Relay Pong, a Pong-inspired web application

Jahrim Gabriele Cesario
jahrim.cesario2@studio.unibo.it

Settembre 2022

Questo progetto verte sullo sviluppo di un'applicazione web basata su una comunicazione di tipo real-time. In particolare, si vuole realizzare una variante del videogioco *Pong* sotto forma di servizio web, quindi accessibile tramite browser, che permetta a più giocatori di competere tra di loro.

Il gioco è una simulazione del ping pong. In ogni partita competono due giocatori, a cui viene assegnata una barra ciascuno, utilizzata per colpire una pallina. Quando la pallina oltrepassa la barra dell'avversario, si guadagna un punto. Lo scopo del gioco è di ottenere un certo numero di punti, prima del proprio avversario.

Nella variante proposta, chiamata *Relay Pong* oppure *Pong a staffetta*, il gioco non è più limitato a due giocatori. In ogni partita competono infatti due squadre, composte da almeno un giocatore, ognuna delle quali controlla una barra. I giocatori della stessa squadra si alternano a controllare la barra della squadra durante lo sviluppo della partita. Le condizioni per vincere una partita rimangono le medesime.

Lo scopo iniziale di questo progetto è quello di realizzare una proof of concept del servizio descritto, quindi un prototipo che mostri la fattibilità della sua realizzazione.

Il servizio si baserà su un'architettura di tipo client-server, in cui il server mantiene l'originale dello stato della partita e lo condivide con i client, mentre i client inviano al server degli eventi, che rappresentano le intenzioni dei giocatori, da cui dipenderanno gli aggiornamenti sullo stato della partita. Per la comunicazione di tipo real-time, si sperimenterà con l'uso del protocollo WebSocket, che, rispetto ad altri protocolli supportati dai browser, privilegia una comunicazione bidirezionale con basso overhead dei pacchetti. Altri punti interessanti saranno la progettazione del modello logico usato per rappresentare il gioco, la progettazione delle dinamiche che regolano lo svolgimento di una partita e la gestione della sincronizzazione dei giocatori con il server.

Sommario

1 Introduzione

Questo progetto è incentrato sulla realizzazione di una variante del gioco *Pong*[18]: un videogioco ispirato al ping-pong, realizzato negli anni '70.

Il gioco, illustrato in Figura 1, può essere giocato da un massimo di due giocatori, ognuno dei quali controlla una racchetta, generalmente rappresentata come una barra. Nel caso in cui fosse presente un solo giocatore, l'altra racchetta può essere controllata dal computer. Durante il gioco, il giocatore deve colpire una pallina, facendola rimbalzare verso il proprio avversario. Se la pallina oltrepassa la barra dell'avversario, si guadagna un punto. Lo scopo del gioco è di ottenere un certo numero di punti, prima del proprio avversario.

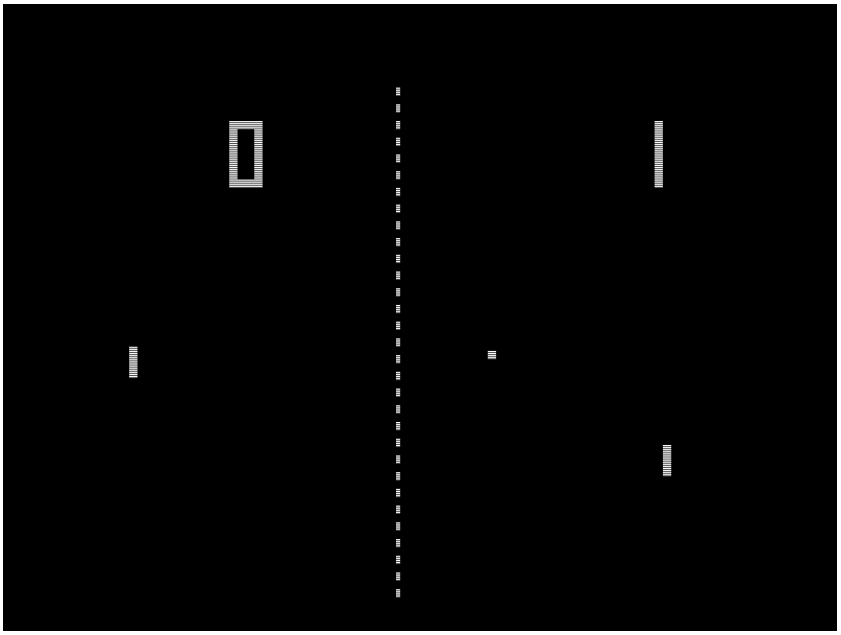


Figure 1: Screenshot rappresentativo di una delle implementazioni del gioco Pong.

La variante che si vuole realizzare è stata battezzata come *Relay Pong*, oppure *Pong a staffetta*, ed è stata pensata per estendere il videogioco *Pong* a più di due giocatori.

Nel *Relay Pong* giocano due squadre, composte da almeno un giocatore, ognuna delle quali controlla una delle barre. I giocatori della stessa squadra giocano a turno, alternandosi a controllare la barra della squadra. In particolare, quando un giocatore colpisce la pallina, cede il controllo della propria barra al prossimo giocatore della sua squadra. Lo scopo del gioco è sempre quello di ottenere un certo numero di punti, prima della squadra avversaria.

Il gioco sarà realizzato come applicazione web, quindi accessibile da browser, sia per dispositivi desktop che mobili, in modo che sia disponibile al maggior numero di persone possibile. Di conseguenza, sarà anche necessario progettare il sistema per essere facilmente scalabile.

L'applicazione dovrà fornire un servizio di hosting per le partite, quindi permettendo agli utenti di creare e configurare dei server di gioco identificabili. Conoscendo l'identificatore di un server, un utente potrà partecipare alla partita gestita da quel server.

Lo scopo di questo progetto sarà produrre un prototipo di tale sistema, focalizzandosi sull'implementazione di solo alcune delle sue funzionalità, privilegiando quelle relative alla comunicazione real-time. In particolare, ci si concentrerà sulla produzione di una demo funzionante del gioco, piuttosto che sull'implementazione del servizio di hosting dei server di gioco, anche se le scelte di design del prototipo saranno comunque influenzate dai requisiti che dovranno essere soddisfatti dal sistema una volta ultimato.

2 Analisi del problema

In questo capitolo, si analizzeranno i requisiti del sistema, ovvero le funzionalità che il sistema dovrà offrire una volta completato questo progetto.

