Final Report on Sudokoop (Sistemi Distribuiti)

Alex Siroli
alex.siroli@studio.unibo.it

Alice Mastrilli
alice.mastrilli@studio.unibo.it

March 2025

Sudokoop è una piattaforma online per il gioco del sudoku, disponibile sia in modalità **singola** che **multiplayer**, progettata per offrire un'esperienza interattiva, collaborativa e competitiva in tempo reale. Gli utenti possono creare o unirsi a stanze di gioco tramite un'applicazione web, scegliendo tra diverse modalità di gioco.

La partita può essere avviata in modalità **Single-Player**, in cui il giocatore affronta il sudoku in autonomia, oppure in modalità **Multi-Player**, che richiede la partecipazione di almeno due giocatori e consente di osservare le azioni degli altri partecipanti durante la partita.

L'architettura di **Sudokoop** è basata su un modello **client-server**, in cui ogni client comunica con il sistema attraverso le **API** esposte dal server. L'interazione in tempo reale tra gli utenti e il sudoku è resa possibile dall'uso di tecnologie basate su protocolli real-time, come **Socket.IO** [**2**], fondamentali in un contesto di Sistemi Distribuiti.

1 Goal/Requirements

In questa sezione verranno fornite informazioni più approfondite sugli obiettivi del progetto, sui requisiti richiesti e sui risultati attesi.

1.1 Questions/Answers

- *Cosa succede all'avvio dell'applicativo?*
  All'avvio dell'applicativo l'utente può effettuare il login oppure registrarsi.

  La registrazione può essere effettuata inserendo username e password e non va a buon fine se un utente con lo stesso username si era già registrato.

Il login, che va a buon fine se un utente si era già registrato, non è consentito se si tenta di accedere con un account già connesso al sistema o se le credenziali inserite sono errate.

- *Cosa succede quando l'utente effettua il login o registrazione?*
  Una volta fatto il login o registrazione, l'utente può scegliere se iniziare una partita in Single-Player, visualizzare una leaderboard con le statistiche relative a tutte le partite Single-Player, creare una lobby oppure unirsi a una esistente. Inotre è possibile consultare le statistiche relative al proprio account e una sezione crediti.

- *Come funziona la modalità Single-Player?*

  In questa modalità, il giocatore seleziona il livello di difficoltà del sudoku e inizia la partita con tre vite a disposizione.

  - All'avvio del gioco viene attivato un timer per tenere traccia del tempo impiegato.
  - Se l'utente inserisce un numero corretto, la cella si colora di verde; in caso di errore, la cella diventa rossa e si perde una vita.
  - Il giocatore vince se riesce a completare il sudoku avendo ancora almeno una vita disponibile.
  - Al termine della partita, il giocatore sceglie se rigiocare o tornare alla Home.

  Al termine di una partita in modalità single-player, vengono registrate le statistiche dell'utente, tra cui il numero di vittorie e sconfitte, consultabili nella sezione **Account** della Home Page. In caso di vittoria, il nome dell'utente, il tempo impiegato e la difficoltà del sudoku vengono salvati nella classifica generale.

- *Come viene visualizzata la classifica generale?*

  Una volta effettuato l'accesso, l'utente può consultare una classifica globale delle vittorie ottenute in modalità single-player. La **leaderboard** mostra il livello di difficoltà dei sudoku e ordina le partite in base al minor tempo impiegato per completarli.

- *Come viene gestita la creazione di una lobby?*

  Creare una lobby significa che il sistema genera un codice univoco di sei caratteri e assegna il ruolo di *master* al giocatore che la crea, che viene aggiunto automaticamente alla lobby. Il server registra la lobby tra quelle disponibili.

- *Come accedere a una lobby esistente?*

  Gli utenti che hanno fatto l'accesso al sistema, possono unirsi a una lobby esistente inserendo il relativo codice. L'operazione non va a buon fine se la lobby è piena o inesistente; il numero massimo di giocatori in lobby è 10.

- *Cosa succede una volta entrato in lobby?*

All'interno della lobby i giocatori possono comunicare tra loro tramite una **chat in lobby**. Viene comunicato per messaggio anche l'ingresso e l'uscita di un giocatore dalla lobby.

Il master può scegliere di iniziare una partita, selezionando la difficoltà e la modalità di gioco (**Coop** o **Versus**). Il gioco può iniziare solamente se ci sono almeno due giocatori in lobby.

- *Come funziona la modalità Coop (cooperativa)?*

  In questa modalità, i giocatori collaborano per risolvere il sudoku insieme.

  - Tutti visualizzano e interagiscono sullo stesso sudoku.
  - L'intera squadra ha a disposizione tre vite complessive per completarlo.
  - La dinamica di gioco è simile a quella del Single-Player, con la differenza che ogni giocatore può vedere le celle selezionate dagli altri in tempo reale.

- *Come funziona la modalità Versus?*

  In questa modalità, la partita non inizia immediatamente ma viene chiesto ai giocatori di dividersi in due squadre: **Blu** o **Gialla**. Il master può avviare la partita solo quando ogni squadra ha almeno un giocatore e tutti hanno scelto la propria fazione.

  Avviata la partita, le due squadre competono tra loro per risolvere il sudoku.

  - Tutti i giocatori vedono e interagiscono sullo stesso sudoku.
  - Quando un giocatore seleziona una cella, questa si colora con il colore della sua squadra ed è visibile da tutti. Se il numero inserito è corretto, la cella rimane del colore di quella squadra, altrimenti diventa rossa.
  - La modalità di gioco è diversa dalle altre viste sopra. Regole speciali del VersusGame:
    - **Nessuna vita**: l'inserimento di un valore errato determina l'eliminazione di quel giocatore. Se tutti i giocatori di una squadra vengono eliminati, vince automaticamente l'altro team.
    - **Assegnazione punti**: ogni numero inserito correttamente assegna un punto alla squadra corrispondente.
    - **Esito della partita**: se al momento del completamento della partita entrambe le squadre sono ancora in gioco, vince la squadra con il maggior numero di punti; in caso di parità, la partita termina con un pareggio.

- *Cosa succede se un giocatore abbandona la partita?*

  Un giocatore può decidere di lasciare la partita in qualsiasi momento. Se ciò accade:

  - Gli altri giocatori possono continuare normalmente la partita, anche se ne rimane solo uno;

  - Se in modalità Versus, l'ultimo giocatore di una squadra abbandona la partita, determina la vittoria dell'altro team.
  - Se il giocatore rimasto da solo tenta di rigiocare, viene riportato alla lobby con un messaggio che segnala l'insufficienza di giocatori;
  - Se il *master* abbandona, il sistema assegna automaticamente il ruolo di *master* a un altro giocatore.

- *Cosa succede al termine di una partita multiplayer se nessun giocatore si è aggiunto alla lobby durante la partita conclusa?*

  Alla conclusione di una partita, il *master* ha la possibilità di:
  - Iniziare una nuova partita con la stessa difficoltà della precedente;
  - Tornare alla lobby.

- *Cosa succede al termine di una partita multiplayer se c'è almeno un nuovo giocatore che si è aggiunto alla lobby durante la partita conclusa?*

  Se un giocatore entra nella lobby mentre una partita è in corso, rimane in attesa per partecipare alla partita successiva.

  Quando il master preme *rigioca* a fine partita:
  - In modalità **CoopGame**, il giocatore in attesa entra direttamente in gioco;
  - In modalità **VersusGame**, il sistema torna alla selezione delle squadre per permettere al nuovo giocatore di scegliere la propria squadra;
  - Se non ci sono almeno due giocatori, il sistema torna alla lobby, informando che il numero di giocatori è insufficiente.

1.2 Scenarios

In uno scenario di utilizzo, un utente che apre l'applicativo deve, come prima cosa, accedere al sistema, facendo un login o procedendo con la registrazione.

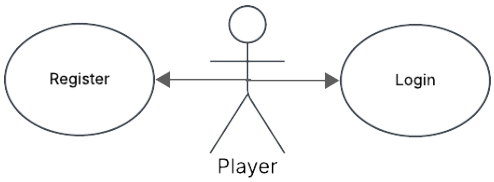

Figure 1: Apertura dell'applicativo

Effettuato l'accesso, le scelte principali dell'utente sono: iniziare una partita in Single-Player, visualizzare una leaderboard con le statistiche relative a tutte le partite Single-Player, creare una lobby oppure unirsi a una esistente. Per unirsi a una lobby esistente, viene richiesto di inserire il codice della lobby a cui ci si vuole unire.

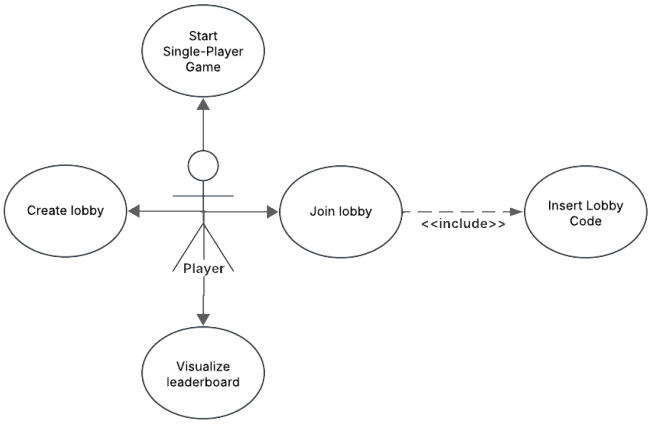

Figure 2: HomePage

Se si sceglie di entrare in lobby, il master può avviare una partita multiplayer se ci sono almeno due giocatori. Deve prima scegliere la difficoltà della partita; nel caso di VersusGame i giocatori devono dividersi in squadre.

