 : (1/2)  
Lobby 2 : (2/2)  
Lobby 3 : (1/2)  
Lobby 4 : (1/2)  
Lobby 5 : (2/2)  
  
[0] Torna al menu  
#####  
  
Codice lobby a cui connettersi:
```

Figure 12: Lobbies list.

Quando due giocatori sono connessi ad una lobby inizia la partita, vengono assegnati i ruoli e il *GUESSER* viene messo in attesa finchè il *CHOOSER* non sceglie una parola.

```
*****  
* GUESSER *  
*****  
  
In attesa che l'altro giocatore scelga la parola...
```

Figure 13: Guesser waiting.



Il *CHOOSER* deve scegliere la parola da far indovinare, che viene successivamente codificata e ricevuta dal *GUESSER*.

```
*****
* CHOOSER *
*****

Parola da far indovinare: cartoncino

- + - - + - - + - +
```

Figure 14: Word choosing.

A questo punto il *CHOOSER* visualizzerà sulla propria interfaccia di gioco tutti i tentativi del *GUESSER* mentre quest'ultimo proverà ad indovinare la parola inserendola per intero o provando una sola lettera. Se il tentativo è errato viene disegnata una parte dell'impiccato e vengono decrementati i tentativi rimasti.

```
Inserire una lettera o una parola: q
-----
| Round vinti: 0/2      |
| Tentativi rimasti: 3 |
-----

ERRORE!
  |----
  |
  |
  |
  |
  |
  |
-----
- + - - + - - + - +
```

Figure 15: Wrong attempt.

Mentre se il tentativo è corretto viene sostituito il carattere corrispondente nella parola codificata.

```
Inserire una lettera o una parola: c
-----
| Round vinti: 0/2      |
| Tentativi rimasti: 3 |
|-----|

INDOVINATO!
|----
|
|
|
|
|
|-----
c + - - + - c + - +
```

Figure 16: Correct attempt.

Se il *GUESSER* riesce ad indovinare la parola prima che venga disegnato per intero l'impiccato (quindi con meno di cinque errori) vince il round.

```
Inserire una lettera o una parola: cartoncino
-----
| Round vinti: 1/2      |
| Tentativi rimasti al guesser: 3 |
|-----|

|----
|
|
|
|
|
|-----
c a r t o n c i n o
ROUND VINTO!
```

Figure 17: Round won.

Se invece vengono commessi cinque errori è il *CHOOSER* a vincere il round e viene mostrata la figura completa dell'impiccato.

```
Inserire una lettera o una parola: cartone

-----
| Round vinti: 0/2                               |
| Tentativi rimasti al guesser: 0 |
-----

  |---|
  |  0
  |  /\
  |  /\
  |
  |
-----
C + - - + - C + - +

ROUND PERSO!
```

Figure 18: Round lost.

Questa tipologia di interazioni viene ripetuta finchè uno dei giocatori non vince due round (con l'assegnazione dei ruoli descritta in precedenza). Al termine della partita i due giocatori vengono reindirizzati al menu principale.

## 8 Conclusions

Con la progettazione e l'implementazione di Hangman Online, è stato realizzato un sistema distribuito basato su un'architettura client-server con l'utilizzo di HTTP come protocollo di comunicazione e la progettazione di ReST API per quanto riguarda l'interazione con il servizio web. Hangman Online consiste quindi in un gioco multiplayer che garantisce scalabilità e consistenza in tutti i suoi possibili casi d'uso da parte degli utenti.

### 8.1 Future Works

Per rendere Hangman Online più realistico sarebbe necessaria un'interfaccia grafica che possa garantire una UX più coinvolgente. Sarebbe inoltre interessante sviluppare maggiormente la parte relativa agli utenti, aggiungendo meccanismi di autenticazione più complessi e integrando il sistema con un database per garantire la persistenza dei dati e realizzare eventuali storici o classifiche.

### 8.2 What did I learned

Durante la realizzazione di questo progetto ho messo in pratica vari aspetti appresi durante lo svolgimento del corso, approfondendo in generale tutte le tematiche che riguardano il design e l'implementazione di un sistema distribuito e nello specifico ho sperimentato una verticalizzazione su tecnologie per realizzare servizi web, collocandoli in un contesto di distribuzione.

## References

- [1] Gson. <https://github.com/google/gson>.
- [2] Javalin. <https://javalin.io/>.
- [3] JUnit 5. <https://junit.org/junit5/>.
- [4] Swagger. <https://swagger.io/>.