I requisiti del sistema saranno analizzati sotto due diverse prospettive, quindi si distingueranno:

- Requisiti prototipali: le funzionalità che dovranno essere implementate dal prototipo del sistema, quindi quelle su cui verterà questo progetto.
- Requisiti futuri: le funzionalità che dovranno essere implementate dal sistema una volta ultimato, quindi quelle su cui verteranno progetti futuri.

I requisiti futuri permetteranno di contestualizzare il prototipo, anticipando i cambiamenti che dovrà subire e quindi giustificando alcune scelte di design.

2.1 Requisiti prototipali

Di seguito, si illustreranno i requisiti del prototipo del sistema. Inoltre, si riporterà un esempio di caso d'uso dell'applicazione da parte dell'utente finale.

2.1.1 Requisiti funzionali

Il prototipo del sistema dovrà offrire le seguenti funzionalità:

- **Participazione a una partita**: l'utente dovrà poter partecipare a una partita di *Relay Pong*. Una partita deve poter accettare la partecipazione di almeno due giocatori;
- **Hosting di una partita**: il primo utente che parteciperà a una partita dovrà ricevere il ruolo di host di quella partita, quindi si dice *ospiterà* tale partita;
- **Visualizzazione della configurazione di una partita**: partecipando a una partita, l'utente dovrà poter visualizzare il proprio nome all'interno della partita, il numero massimo di giocatori che possono partecipare alla partita e il numero massimo di set che potranno essere giocati all'interno della partita;
- **Inizio di una partita**: gli utenti dovranno poter decidere quando cominciare una partita. Una volta che tutti gli utenti avranno espresso la loro intenzione di iniziare a giocare, l'host potrà effettivamente cominciare la partita;
- **Visualizzazione dello stato di una partita in corso**: una volta cominciata la partita, gli utenti dovranno poter visualizzare la posizione dei componenti del gioco, i punteggi delle due squadre, i nomi dei giocatori attualmente in controllo delle barre e di quelli a cui il controllo sarà ceduto successivamente;
- **Controllo della barra della propria squadra**: l'utente dovrà poter muovere la barra della propria squadra quando ne ha il controllo;

- **Cessione del controllo della barra di una squadra**: se una squadra ha almeno due giocatori, l'utente di quella squadra che ha il controllo della barra, prima o poi dovrà cedere il controllo al giocatore successivo;
- **Termine della partita**: la partita dovrà terminare quando una delle due squadre si è aggiudicata la maggioranza dei set;
- **Visualizzazione del risultato di una partita**: una volta terminata la partita, l'utente dovrà essere notificato dell'eventuale vittoria, sconfitta o pareggio, conseguito dalla propria squadra;
- **Gestione del movimento delle barre**: cominciata una partita, una barra dovrà potersi muovere verticalmente, verso l'alto o verso il basso, e dovrà potersi fermare, in base alle intenzioni dell'utente che ne ha il controllo;
- **Gestione del movimento della pallina**: cominciata una partita, la pallina dovrà potersi muovere all'interno dell'area di gioco. All'utente dev'essere dato modo di controllare la direzione della partita quando la colpisce con la barra della propria squadra;
- **Gestione delle collisioni**: cominciata una partita, dovranno essere gestite le collisioni tra la pallina ed il perimetro di gioco e tra la pallina e le barre delle due squadre. Inoltre, ogni componente del gioco dovrà sempre essere confinato all'interno del perimetro di gioco;
- **Gestione dei punteggi delle squadre**: cominciata una partita, quando la pallina supera la barra della squadra avversaria, la propria squadra deve ricevere un punto.

2.1.2 Requisiti non funzionali

Il prototipo del sistema dovrà soddisfare le seguenti proprietà:

- **Accesso tramite desktop**: il servizio dovrà essere accessibile tramite dispositivi desktop;
- **Compatibilità con browser diversi**: il servizio dovrà essere accessibile tramite browser diversi, tra quelli più utilizzati;
- **Tolleranza alle disconnessioni**: il sistema dovrà reagire alle disconnessioni degli utenti sia prima che durante lo svolgimento della partita, terminando la partita solo quando strettamente necessario;
- **Reattività alle richieste dell'utente**: il sistema deve essere sufficientemente reattivo da permettere agli utenti di interagire efficacemente con il gioco durante una partita;

- **Fluidità nella visualizzazione dello stato di una partita**: il sistema dovrà essere sufficientemente fluido da permettere agli utenti di comprendere facilmente come sta evolvendo lo stato del gioco durante una partita;
- **Continuità dell'applicazione**: il sistema non dovrà terminare quando è terminata una partita, ma dovrà permettere di partecipare a una nuova partita.

2.1.3 Scenarios

Per accedere al prototipo del sistema, l'utente necessiterà di un browser, attraverso il quale potrà collegarsi alla schermata principale del servizio. Dalla schermata principale, l'utente potrà quindi accedere alle funzionalità del sistema. In particolare, gli sarà permesso di partecipare a una partita di *Relay Pong*.

Partecipando a una partita, l'utente sarà assegnato automaticamente a una delle due squadre del gioco. Inoltre, se l'utente è il primo a partecipare alla partita, ne diventerà l'host. In tal caso, dovrà aspettare che altri utenti si connettano al servizio e partecipino alla partita. Quando ogni squadra ha almeno un giocatore e tutti i partecipanti sono pronti per iniziare la partita, l'host potrà cominciare la partita.

Durante la partita, l'utente potrà controllare la barra della propria squadra, solo quando è il suo turno. L'obiettivo dell'utente dovrà essere quello di intercettare la pallina, muovendo la barra della propria squadra verso l'alto o verso il basso, oppure fermandola in una certa posizione. Allo stesso tempo, l'utente dovrà cercare di direzionare la pallina, in modo che oltrepassi la barra della squadra avversaria, segnando un punto per la propria squadra.

Al termine della partita, quando una delle due squadre si è aggiudicata la maggioranza dei set, all'utente dev'essere mostrato il risultato della partita. Infine, l'utente sarà riportato alla schermata principale del servizio.

2.2 Requisiti futuri

Di seguito, si illustreranno i requisiti che il sistema dovrà soddisfare una volta ultimato. Questi requisiti non saranno oggetto di questo progetto, ma saranno considerati per compiere alcune scelte di design ed implementazione del prototipo.