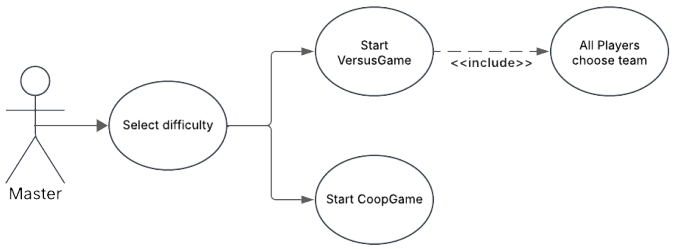

Figure 3: Lobby

2 Requirements Analysis

User Requirements

- Possibilità di registrarsi con username e password, mantenendo uno stato online/offline.
- Creare o unirsi a una lobby già esistente.
- Selezionare una modalità di gioco (SinglePlayer, Coop, Versus).
- Vedere in tempo reale le mosse degli altri giocatori, con colorazione delle celle.
- Utilizzare la chat interna alla lobby.

- Gestire la fine della partita: in caso di Coop, condivisione delle vite; in caso di Versus, eliminazione e punteggio.

Functional Requirements

- **REST API** per gestire registrazione, login, logout e modalità single-player.
- **Socket.IO** per la lobby, la chat e la modalità multi-giocatore.
- Gestione di un timer e di vite comuni o punteggi.
- Persistenza su MongoDB dei dati degli account dei giocatori, delle loro statistiche e della leaderboard. Salvataggio in *sessionStorage* dei dati delle lobby.
- Generazione dei codici univoci di 6 caratteri per le lobby.

Non-Functional Requirements

- L'architettura deve supportare la comunicazione in tempo reale con latenza ridotta.
- Il sistema deve essere resistente a disconnessioni inattese: ad esempio, se il master abbandona, il ruolo di master passa automaticamente a un altro giocatore.
- Deve essere possibile eseguire facilmente il deployment con Docker [3].
- Interfaccia user-friendly, ottimizzata anche per dispositivi mobili.

Implicit Hypothesis

- Si assume che i giocatori abbiano una connessione internet stabile, pur gestendo riconnessioni automatiche in caso di breve disconnessione.
- Si presuppone che i giocatori non superino un numero tale da saturare le risorse del server (limite massimo 10 per lobby, modificabile in futuro).
- Si assume che i giocatori conoscano le regole base del sudoku (possibile inserimento futuro di un tutorial al primo login).

3 Design

L'applicazione è sviluppata in un'ottica client-server con comunicazioni real-time e si basa su un'architettura MEVN (MongoDB, Express.js, Vue.js, Node.js [1]).

- **Database (MongoDB)**: contiene tutte le informazioni relative agli utenti, le partite giocate da ciascuno e le classifiche generali. Viene gestito tramite **Mongoose** [**4**], che fornisce un'interfaccia schema-based per MongoDB.
- **Backend (Node.js + Express.js)**:

  - Espone API REST per le operazioni sincrone come registrazione, login e gestione del sudoku in modalità single-player, utilizzando **Axios** per la comunicazione con il frontend.
  - Gestisce le funzionalità real-time (lobby, chat e partite multiplayer) attraverso Socket.io.

- **Frontend (Vue.js)**: organizzato in una struttura modulare con componenti Vue, creati con Options API, e routing gestito tramite Vue Router.

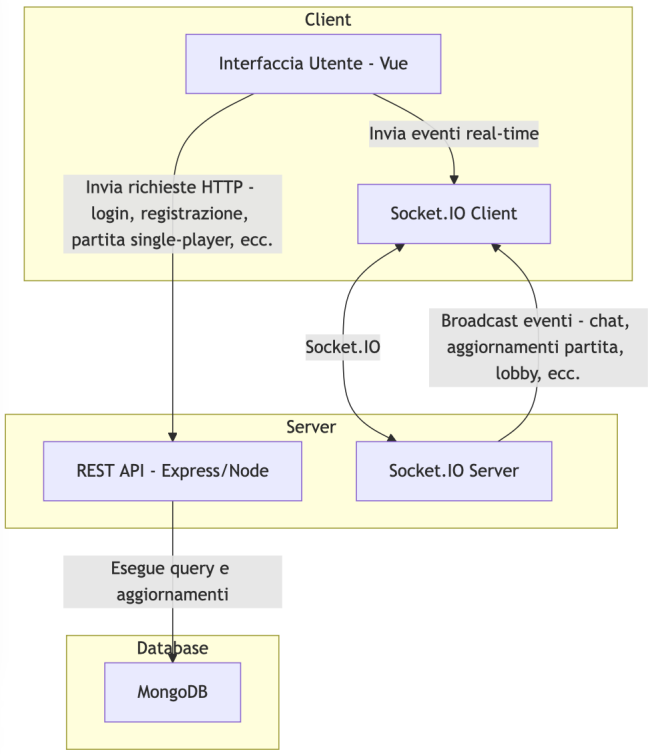

Figure 4: Struttura generale del sistema.

3.1 Structure

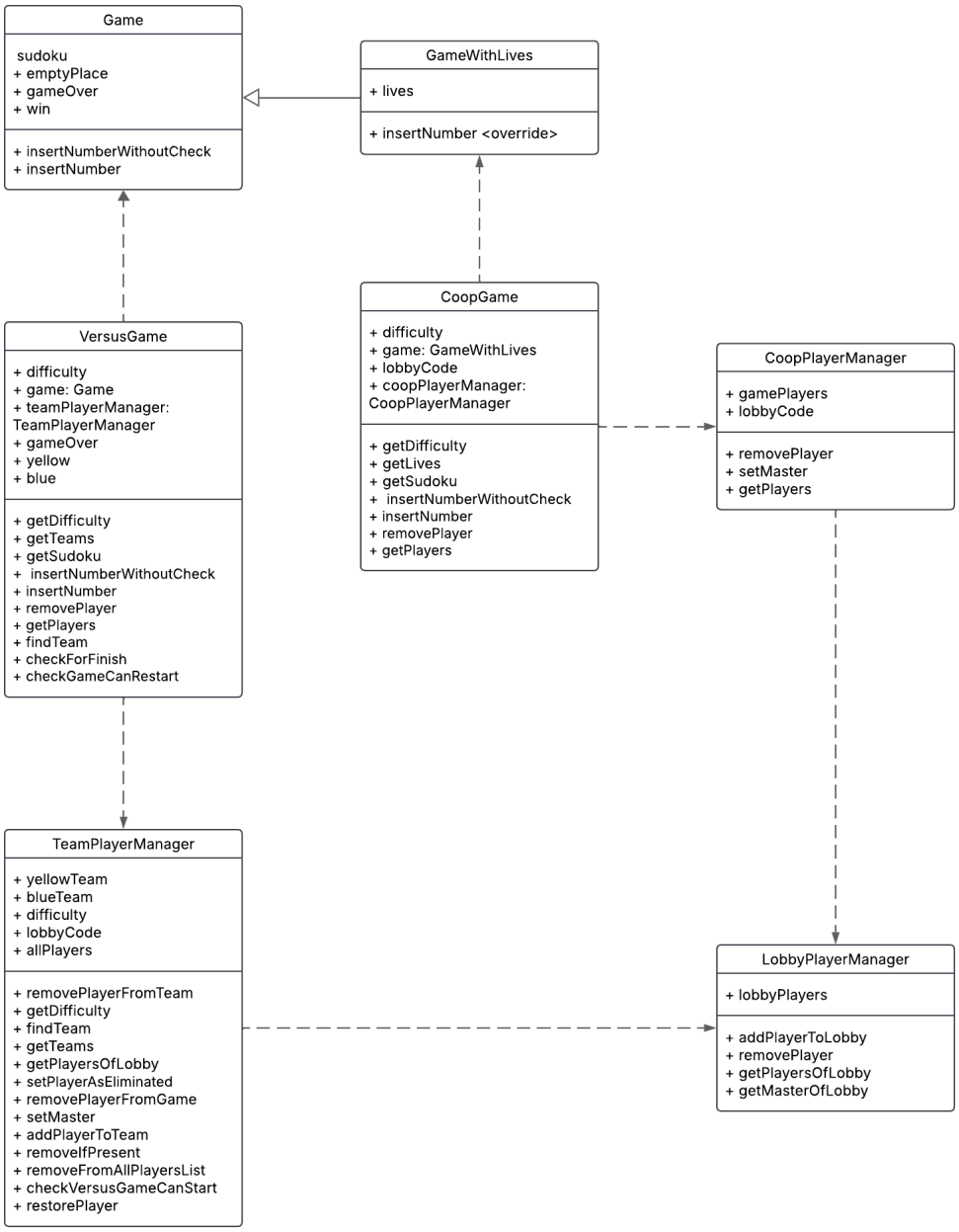

Figure 5: UML delle classi per la gestione del gioco

Il backend è stato organizzato in modo da garantire la massima modularità, consentendo di estendere con facilità il dominio, qualora in futuro si volesse aggiungere una nuova modalità di gioco o modificare queste presenti.