2.2.1 Requisiti funzionali

Il sistema, una volta ultimato, dovrà offrire le seguenti funzionalità:

- **Creazione di una partita**: l'utente dovrà poter istanziare un server di gioco per gestire una nuova partita di cui sarà l'host;
- **Configurazione di una partita**: durante la creazione di una partita, l'utente dovrà poterla configurare definendo l'identificatore della partita, il numero massimo di giocatori che possono partecipare alla partita ed il numero massimo di set che possono essere giocati all'interno della partita. Altre configurazioni disponibili possono riguardare la visibilità della partita (pubblica o privata);

- **Partecipazione a una partita tramite identificatore**: l'utente dovrà poter partecipare a una partita pubblica o privata conoscendone l'identificatore;
- **Partecipazione a una partita tramite ricerca**: l'utente dovrà poter ricercare una partita tra quelle pubbliche e quindi selezionarla per partecipare a tale partita;
- **Scelta della squadra di appartenenza**: partecipando a una partita, l'utente dovrà poter scegliere la squadra a cui essere assegnato;
- **Gestione del profilo utente**: all'utente dovrà essere permesso di gestire un proprio profilo, in cui potrà specificare il nome con il quale sarà riconoscibile all'interno delle partite in cui gioca;
- **Modalità giocatore singolo**: all'utente dovrà essere permesso di creare una partita in cui giocherà da solo contro il computer;
- **Modalità multigiocatore hotseat**: all'utente dovrà essere permesso di creare una partita in cui giocherà contro un altro giocatore attraverso lo stesso dispositivo.

2.2.2 Requisiti non funzionali

Il sistema, una volta ultimato, dovrà soddisfare le seguenti proprietà:

- **Accesso tramite smartphone**: il servizio dovrà essere accessibile sia tramite desktop che tramite smartphone;
- **Scalabilità del sistema**: il sistema dovrà essere sufficientemente scalabile, in modo da poter gestire efficacemente la variazione della richiesta del servizio offerto;
- **Modularità del sistema**: il sistema dovrà essere sufficientemente modulare, in modo da poter scalare solo una specifica funzionalità del servizio quando richiesta, invece che l'intero servizio.

3 Design

In questo capitolo, si descriverà come è stato progettato il sistema, con l'intenzione di soddisfare i requisiti prototipali introdotti in fase di analisi. In particolare, si introdurranno i componenti del sistema ad alto livello, descrivendone per ognuno lo scopo, le caratteristiche e le interazioni con gli altri componenti. Infine, si scenderà più nel dettaglio, analizzando le tecnologie scelte per realizzare le varie componenti ed avvicinandosi alla loro implementazione.

3.1 Design architetturale

In questa sezione, si individueranno i componenti principali e le interazioni che caratterizzano il sistema.

3.1.1 Componenti del sistema

Durante la progettazione, si è cercato di suddividere il sistema in componenti ben distinti in base alle loro funzionalità. In particolare, ci si riferirà ai componenti che espongono le funzionalità proprie del sistema anche con il nome di *servizi*.

Di seguito, si riportano i componenti principali che caratterizzano il sistema.

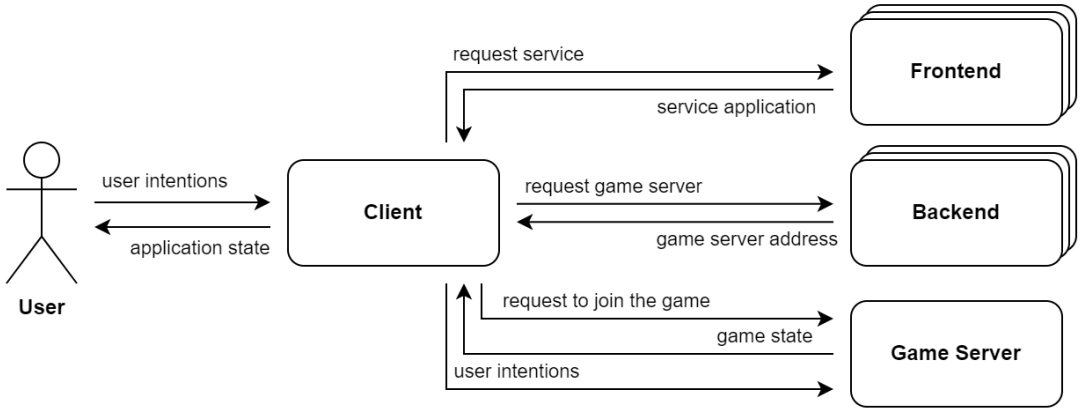


Figure 2: Diagramma che rappresenta i componenti del sistema ad alto livello e le interazioni tra di essi.

Come illustrato in Figura 2, il sistema è caratterizzato da quattro componenti:

- **Client**: gestisce le interazioni dell'utente con il sistema. In particolare, interpreta le intenzioni dell'utente, traducendole in messaggi verso i vari componenti del sistema, e notifica l'utente dei cambiamenti avvenuti all'interno dell'applicazione;
- **Frontend**: gestisce e vincola il flusso di interazioni dell'utente con il sistema. In particolare, fornisce un'applicazione utilizzabile dal Client per interagire con il sistema e per mostrare all'utente lo stato del servizio o del gioco in corso. D'ora in

avanti, quando si parlerà di "applicazione", si intenderà l'applicazione fornita da questo componente;

- **Backend**: gestisce la logica di ricerca dei server di gioco. In particolare, risponde alle richieste dell'utente, ricercando la partita a cui l'utente vuole partecipare, quindi restituendo l'indirizzo del server di gioco che gestisce tale partita;

- **Game Server**: gestisce la creazione e lo sviluppo di una specifica partita di *Relay Pong*. In particolare, risponde alla richieste di partecipazione alla partita che gestisce, inviando periodicamente lo stato del gioco ai suoi partecipanti e aggiornandolo in base alle intenzioni da loro ricevute. D'ora in avanti, quando si parlerà di "server di gioco", si intenderà un'istanza di questo componente.

Questa suddivisione del sistema permette ad ogni componente di essere facilmente scalabile, ad eccezione dei Game Server. Infatti, il Frontend ed il Backend sono due servizi *stateless* [9], per cui all'utente non interessa a quale replica siano inoltrate le sue richieste; al contrario, il Game Server è un servizio *stateful*, per cui all'utente interessa comunicare con una replica precisa del servizio, in particolare quella che gestisce la partita a cui l'utente vuole partecipare.

Dopo aver individuato le componenti principali del sistema, è necessario progettare le interazioni che ne permettono un corretto funzionamento.

3.1.2 Interazioni del sistema

Per la progettazione delle interazioni tra i componenti, si è adottato un approccio di tipo *Outside-In*, quindi si è partiti dalle interazioni che l'utente finale vorrebbe poter avere con il sistema.

Come si può osservare in Figura 3, per permettere ad un utente di usufruire del servizio, è necessaria la seguente sequenza di interazioni:

1. Per prima cosa, l'utente deve connettersi al Frontend, così da ricevere l'applicazione che gli consentirà di comunicare con il sistema;

2. Ottenuta l'applicazione dal Frontend, all'utente viene mostrata la schermata principale dell'applicazione, dalla quale può richiedere di partecipare a una partita del sistema. Quando ciò accade, l'utente invia una richiesta al Backend per conoscere l'indirizzo del Game Server che ospita la partita a cui vuole partecipare, quindi si connette a quello specifico server di gioco;

3. Una volta connesso al server di gioco, all'utente viene mostrata la lobby della partita, dalla quale può avvertire il Game Server di essere pronto per iniziare la partita. Quando tutti i giocatori nella lobby sono pronti, l'host della partita può comandare al Game Server di cominciare la partita;

4. Cominciata la partita, all'utente viene mostrata la schermata di gioco, dalla quale può osservare lo stato della partita evolversi in tempo reale ed inviare i propri comandi di gioco al Game Server;

5. Terminata la partita, all'utente viene mostrato il risultato della partita, quindi può scegliere di ritornare alla schermata principale dell'applicazione, eventualmente partecipando a una nuova partita.

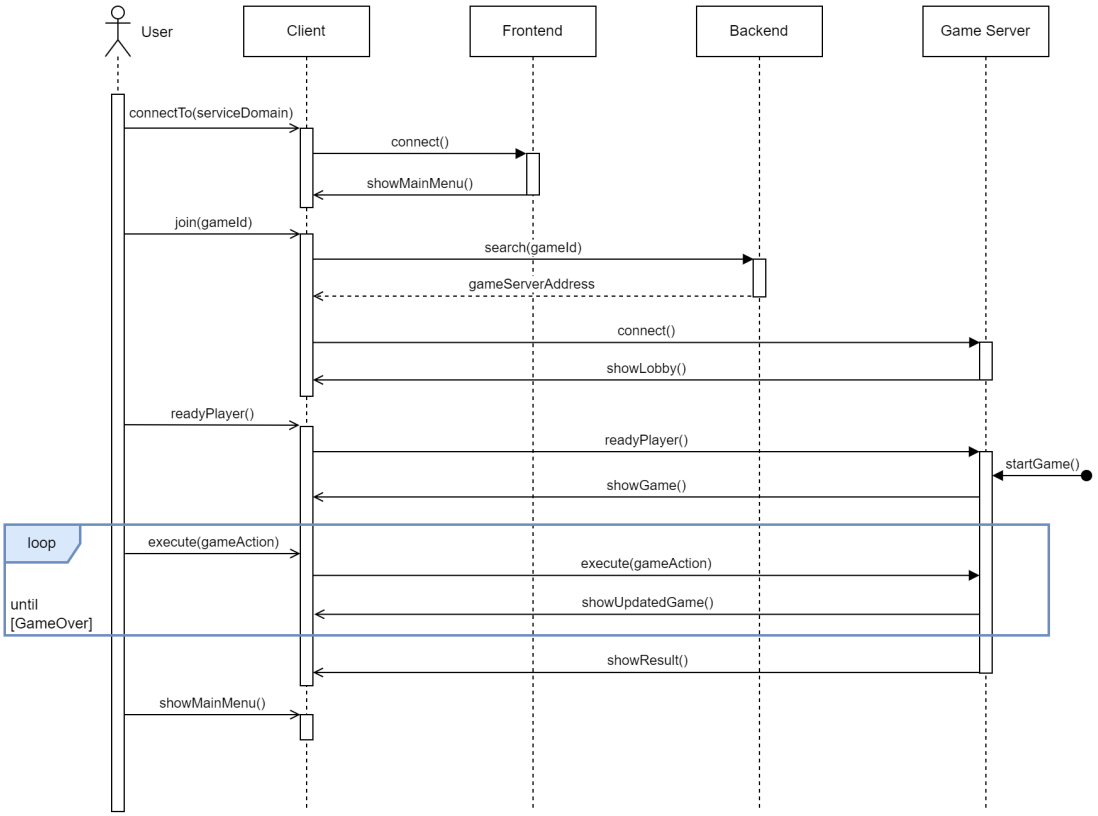


Figure 3: Diagramma di sequenza che descrive il flusso di interazioni che permette ad un utente di giocare ad una partita di *Relay Pong*.

Questo flusso di interazioni sarà la base utilizzata per progettare le interfacce dei vari servizi all'interno del sistema.

3.1.3 Frontend

Il Frontend è il servizio che fornisce all'utente l'applicazione per interagire con gli altri componenti del sistema.

L'interfaccia dell'applicazione deve prevedere almeno quattro schermate, come anticipato nella descrizione delle interazioni nel sistema:

- **Schermata principale**: la prima schermata dell'applicazione. Deve permettere all'utente di partecipare a una partita, connettendosi al server di gioco che la gestisce. Nel sistema ultimato, dovrà anche permettere all'utente di creare una nuova partita;

- **Schermata della lobby di una partita**: mostrata quando l'utente si è connesso con successo a uno specifico server di gioco. Deve permettere all'utente di visualizzare il proprio nome, il numero dei partecipanti alla partita, quanti di questi sono pronti a cominciarla, il numero massimo dei partecipanti alla partita ed il numero massimo di set che potranno essere giocati all'interno della partita. Inoltre, deve permettere all'utente di avvertire il server di gioco di essere o non essere pronto a giocare. Infine, deve permettere all'host di iniziare la partita quando tutti i partecipanti sono pronti;

- **Schermata di gioco**: mostrata quando comincia la partita a cui l'utente è connesso. Deve permettere all'utente di visualizzare lo stato della partita, ovvero la posizione della barre e della pallina, oltre che lo stato delle squadre, ovvero i loro punteggi, i giocatori che hanno attualmente il controllo delle barre di gioco e quelli a cui il controllo sarà ceduto successivamente;

- **Schermata del risultato della partita**: mostrata quando termina la partita a cui l'utente è connesso. Deve permettere all'utente di visualizzare se la propria squadra ha vinto, perso o pareggiato. Inoltre, deve permettere di ritornare alla schermata principale dell'applicazione.