Di seguito vengono descritte le principali classi del sistema.

3.1.1 Game

La classe principale che gestisce la logica del sudoku è **Game**.

Questa classe si occupa di creare e amministrare un'istanza specifica di sudoku in base al livello di difficoltà scelto. Il sudoku viene generato utilizzando un'apposita libreria GitHub, **sudoku-gen** (`https://github.com/petewritescode/sudoku-gen`), che fornisce sia la soluzione completa sia una stringa (puzzle) rappresentante il sudoku con alcune celle vuote. Quest'ultima viene utilizzata come griglia di gioco, che gli utenti dovranno completare.

Il metodo **insertNumber** consente di inserire un valore in una determinata posizione (riga e colonna) del puzzle. Prima di aggiungere il numero, il metodo verifica la correttezza dell'inserimento confrontandolo con la soluzione del sudoku generato.

3.1.2 GameWithLives

Questa classe estende **Game**, introducendo il concetto di vite e rappresentando la modalità **Single-Player**. Il metodo **insertNumber** viene modificato in modo da decrementare il numero di vite disponibili ogni volta che un giocatore inserisce un valore errato.

3.1.3 CoopGame

Questa classe rappresenta la modalità di gioco **multiplayer cooperativa**. Utilizza le classi **GameWithLives** per la gestione della partita e **CoopPlayerManager** per la gestione dei giocatori coinvolti.

3.1.4 VersusGame

Questa classe rappresenta la modalità di gioco **multiplayer versus**.

A differenza della modalità cooperativa, qui non sono previste vite: il gioco è quindi modellato direttamente dalla classe **Game**. Per la gestione dei giocatori, VersusGame utilizza **TeamPlayerManager**, che si occupa di gestire le due squadre di giocatori.

Il metodo **insertNumber** viene modificato con le seguenti regole:

- Se un giocatore inserisce un numero errato, viene eliminato dalla partita.
- Se il numero è corretto, la sua squadra guadagna un punto.

Le squadre e i rispettivi punteggi sono gestiti attraverso i campi **yellow** e**blue**.

3.1.5 I PlayerManager

I gestori dei giocatori all'interno dell'applicazione sono **CoopPlayerManager**, **TeamPlayerManager** e **LobbyPlayerManager**.

Tutti i giocatori devono prima entrare in una lobby per poter partecipare a una partita multiplayer. Per questo motivo, sia CoopPlayerManager che TeamPlayerManager dipendono da LobbyPlayerManager, che gestisce l'ingresso e l'uscita dei giocatori dalla lobby.

L'esigenza di avere manager separati nasce dal fatto che, in alcuni casi, il numero di giocatori in una lobby potrebbe non coincidere con il numero di giocatori effettivamente in partita. Questo accade, ad esempio, quando un giocatore si unisce alla lobby mentre una partita è già in corso: in tal caso, dovrà attendere la fine della partita prima di poter giocare.

CoopPlayerManager

Il **CoopPlayerManager** viene creato all'avvio di una partita multiplayer cooperativa e inizialmente contiene gli stessi giocatori presenti in lobby in quel momento. Se un giocatore abbandona la lobby o la partita, verrà automaticamente rimosso sia dal LobbyPlayerManager che dal CoopPlayerManager. Se, invece, un nuovo giocatore entra nella lobby a partita iniziata, rimarrà lì fino al termine della partita. Quando verrà avviata una nuova partita, entrerà automaticamente nel nuovo CoopPlayerManager.

TeamPlayerManager

Il **TeamPlayerManager** viene creato subito dopo aver selezionato la modalità VersusGame, nel momento in cui i giocatori devono dividersi in squadre. Questa classe è più complessa rispetto al CoopPlayerManager, poiché deve gestire l'ingresso e l'uscita dei giocatori sia durante la fase di selezione delle squadre che nel corso della partita stessa. Possiede, inoltre, il metodo *checkVersusGameCanStart*, per verificare che tutti i giocatori abbiano scelto una squadra e che ciacun team abbia almeno un membro, e il metodo *setPlayerAsEliminated*, chiamato durante la partita per segnare che un giocatore è stato eliminato dopo che ha inserito di un numero sbagliato.

3.2 Behaviour

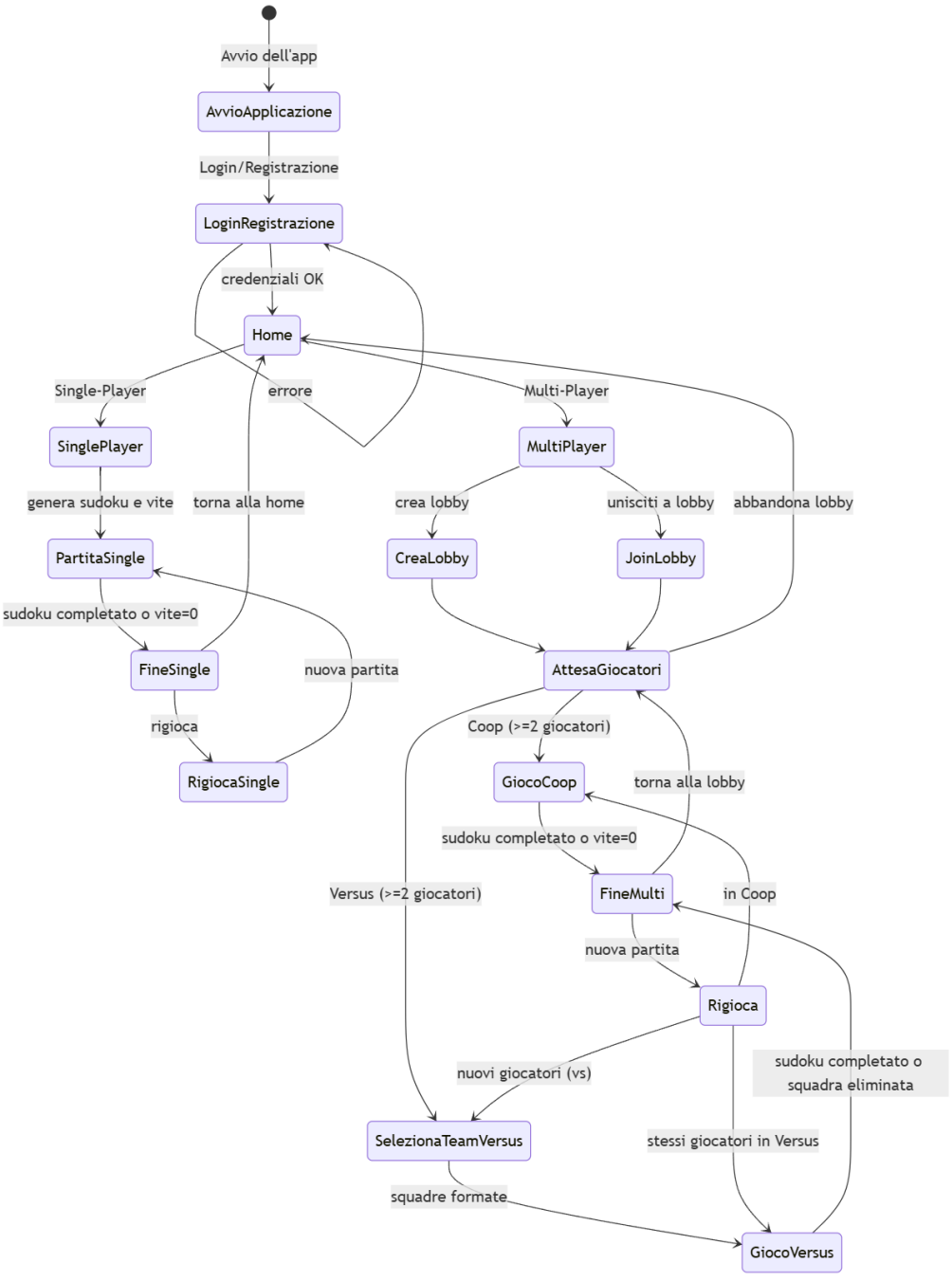

Figure 6: Diagramma degli stati

La sequenza di stati inizia con l'**Avvio dell'applicazione**, in cui l'utente non è ancora autenticato. Da qui si passa alla fase di **Login/Registrazione**, dove l'utente inserisce le credenziali o ne crea di nuove. Se le credenziali sono accettate, si approda alla **Home**, mentre in caso di errore si rimane sullo stesso stato (*LoginRegistrazione*).