3.1.4 Backend

Il Backend è il servizio che permette all'utente di ricercare una partita nel sistema e quindi di conoscere l'indirizzo del server di gioco che la gestisce.

Nella versione prototipale del sistema, non è prevista la gestione della creazione di nuovi server di gioco da parte dell'utente. Pertanto, il compito del Backend è più semplicemente quello di reindirizzare l'utente a uno dei server di gioco pre-esistenti.

L'interfaccia del Backend può essere descritta dal seguente contratto, che descrive l'unica funzionalità esposta dal servizio:

- **Ricerca di una partita tramite identificatore**: riceve una richiesta contenente l'identificatore di una partita, rispondendo con l'indirizzo del server di gioco che ospita la partita.

3.1.5 Game Server

Il Game Server è il servizio che ospita una specifica partita, permettendo a nuovi utenti di parteciparvi e gestendo i comandi ricevuti dai suoi partecipanti.

L'interfaccia del Game Server può essere descritta dal seguente contratto, che descrive le funzionalità esposte dal servizio:

- **Partecipazione alla partita ospitata**: riceve una richiesta da un utente specifico e registra tale utente come uno dei partecipanti alla partita ospitata. Inizialmente, all'utente appena registrato è associato uno stato, indicante che non è pronto a cominciare la partita. Inoltre, se l'utente è il primo partecipante, gli viene assegnato il ruolo di host della partita;

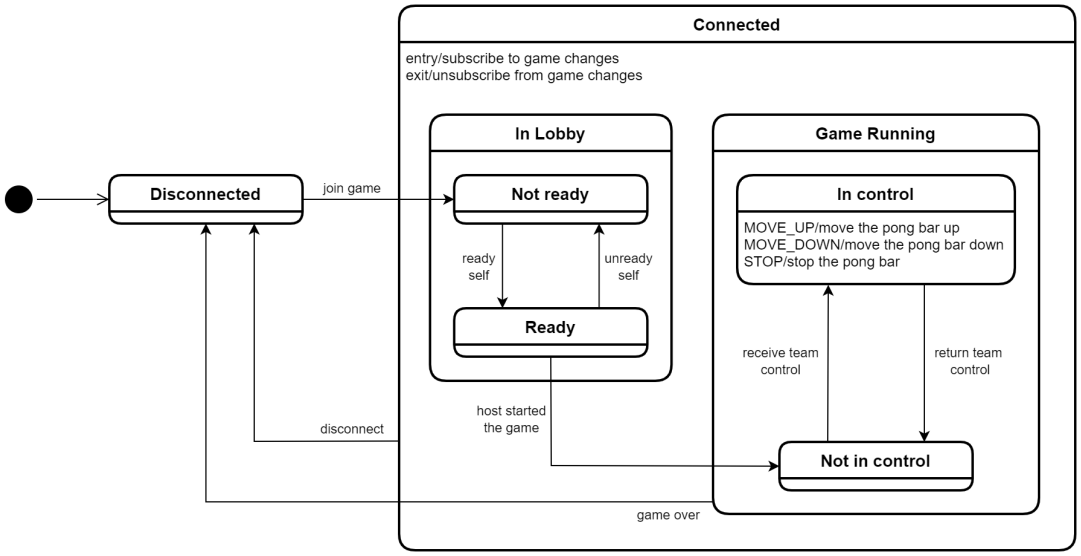
- **Aggiornamento dello stato di un utente**: riceve una richiesta da un utente specifico, contenente uno stato che descrive se l'utente è pronto o meno a cominciare la partita, quindi aggiorna lo stato di tale utente di conseguenza;
- **Inizio della partita ospitata**: riceve una richiesta dall'host della partita e comincia la partita se tutti i suoi partecipanti sono pronti a giocare;
- **Interpretazione dei comandi di gioco**: riceve una richiesta da un suo partecipante, contenente un comando che tale utente vuole eseguire nel gioco. I comandi supportati sono:
  - **Muovere una barra di gioco verso l'alto**: muove la barra della squadra a cui appartiene il giocatore verso l'alto, se ne ha il controllo;
  - **Muovere una barra di gioco verso il basso**: muove la barra della squadra a cui appartiene il giocatore verso il basso, se ne ha il controllo;
  - **Fermare una barra di gioco**: ferma la barra della squadra a cui appartiene il giocatore, se ne ha il controllo.
- **Ottenimento dello stato della partita**: riceve una richiesta da un utente, rispondendo con lo stato attuale della partita. Lo stato della partita deve contenere:
  - **Le informazioni relative alla sua configurazione**: il *massimo numero di partecipanti* e il *massimo numero di set* che possono essere giocati all'interno della partita;
  - **Le informazioni relative ai suoi partecipanti**: per ogni giocatore, il suo *nome*, il suo *stato* (pronto o non pronto), la *squadra* di appartenenza e il suo *ruolo* (host oppure ospite);
  - **Le informazioni relative alle due squadre**: il *punteggio* attuale, il *giocatore che ha il controllo* della barra di gioco ed il *giocatore a cui sarà ceduto il controllo* della barra di gioco;
  - **Le informazioni relative all'arena di gioco**: la *posizione delle barre* delle due squadre e la *posizione della pallina*.

Come già anticipato, il Game Server è un servizio *stateful*, pertanto le funzionalità che espone ad un certo utente cambiano in base allo stato in cui si trova.