Dalla **Home**, l'utente ha due principali scelte:

- **Single-Player:** Selezionando questa modalità, il sistema genera un Sudoku con 3 vite, e si entra nello stato *PartitaSingle*. La partita termina in *FineSingle* al completamento del puzzle o all'esaurimento delle vite. Da *FineSingle*, l'utente può rigiocare immediatamente (stato *RigiocaSingle*) oppure tornare alla *Home*. In caso di rigioco, viene creata una nuova partita single-player (*PartitaSingle*).
- **Multi-Player:** L'utente può scegliere di *creare una lobby* (*CreaLobby*) oppure *unirsi a una esistente* (*JoinLobby*). In entrambi i casi, si approda allo stato di *AttesaGiocatori* (Lobby), dove altri utenti possono unirsi o il giocatore può abbandonare tornando alla *Home*. Una volta raggiunto il numero minimo di giocatori, si può avviare:
  - **Coop:** si entra nello stato *GiocoCoop*, in cui tutti collaborano sullo stesso Sudoku, condividendo le vite. Al termine (puzzle risolto o vite esaurite), si passa a *FineMulti*.
  - **Versus:** occorre prima definire le squadre in *SelezionaTeamVersus*; poi inizia lo stato *GiocoVersus*, con punteggi e possibili eliminazioni. Quando la partita si chiude (Sudoku completo o una squadra interamente eliminata), si arriva comunque a *FineMulti*.

Nello stato di *FineMulti*, il *master* decide se iniziare una *nuova partita* (*Rigioca*) oppure *tornare alla lobby* (stato *AttesaGiocatori*). In caso di rigioco, si può restare nella stessa modalità, oppure verificare la presenza di nuovi giocatori (in Versus ciò può comportare un ritorno a *SelezionaTeamVersus*).

3.3 Interaction

Gestione delle API e dei WebSocket

L'autenticazione degli utenti (registrazione e login) e la modalità single-player si basano su un sistema RESTful, in cui il client comunica con il server attraverso richieste HTTP gestite da Express.js.

- **Registrazione**

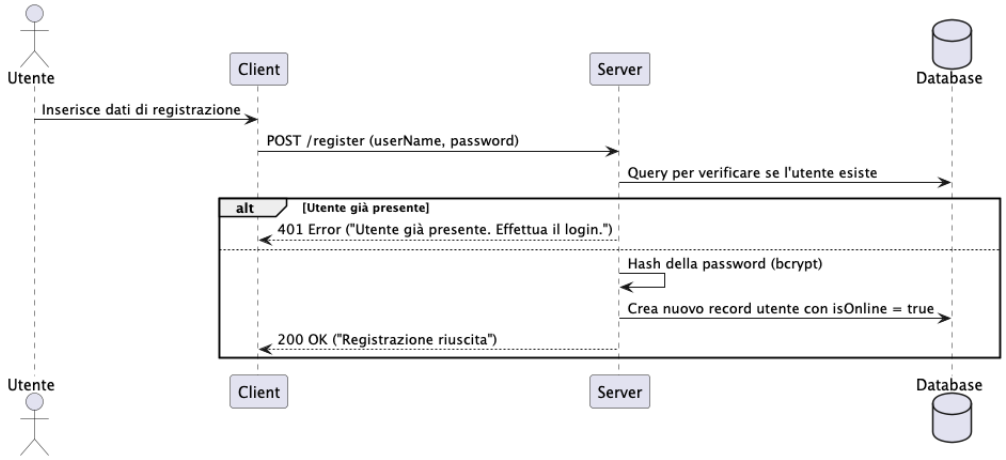

Figure 7: Registrazione di un utente

Per registrarsi, il *client* effettua una `POST /register` inviando un `username` e una `password`. Il *server* controlla innanzitutto se l'utente esista già nel database. In caso affermativo, risponde con un errore 401; in caso contrario, procede con l'hashing della password (usando **bcrypt**) e salva l'utente. Conclusa tale operazione, il *server* invia una risposta di successo (`200 OK`) al *client*.

- **Login**

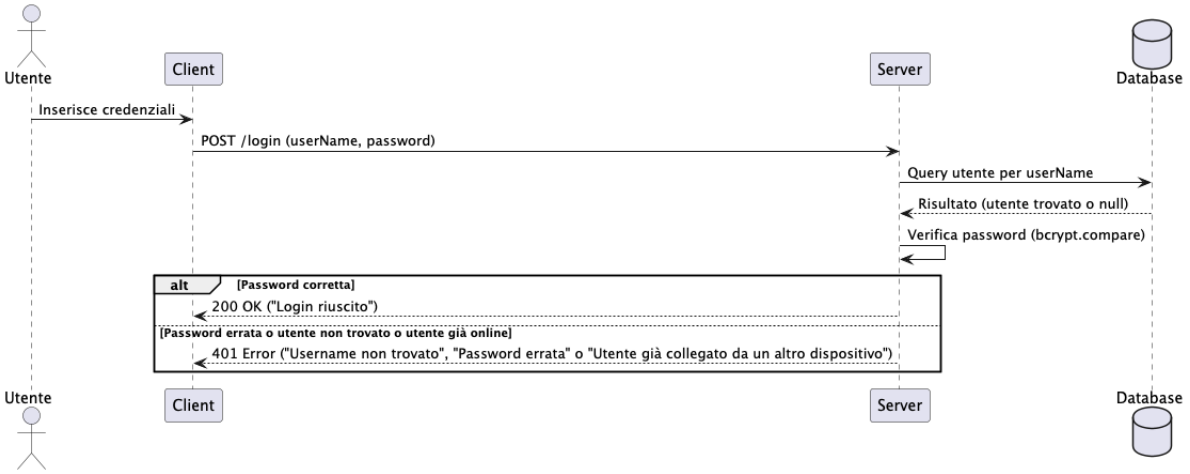

Figure 8: Login di un utente

La fase di login ha inizio quando il *client* invia una `POST /login` con le credenziali (`username`, `password`). Il *server* recupera l'utente dal database e confronta la password ricevuta con l'hash memorizzato, usando `bcrypt.compare()`. Qualora non coincidano, oppure nel caso in cui l'utente sia già online, risponde con errore 401; altrimenti, aggiorna lo stato `isOnline=true` per l'utente e ritorna `200 OK`.

- Logout

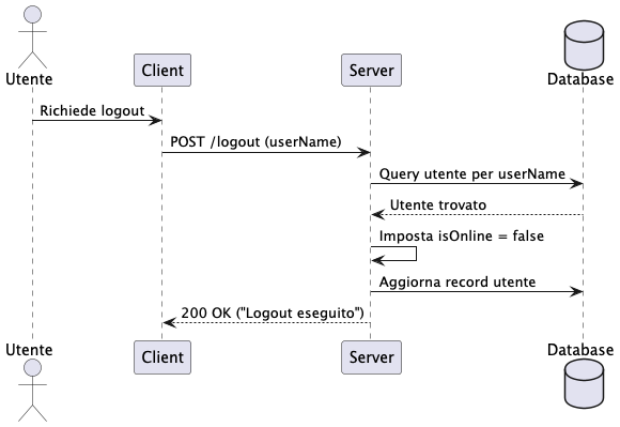

Figure 9: Logout di un utente

Per il logout, il *client* invia una `POST /logout` specificando lo `username`. Il *server* individua l'utente nel database, imposta `isOnline=false` e salva la modifica. Infine, fornisce al *client* un riscontro di successo (`200 OK`).

- **Modalità Single-Player**

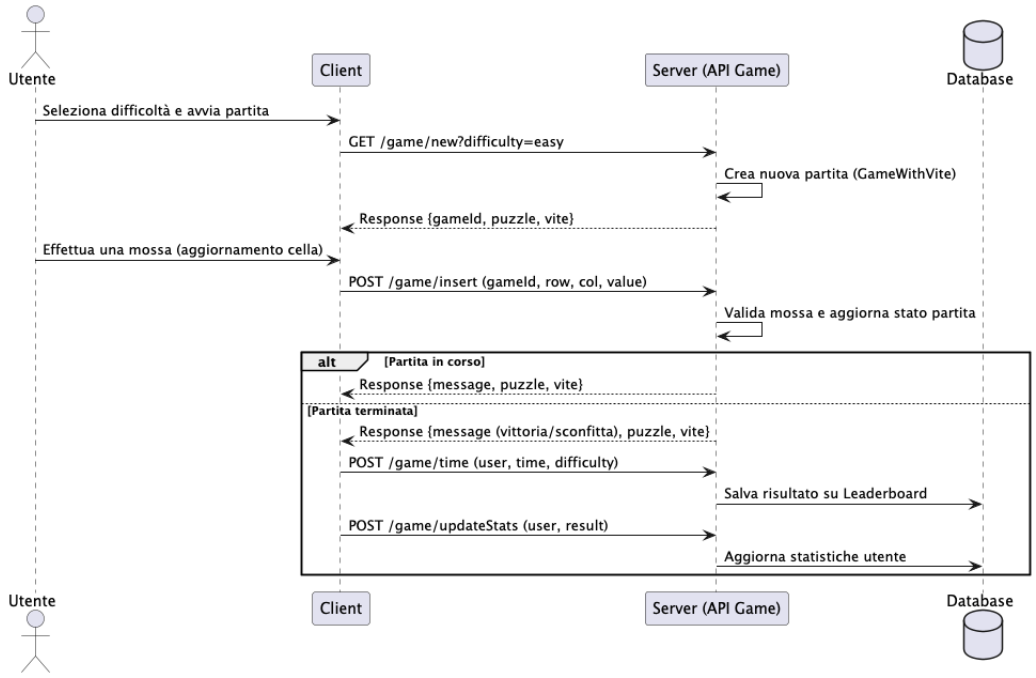

Figure 10: Gioco Single-Player

La modalità *Single-Player* si avvia quando l'utente richiede `GET /game/new?difficulty=...`, selezionando la difficoltà desiderata. Il *server* genera un sudoku con `sudoku-gen`, istanzia un oggetto `GameWithLives` (che include la gestione delle vite) e restituisce al *client* il puzzle iniziale e il numero di vite (generalmente 3). Per ogni mossa, il *client* invia una `POST /game/insert` con le coordinate e il valore inserito: se corretto, il puzzle viene aggiornato, altrimenti le vite calano. Il gioco termina al completamento corretto di tutte le celle o allo scadere delle vite. In caso di vittoria, il *client* salva il tempo nella *leaderboard* e aggiorna le statistiche `wins/losses` dell'utente.

Comunicazione in tempo reale (Socket.io)

Le funzionalità *multiplayer* e la *lobby* utilizzano *Socket.io*.

- Creazione Lobby

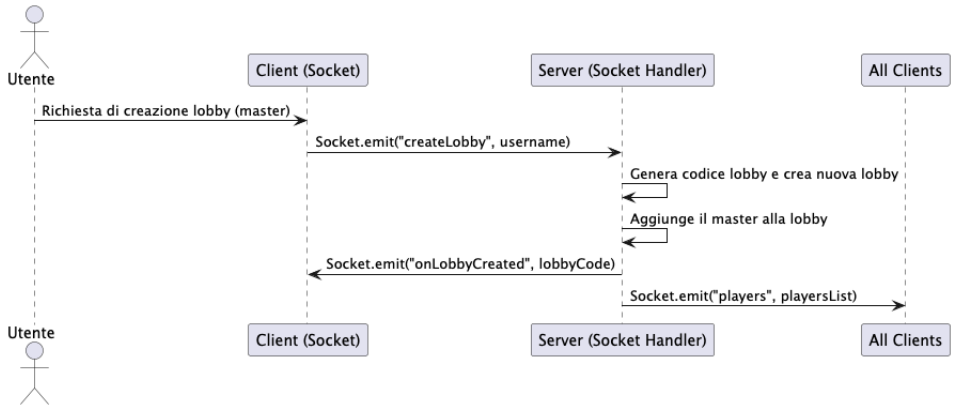

Figure 11: Creazione di una nuova lobby (Coop o Versus).

Quando un utente desidera creare una lobby, emette l'evento `createLobby` inviando il proprio `username`. Il *server* genera un nuovo *lobbyCode*, assegna all'utente il ruolo di **master** e lo fa unire automaticamente alla relativa stanza di Socket.io. Infine, comunica al *client* il codice appena creato (`onLobbyCreated(lobbyCode)`).

- Accesso a una Lobby

Per entrare in una lobby già esistente, il *client* emette `joinLobby{username, code}`. Il *server* controlla che la lobby sia valida e che non abbia già raggiunto il limite di giocatori (10). In caso positivo, aggiunge l'utente alla stanza di socket.io e risponde con `"Ok"`. Altrimenti, ritorna l'errore opportuno (lobby inesistente o piena). A ogni modifica della lista dei giocatori, il *server* aggiorna tutti gli utenti in broadcast con l'evento `players`.

- Chat in Lobby

La chat in lobby si realizza emettendo `socket.emit("lobbyMessage", {lobbyCode, author, text})` dal *client* verso il *server*, il quale inoltra il messaggio a tutti i partecipanti (`socket.emit("lobbyMessage", {...})` in broadcast). In tal modo, ciascun utente visualizza immediatamente i messaggi inviati da chiunque si trovi nella stessa *lobby*.

- Avvio Partita in Modalità Coop

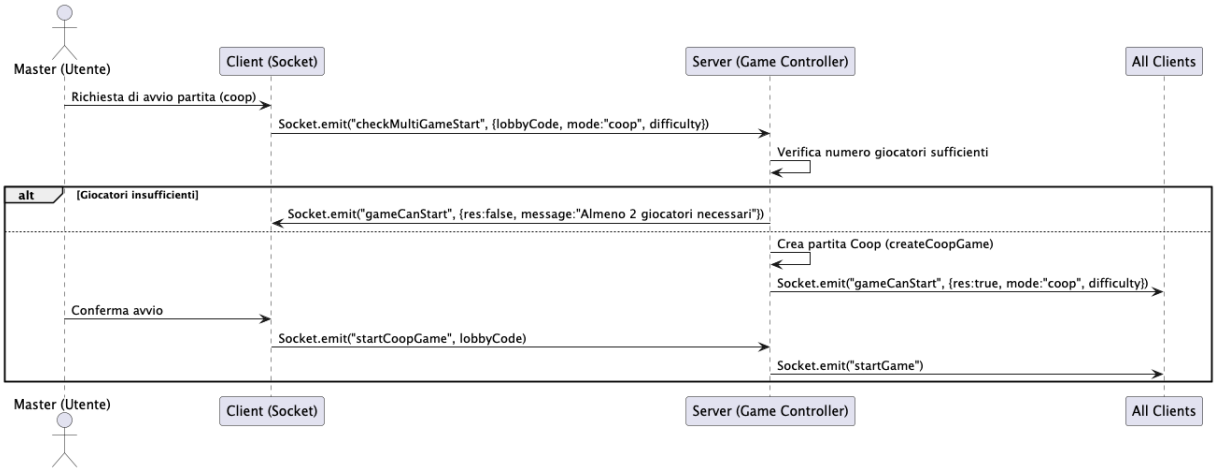

Figure 14: Richiesta di avvio Coop-Game da parte del Master

Per avviare la modalità *Coop*, il *master* verifica anzitutto che nella *lobby* siano presenti almeno due giocatori, emettendo al server l'evento `checkMultiGameStart`, e indicando `coop` come modalità di gioco. In caso il numero di giocatori non sia sufficiente, il server lo comunica al master, altrimenti crea una nuova istanza di `CoopGame` e comunica a tutti i client della lobby che il gioco può iniziare. Alla ricezione dell'evento `gameCanStart`, tutti i client passano alla schermata di gioco.

- Avvio Partita in Modalità Versus

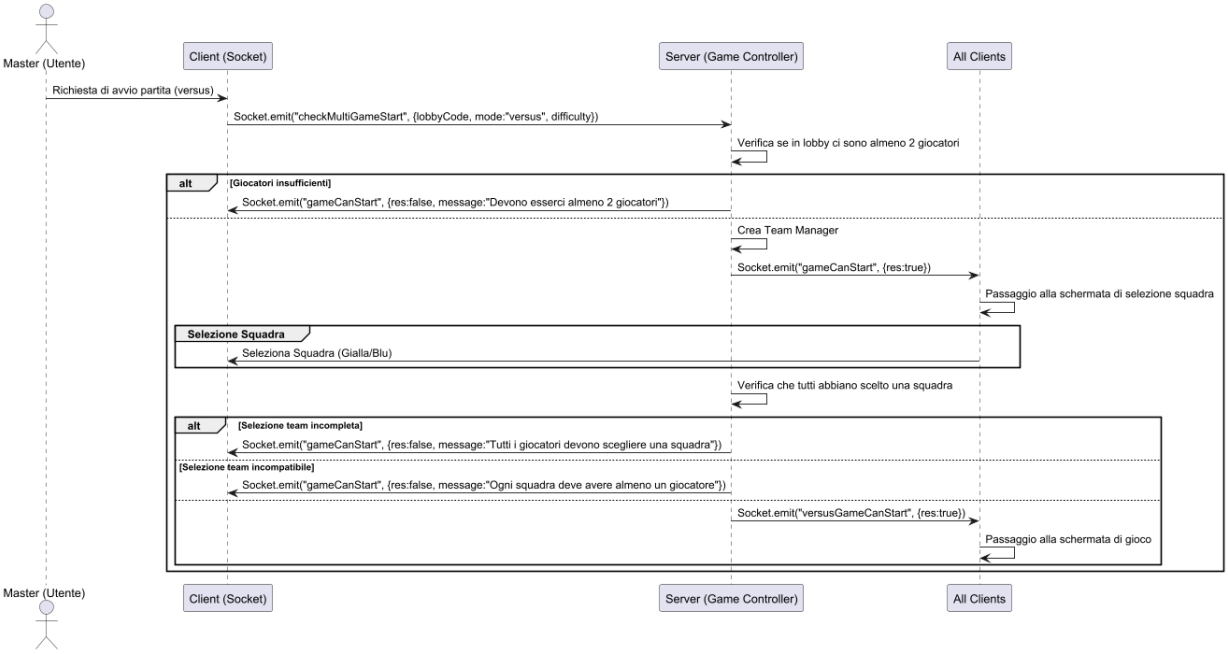

Figure 15: Avvio Versus-Game

Per avviare la modalità *Versus*, il *master* verifica anzitutto che nella *lobby* siano presenti almeno due giocatori, emettendo al server l'evento `checkMultiGameStart`, specificando `versus` come modalità di gioco.