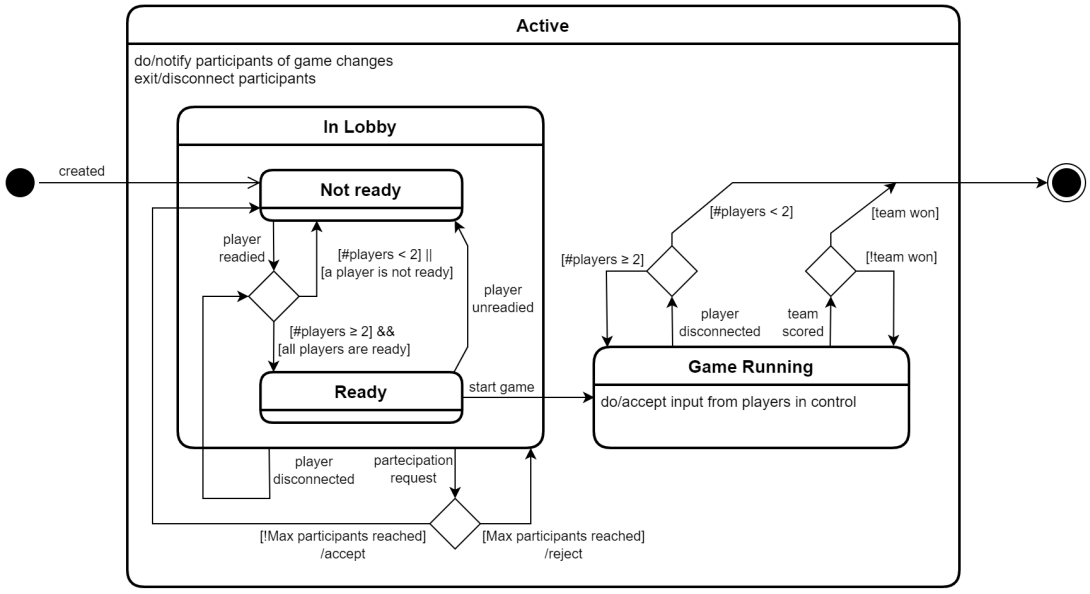
La Figura 4 evidenzia quali funzionalità sono disponibili all'utente durante il ciclo di vita del Game Server a cui è connesso, mostrando i possibili stati di un server di gioco, sia dal punto di vista di un utente ospite, ovvero non host, che dal punto di vista del server stesso.

Inizialmente, un utente non ancora connesso a un server di gioco può inviare una richiesta di *partecipazione a una partita* a uno dei server disponibili. Se il server non ha raggiunto la capacità massima di partecipanti, la richiesta viene accettata.

Partecipando a una partita, l'utente viene continuamente *notificato dello stato della partita* man mano che evolve. Inoltre, può *aggiornare il proprio stato*, indicando al Game Server di essere o non essere pronto a cominciare la partita.



(a) Punto di vista di un utente ospite (non host).



(b) Punto di vista del server stesso.

Figure 4: Diagramma a stati che descrive il ciclo di vita di un server di gioco.

Una volta che ci sono almeno due partecipanti e tutti i partecipanti sono pronti, nel caso in cui avesse il ruolo di host della partita, l'utente può *iniziare la partita*. Quando ciò accade, il Game Server permette agli utenti che hanno il controllo della barra della propria squadra di *inviare i comandi di gioco*, finché la partita non si conclude. Una

partita può concludersi se una delle due squadre si è aggiudicata la maggioranza dei set o se non ci sono più abbastanza giocatori per continuare a giocare. Infatti, in un qualsiasi momento, i giocatori possono disconnettersi dal server di gioco.

3.2 Design di dettaglio

In questa sezione, si introdurranno brevemente le tecnologie utilizzate in questo progetto, motivandone la scelta. Dopodiché, si approfondiranno il funzionamento e l'interfaccia dei vari componenti dei sistema, avvicinandosi alla loro implementazione.

3.2.1 Tecnologie adottate

Per questo progetto, sono state selezionate diverse tecnologie per la realizzazione dei componenti precedentemente descritti.

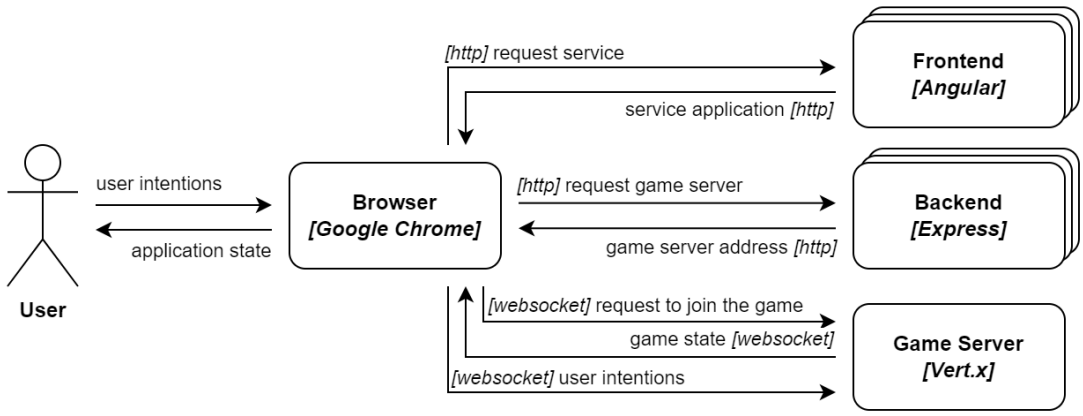


Figure 5: Diagramma che rappresenta le tecnologie utilizzate per realizzare i componenti del sistema e le interazioni tra di essi.

Come riportato in Figura 5, le tecnologie adottate per implementare i vari componenti del sistema sono:

- Per il Client, si è scelto di utilizzare un *browser*, come richiesto dai requisiti del prototipo, descritti in fase di analisi. In particolare, si è scelto **Google Chrome** [7] come browser di riferimento durante lo sviluppo;
- Per il Frontend, si è scelto di utilizzare il framework **Angular** [5], tramite il quale è possibile realizzare applicazioni web fluide in modo modulare, sotto forma di *Single Page Application* [19]. Angular permette anche di realizzare facilmente applicazioni basate su un'architettura *MVVM (Model View ViewModel)* [17], ideale per interfacce che rappresentano lo stato di un sistema, come in questo caso, in cui la schermata di gioco rappresenta lo stato della partita.
  L'applicazione web permette all'utente di connettersi ad un server di gioco tramite il protocollo *WebSocket* [21], in particolare attraverso la libreria *SockJS* [14];

- Per il Game Server, si è scelto di utilizzare il modulo web del framework **Vert.x** [3], che fornisce una api compatibile con SockJS. Questa scelta è anche dovuta alla complessità notevole del Game Server, che ha portato ad optare per un ambiente di lavoro più familiare, basato sul linguaggio di programmazione *Java* [15];

- Per il Backend, si è scelto di utilizzare il framework **Express** [4], che permette di realizzare applicazioni server in modo rapido. Come linguaggio di programmazione, è stato utilizzato *Typescript* [11].

Riguardo alle interazioni all'interno del sistema, il protocollo **WebSocket** è necessario per la comunicazione a bassa latenza tra l'utente ed il server di gioco, mentre il protocollo **HTTP** [20] risulta più che adatto per le altre interazioni.

Per la versione prototipale del sistema, differentemente da quanto descritto nella fase di design architetturale, si prevede un'unica istanza del servizio Game Server, quindi un unico server di gioco. In ogni caso, il Game Server dev'essere in grado di gestire almeno una partita e si prevede che il prototipo ospiti sempre e solo un'unica partita alla volta a cui gli utenti possono connettersi.