Se il numero di giocatori è insufficiente, il server lo notifica al master. In caso contrario, crea una nuova istanza di `TeamManager` e informa tutti i client della lobby che il gioco può iniziare.

Alla ricezione dell'evento `gameCanStart`, tutti i client passano alla schermata di selezione della squadra, dove ogni giocatore sceglie tra i team `Squadra Gialla` e `Squadra Blu`. Quando il master emette l'evento `versusGameCanStart`, il server verifica che i requisiti per l'avvio del gioco siano soddisfatti: tutti i giocatori devono aver selezionato una squadra e ciascun team deve avere almeno un membro.

Se le condizioni sono rispettate, il server crea una nuova istanza di `VersusGame` e comunica a tutti i client della lobby che il gioco può iniziare. Alla ricezione dell'evento `versusGameCanStart`, tutti i client passano alla schermata di gioco.

- Dinamica di gioco Multiplayer

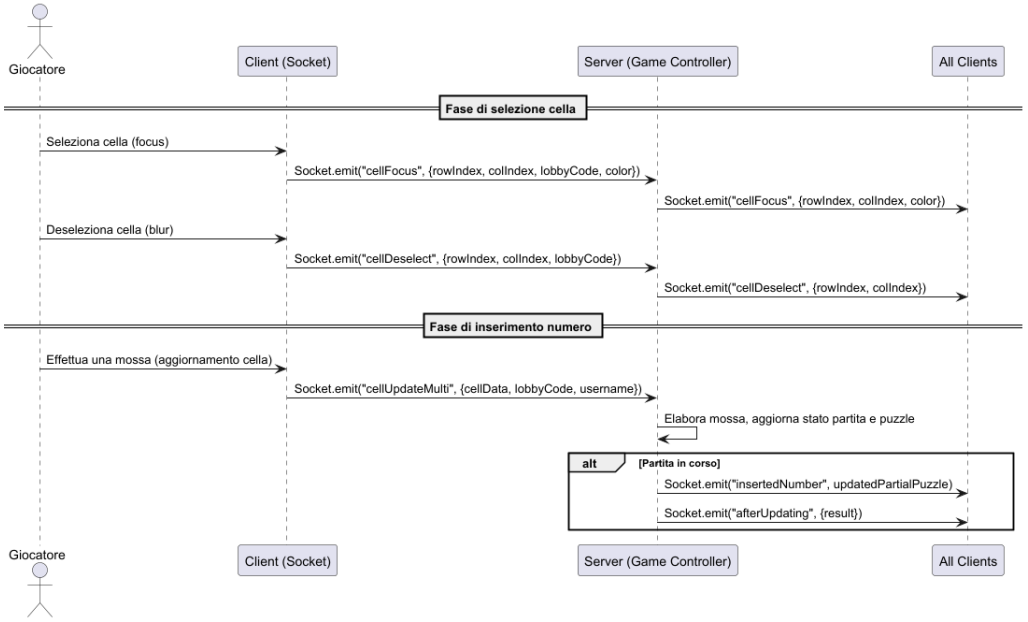

Figure 16: Dinamica di gioco in modalità Multiplayer (Coop/Versus).

Quando un giocatore seleziona o deseleziona una cella, emette rispettivamente l'evento `cellFocus` o `cellDeselect` al server, il quale propaga l'evento a tutti gli altri client nella lobby. Alla ricezione, ciascun client aggiorna la propria interfaccia colorando o decolorando in maniera opportuna la cella indicata.

Quando un giocatore inserisce un numero in una cella, emette l'evento `cellUpdateMulti`, specificando la cella interessata. Il *server* esegue quindi le seguenti operazioni:

- Emette l'evento `insertedNumber`, che ha l'unico scopo di notificare a tutti i client il valore inserito, in modo che venga visualizzato correttamente nelle loro interfacce.
- Chiama il metodo `insertNumber` della classe corrispondente alla modalità di gioco in corso (`CoopGame` o `VersusGame`) e invia a tutti i client la risposta. Questa include informazioni riguardanti la correttezza del numero inserito e lo stato della partita, come:
  - il sudoku dopo l'inserimento del valore (`puzzle`);
  - un messaggio che indica l'esito dell'inserimento;
  - il termine della partita (`gameOver`);
  - la soluzione in caso di sconfitta;
  - il numero di vite rimanenti in `CoopGame`;
  - i punti accumulati e i giocatori delle squadre in `VersusGame`.

Sulla base delle informazioni ricevute, ogni client aggiorna di conseguenza la propria interfaccia grafica.

Dinamica di fine gioco Multiplayer

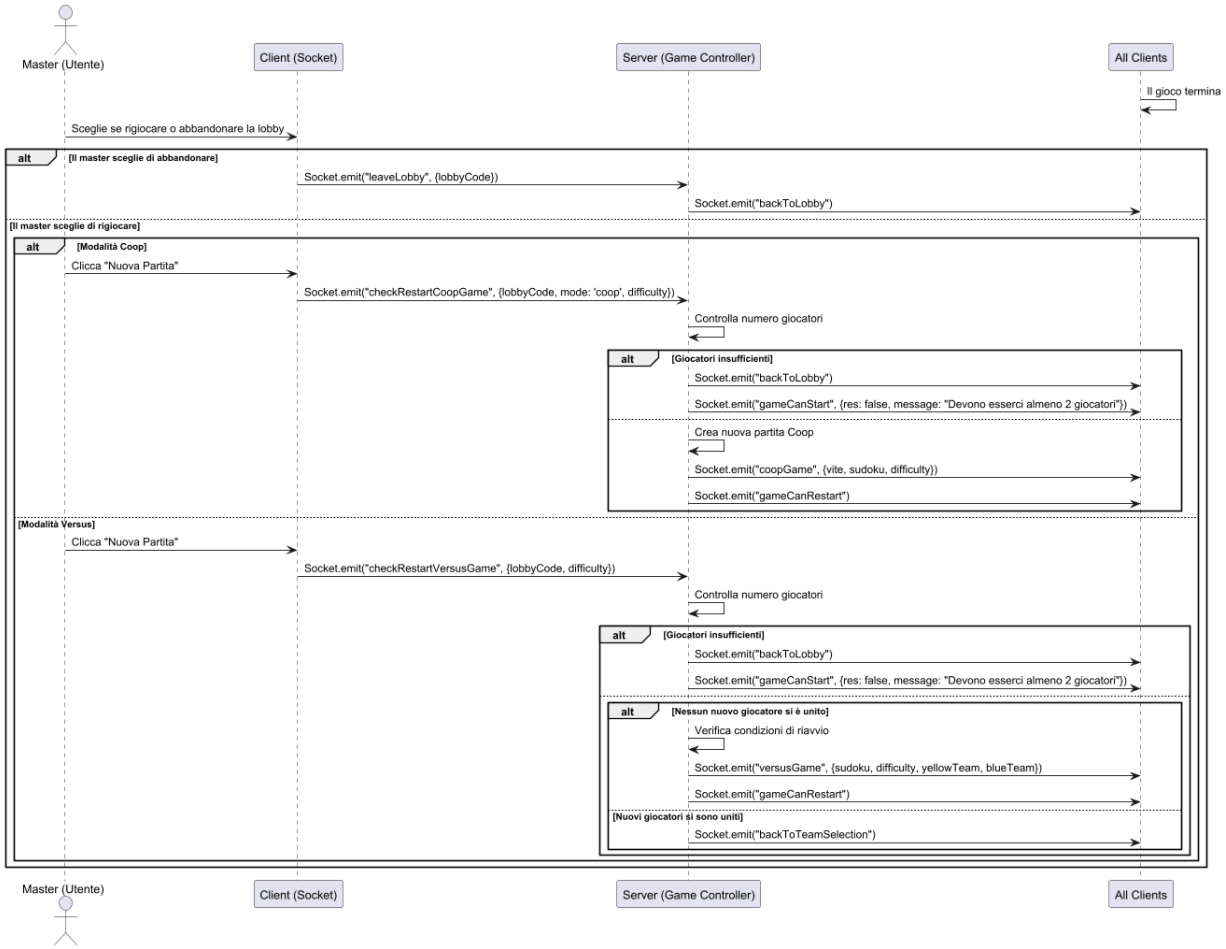

Figure 17: Dinamica di fine gioco in modalità Multiplayer (Coop/Versus).

Alla conclusione di una partita multiplayer, il sistema notifica tutti i client della lobby riguardo al termine del gioco. A questo punto, il **Master** ha la possibilità di scegliere se **tornare alla lobby** o **iniziarne una nuova**.

Se il Master sceglie di tornare alla lobby, tramite il server lo comunica a tutti i client, dunque questi mostrano la schermata della lobby.

Se il Master sceglie di rigiocare, in base alla modalità di gioco corrente, si osservano diverse condizioni.

- Modalità Coop

Quando il Master preme il pulsante "Nuova Partita", emette l'evento `checkRestartCoopGame`, in modo che il server possa controllare il numero di giocatori in lobby:

- Se c'è un unico giocatore, il server risponde con l'evento `backToLobby` e lo riporta in lobby avvisandolo che ci devono essere almeno due giocatori per poter iniziare una nuova partita.
- Se il numero di giocatori è sufficiente, il server crea una nuova istanza di gioco e invia ai client l'evento `coopGame`, contenente le configurazioni necessarie per avviare una nuova partita. Successivamente, il server emette l'evento `gameCanRestart`, permettendo ai client di reimpostare i valori che erano stati modificati nel match precedente, così da rappresentare correttamente il nuovo sudoku.

- Modalità Versus

Quando il Master preme il pulsante "Nuova Partita", emette l'evento `checkRestartVersusGame`, in modo che il server possa controllare il numero di giocatori in lobby:

- Come nel caso Coop, se c'è un unico giocatore, il server risponde con l'evento `backToLobby` e lo riporta in lobby avvisandolo che ci devono essere almeno due giocatori per poter iniziare una nuova partita.
- Se il numero di giocatori è sufficiente, il server verifica se si sono aggiunti nuovi giocatori rispetto alla partita precedente:
  - Se **nessun nuovo giocatore** si è unito, il server ricrea la partita e invia ai client l'evento `versusGame`, con le informazioni necessarie per configurare la nuova partita. Emette, come nel CoopGame, anche l'evento `gameCanRestart` per confermare l'inizio del nuovo match.
  - Se **si sono aggiunti nuovi giocatori**, la composizione dei team deve essere aggiornata. Il server emette l'evento `backToTeamSelection`, in modo da riportare i giocatori alla schermata di selezione della squadra.

4 Implementation Details

In questa sezione si riportan alcuni dettagli implementativi degni di nota, focalizzati sulla distribuzione del sistema e la comunicazione real-time.

- **Stack MEVN**: Node.js (Express) + MongoDB (Mongoose) + Vue.js + Socket.IO.
- **Docker Compose**: permette di avviare tre container (server, client, database) in un unico comando.
- **Server Socket.IO**: definito in `server/index.js`, gestisce la creazione e gestione delle stanze, i comandi di chat e le partite in tempo reale multiplayer (focus celle, inserimento numeri, fine partita).

- **Persistenza**: ogni partita single-player completata aggiorna le statistiche su MongoDB.
- **Sicurezza**: Password hash tramite `bcrypt`. Lato socket, l'utente deve essere autenticato prima di emettere eventi.

5 Self-assessment / Validation

Test automatici

All'interno della directory `/server/tests` e relative sottocartelle, sono stati creati dei test su diverse componenti critiche:

- **Controller di gioco e di lobby**: testano la corretta creazione e gestione di partite singleplayer e multiplayer, la generazione dei codici univoci di lobby, le condizioni di vittoria/sconfitta e l'aggiornamento delle statistiche.
- **Modelli come `GameWithLives`, `VersusGame`, `CoopGame`**: testano l'inserimento di numeri corretti/errati, la decrementazione delle vite, la gestione dei punti in `VersusGame` e l'eliminazione dei giocatori in caso di errore.
- **UserController e servizi di login/registrazione**: verificano che non si possa registrare due volte lo stesso utente, che un utente esistente possa fare login solo con password corretta, ecc.

Tali test sono stati realizzati con **Jest** [**5**] (per simulare le richieste HTTP verso le rotte REST).

Test con utenti reali

Oltre ai test automatici, sono stati condotte numerose prove manuali con:

- **Ambiente locale**: per verificare il funzionamento in modalità sviluppo e la corretta integrazione di Socket.io, testando la presenza di ritardi e problemi di sincronizzazione.
- **Utenti esterni (amici e parenti)**: li abbiamo invitati a registrarsi, provare la lobby, creare partite in modalità Coop e Versus, e darci feedback sulle prestazioni e l'usabilità. Queste prove "reali" hanno permesso di:
  - verificare la chiarezza delle interfacce;
  - rilevare eventuali bug di disconnessione o edge case (es. abbandono del *master*);
  - ricevere suggerimenti di miglioramento, come rendere più evidente la selezione delle celle e la distinzione cromatica tra le squadre.

6 Deployment Instructions

Esecuzione in locale (Docker)

È possibile eseguire Sudokoop in locale tramite Docker seguendo questi passaggi:

1. Clonare il repository:

   ```
   git clone https://dvcs.apice.unibo.it/pika-lab/courses/ds/projects/
        ds-project-mastrilli-siroli-ay2223.git
   cd ds-project-mastrilli-siroli-ay2223/sudokoop
   ```

2. Avviare Docker Compose:

   ```
   docker-compose up --build
   ```

   Questo comando crea e avvia i container seguenti:

   - `mongo`: database MongoDB su porta `27017`
   - `sudokoop-server` (Node/Express) su porta `5001`
   - `sudokoop-client` (Vue) su porta `8080`

3. Aprire il browser su: `http://localhost:8080`

7 Usage Examples

Di seguito alcuni casi d'uso pratici.

Esempio 1: Single-Player

1. L'utente effettua login, seleziona la difficoltà e preme `Inizia`.
2. Riceve dal server un sudoku.
3. Il client mostra il sudoku con 3 vite. Ad ogni errore, la cella si colora di rosso e una vita viene persa; se il numero è corretto, la cella diventa verde.
4. Completato il sudoku con una vittoria, il tempo di gioco viene inviato al server e salvato nella leaderboard.

Figure 18: Single Player su PC e Smartphone.

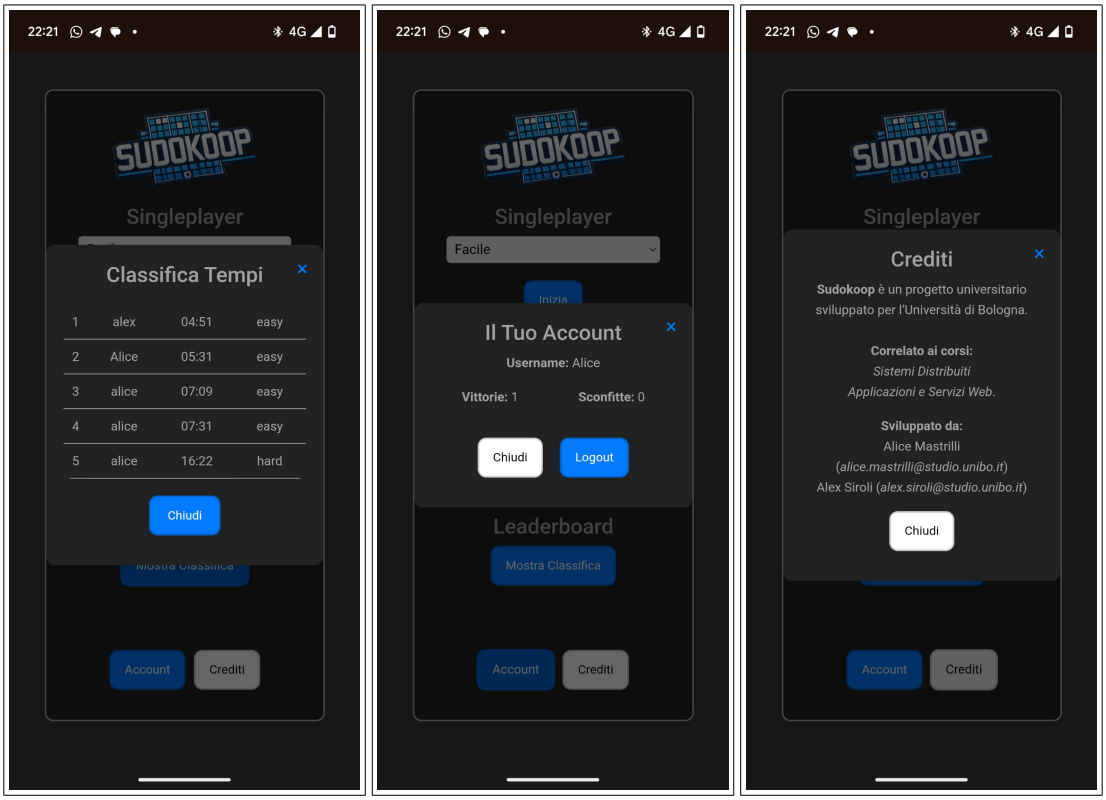

Figure 19: Leaderboard, Account e Credits su Smartphone.

Esempio 2: Creazione Lobby e Modalità Coop

1. Un utente loggato preme `Crea Lobby`, e il server risponde con `lobbyCode` (es. ABC123). Egli sarà il master.
2. Altri utenti possono inserire il codice e unirsi a quella lobby. Vengono mostrati i nomi dei giocatori collegati.
3. Il master sceglie la modalità *Coop* e la difficoltà, poi il server verifica che ci siano almeno 2 giocatori.
4. Viene creato un Sudoku comune con 3 vite totali e tutti lo vedono.
5. I giocatori collaborano inserendo numeri. Se un giocatore sbaglia, le vite comuni diminuiscono.