Tenendo presente le tecnologie che sono state utilizzate per realizzare ogni componente del sistema, è ora possibile descrivere più precisamente l'interfaccia di ogni servizio.

3.2.2 Frontend

Come già anticipato, l'interfaccia dell'applicazione fornita dal servizio di Frontend deve prevedere almeno quattro schermate. Tenendo presente che, almeno in fase prototipale, l'applicazione deve essere accessibile solo a dispositivi desktop tramite browser, le sue schermate sono state progettate seguendo un approccio di tipo *desktop-first*. Successivamente al completamento di questo progetto, sarà necessario riadattare queste schermate per essere supportate anche da dispositivi mobili, come invece richiesto dai requisiti del sistema ultimato.

Di seguito, si riportano degli schemi rappresentativi delle quattro schermate individuate nella fase di design architetturale, usati come riferimento durante l'implementazione:

- **Schermata principale** (Figura 6): espone all'utente le funzionalità principali dell'applicazione. Premendo il pulsante *Join Game* si tenta la connessione all'unico server di gioco presente nel sistema. Una volta avvenuta la connessione, si mostra all'utente la lobby della partita a cui si è unito. Nel prototipo, il pulsante *Create Game* è disabilitato;

- **Schermata della lobby di una partita** (Figura 7): mostra lo stato corrente della lobby della partita a cui l'utente sta partecipando. Premendo sul checkbox *Ready*, l'utente notifica il server di essere o non essere pronto a cominciare la partita. Premendo il pulsante *Start Game*, l'host può cominciare la partita se il server è pronto ad iniziarla. In tal caso, viene mostrata la schermata di gioco dell'applicazione;

- **Schermata di gioco** (Figura 8): mostra lo stato corrente della partita a cui l'utente sta partecipando. Quando ha il controllo della barra della propria squadra, l'utente può muoverla verticalmente o fermarla. Alla vittoria di una delle due squadre, viene mostrato il risultato della partita all'utente;
- **Schermata del risultato della partita** (Figura 9): mostra il risultato della partita a cui l'utente ha partecipato. Premendo il pulsante *Return to homepage*, viene mostrata la schermata principale all'utente.

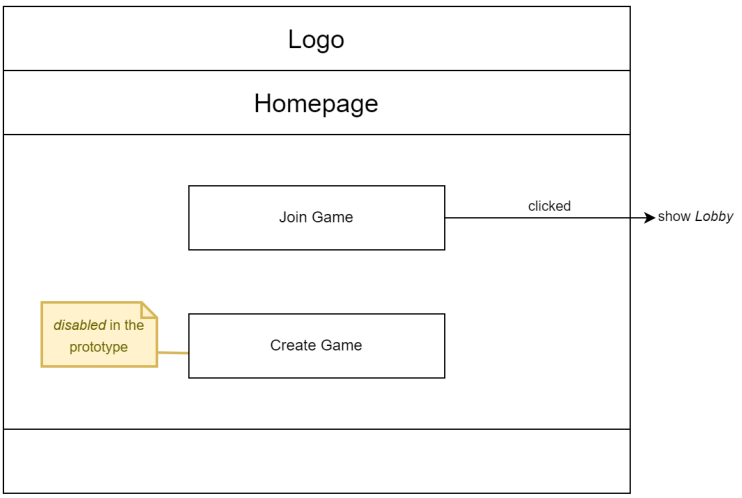


Figure 6: Mockup della schermata principale dell'applicazione.

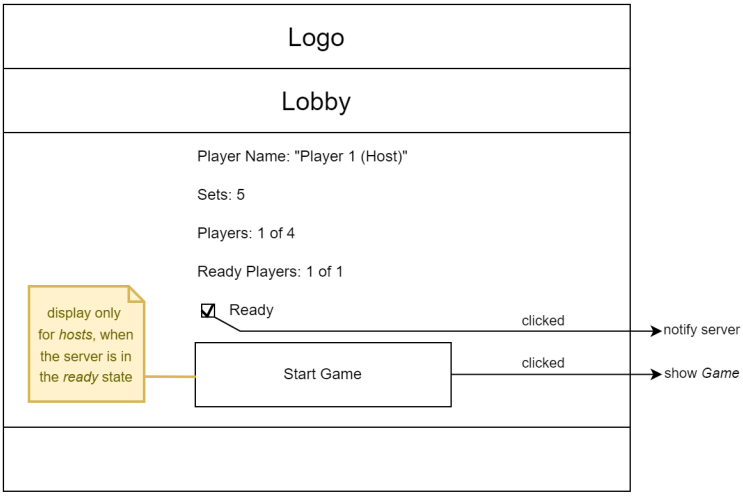


Figure 7: Mockup della schermata della lobby di un server di gioco.

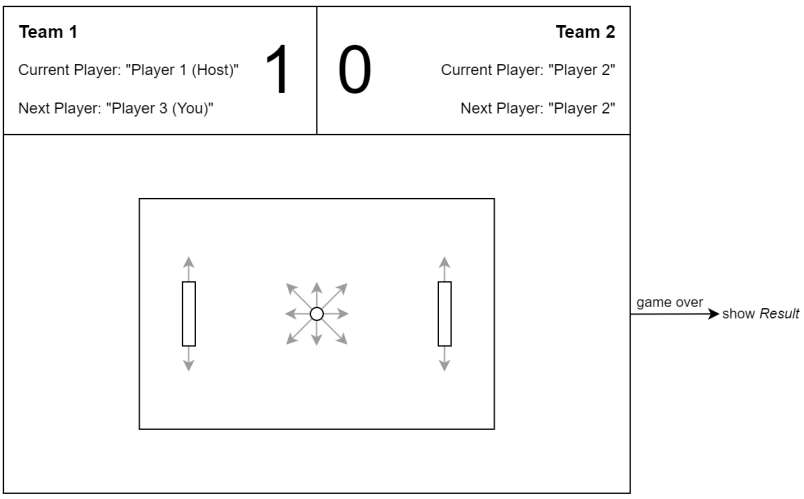


Figure 8: Mockup della schermata di gioco dell'applicazione.