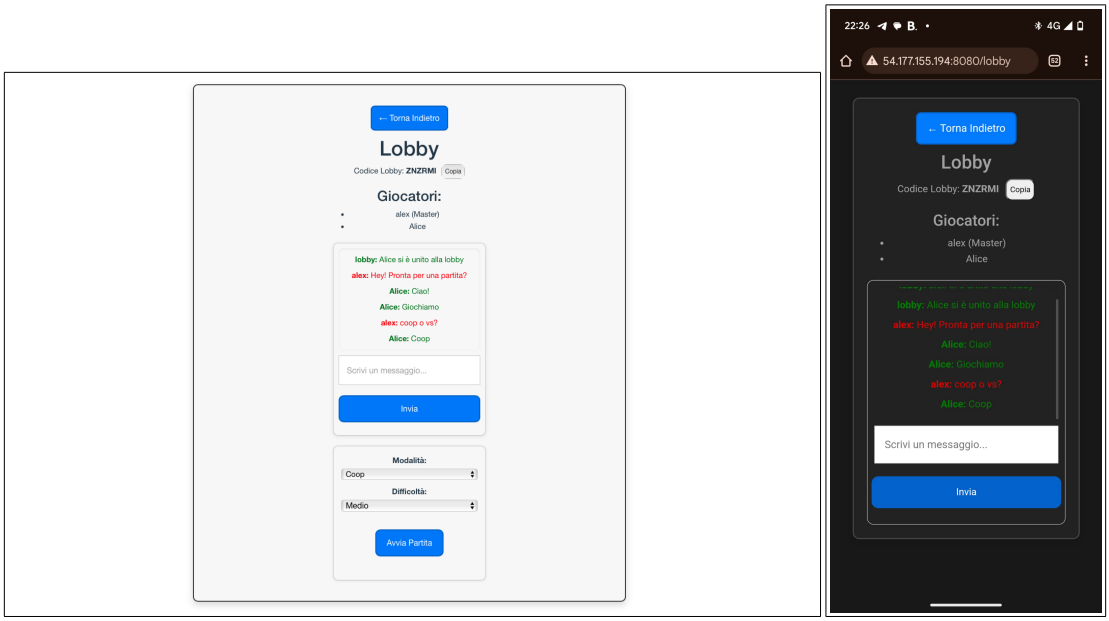

Figure 20: Lobby su PC e Smartphone.

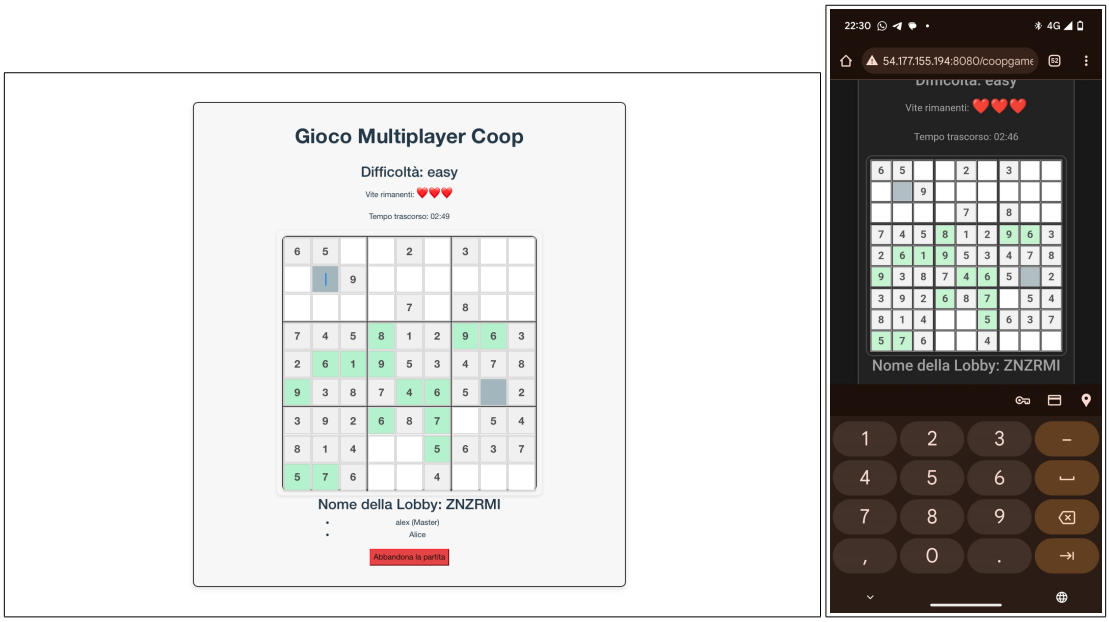

Figure 21: Modalità Cooperativa su PC e Smartphone.

Esempio 3: VersusGame

1. Nella lobby, il master seleziona *Versus* e la difficoltà, e ogni utente sceglie una squadra (Blu o Gialla).
2. Il server crea un Sudoku comune e ogni mossa corretta assegna un punto alla squadra.
3. Se un utente sbaglia numero, viene eliminato e non può più inserire cifre.
4. Il gioco termina o per completamento del Sudoku o per eliminazione di tutti i membri di una squadra.

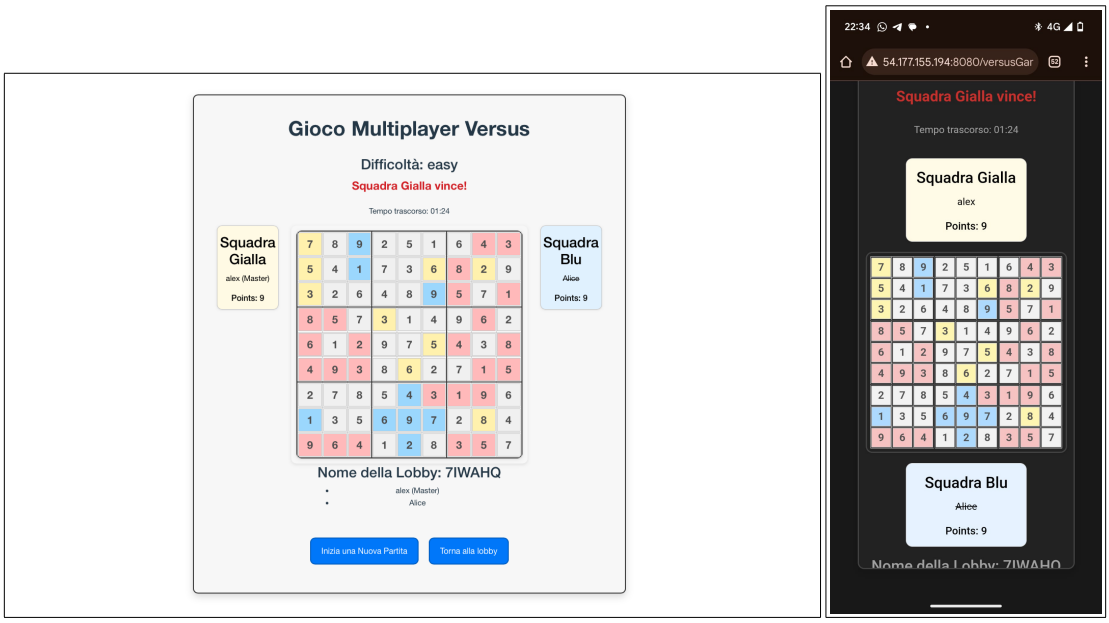

Figure 22: Modalità Competitiva su PC e Smartphone.

8 Conclusions

Sudokoop è un progetto nato per esplorare le potenzialità dei Sistemi Distribuiti applicati a un gioco da tavolo come il sudoku. L'impiego di Socket.IO, Docker e MongoDB ha consentito di realizzare un ambiente di gioco real-time, scalabile e facile da distribuire.

Nel corso del progetto si è approfondita la gestione delle comunicazioni asincrone tra client e server, il coordinamento di più client in tempo reale e la persistenza dei dati di gioco. Le sessioni di test con utenti reali hanno dimostrato la robustezza del sistema, offrendo al contempo spunti di miglioramento.

8.1 Future Works

Tra i possibili sviluppi futuri, abbiamo pensato:

- Evidenziare in modo più chiaro la selezione di una cella da parte di un utente, non solo colorandola, ma anche mostrando il nome dell'utente che l'ha selezionata.
- Introdurre nuove modalità di gioco per offrire un'esperienza più innovativa e differenziata rispetto ai classici sudoku online.
- Implementare un bot, in modo che i giocatori possano sfidarlo nella risoluzione dei sudoku.
- Creare una sezione dedicata ai `tornei`, permettendo sfide 1vs1 o a squadre tra i partecipanti, senza la necessità di accedere a una lobby specifica e senza limiti di giocatori.
- Rendere l'esperienza dell'utente più personalizzabile, ad esempio consentendogli di modificare il proprio username dopo la registrazione o di scegliere un colore personalizzato.

8.2 What did we learned

Questo progetto ci ha permesso di sperimentare in modo pratico l'architettura client-server, la sincronizzazione real-time tramite Socket.IO e la persistenza su MongoDB. Abbiamo imparato a progettare e gestire la concorrenza e le problematiche tipiche di un ambiente distribuito (come la gestione dei tempi di latenza e la necessità di un robusto meccanismo di sessione). L'utilizzo di Docker ci ha consentito di facilitare notevolmente il deployment.

References

[1] Node.js: `https://nodejs.org/en`

[2] Socket.IO: `https://socket.io/docs/v4`

[3] Docker: `https://www.docker.com`

[4] Mongoose: `https://mongoosejs.com`

[5] Jest: `https://jestjs.io`