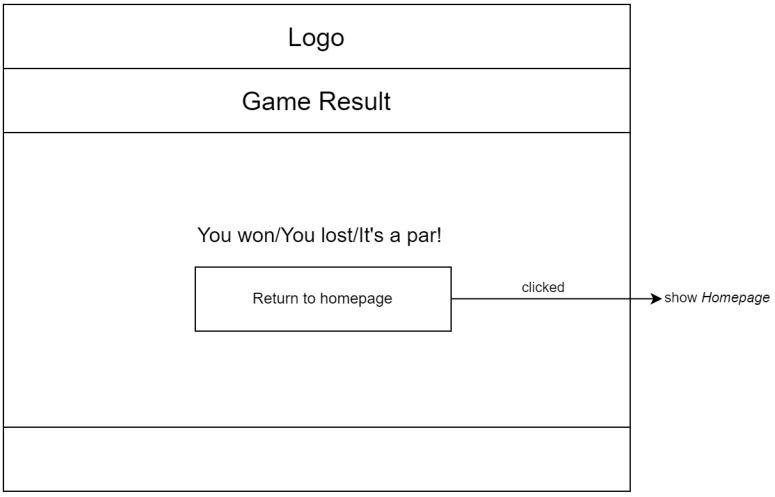


Figure 9: Mockup della schermata contenente il risultato di una partita.

3.2.3 Backend

L'interfaccia del Backend deve permettere all'utente di ottenere l'indirizzo di un server di gioco per ospitare o partecipare a una partita. Nella versione prototipale, considerando che esiste un'unica istanza del servizio Game Server, il Backend si occupa semplicemente di reindirizzare l'utente a quel server di gioco.

Le funzionalità del servizio di Backend sono accessibili mediante il protocollo HTTP. Pertanto, il suo contratto è stato modellato attraverso la seguente interfaccia *REST* [8]:

- `GET /findOrHostGame/{gameId}`: restituisce l'indirizzo a cui l'utente può connet-

tersi via Websocket per ospitare o partecipare a una partita all'interno dell'unico server di gioco presente nel sistema. La richiesta dell'utente contiene l'identificatore della partita. La risposta è restituita in formato *text/plain*.

3.2.4 Game Server

L'interfaccia del Game Server deve permettere all'utente di partecipare a una specifica partita, tra quelle gestite dal server di gioco. Infatti, si è deciso che un singolo Game Server possa ospitare più partite, separando in questo modo la creazione logica di una partita dalla creazione fisica di un server di gioco. In questo nuovo contesto, il ciclo di vita del Game Server, descritto in fase di design architetturale, è invece da intendersi come ciclo di vita di una partita all'interno del server di gioco.

Le funzionalità del servizio Game Server sono esposte mediante una connessione basata sul protocollo WebSocket. Il suo contratto è il seguente:

- `WS /findOrHost/{gameId}`: stabilisce una connessione WebSocket tra l'utente ed il server di gioco, registrando tale utente come un partecipante alla partita specificata, se possibile. La richiesta dell'utente deve contenere l'identificatore della partita a cui vuole partecipare. Nel caso in cui non esistesse nessuna partita con quell'identificatore, essa viene creata e l'utente ne diventa automaticamente l'host. Invece, nel caso in cui tale partita non accettasse altri partecipanti, la connessione viene chiusa.

Nella prospettiva del server di gioco, all'interno della connessione con un utente, si possono distinguere due tipologie di messaggi:

- **In-bound**: inviati dall'utente e ricevuti dal server di gioco. Questi messaggi sono i comandi che l'utente invia al server di gioco, in formato *testuale*. I comandi che sono resi disponibili ad un utente sono:
  - **Set Ready**: memorizza che il mittente è pronto a cominciare la partita. Disponibile solo nella lobby della partita;
  - **Set Not Ready**: memorizza che il mittente non è pronto a cominciare la partita. Disponibile solo nella lobby della partita;
  - **Start Game**: comincia la partita, se può cominciare. Disponibile a un host, solo nella lobby della partita;
  - **Move Up**: muove una barra di gioco verso l'alto. Disponibile ai partecipanti in controllo di una barra, solo durante lo svolgimento della partita;
  - **Move Down**: muove una barra di gioco verso il basso. Disponibile ai partecipanti in controllo di una barra, solo durante lo svolgimento della partita;
  - **Stop**: ferma una barra di gioco. Disponibile ai partecipanti in controllo di una barra di gioco, solo durante lo svolgimento della partita.
- **Out-bound**: inviati dal server di gioco e ricevuti dall'utente. L'unico messaggio di questo tipo, contiene lo stato aggiornato di una partita, in formato *JSON* [10], e

viene inviato ai suoi partecipanti ogni volta che la partita subisce un cambiamento. Il contenuto del messaggio ha la seguente struttura:

```
{
  gameId: "the id of the game",
  gameState : "the state of the game",
  teams : [
    {
      players: [
        {
          id: "the id of the player",
          name: "the name of the player",
          isReady: "true if the player is ready",
          isThisPlayer?: "defined if he is the receiver",
          isHost?: "defined if he is the host",
          hasControl?: "defined if he has control of his team",
          hasControlNext?: "defined if he'll have control next",
        }, ...
      ],
      score: "the current score of the team",
    }, ...
  ],
  arena: {
    firstBarPosition: "the vertical position of the first bar",
    secondBarPosition: "the vertical position of the second bar",
    ballPosition: "the position of the ball",
  },
  settings: {
    playerCapacity: "the maximum number of participants",
    numberOfSets: "the maximum number of sets",
  },
}
```

Listing 1: Struttura dello stato di una partita in formato JSON.

Chiaramente, nella prospettiva dell'utente, i messaggi *in-bound* per il server di gioco sono messaggi *out-bound*, mentre quelli *out-bound* per il server di gioco sono messaggi *in-bound*.

4 Implementazione

In questo capitolo, si descriveranno gli artefatti software prodotti per implementare i componenti del sistema individuati in fase di progettazione, ovvero il Game Server, il Backend ed il Frontend. In particolare, per ogni componente del sistema, si entrerà nel merito dei moduli che lo compongono e della loro funzione. Maggiori informazioni sono incluse nella documentazione del codice.

4.1 Game Server

Per realizzare il Game Server, si è scelto di adottare come architettura il **Game Loop** [13], in cui i componenti sono progettati attorno all'esecuzione di un loop che gestisce lo svolgimento di una partita, chiamato *game loop* per l'appunto. In particolare, ad ogni iterazione, il game loop prima processa l'input ricevuto dai giocatori, dopodiché aggiorna lo stato del gioco di conseguenza, infine notifica i giocatori del nuovo stato, finché il gioco non termina.

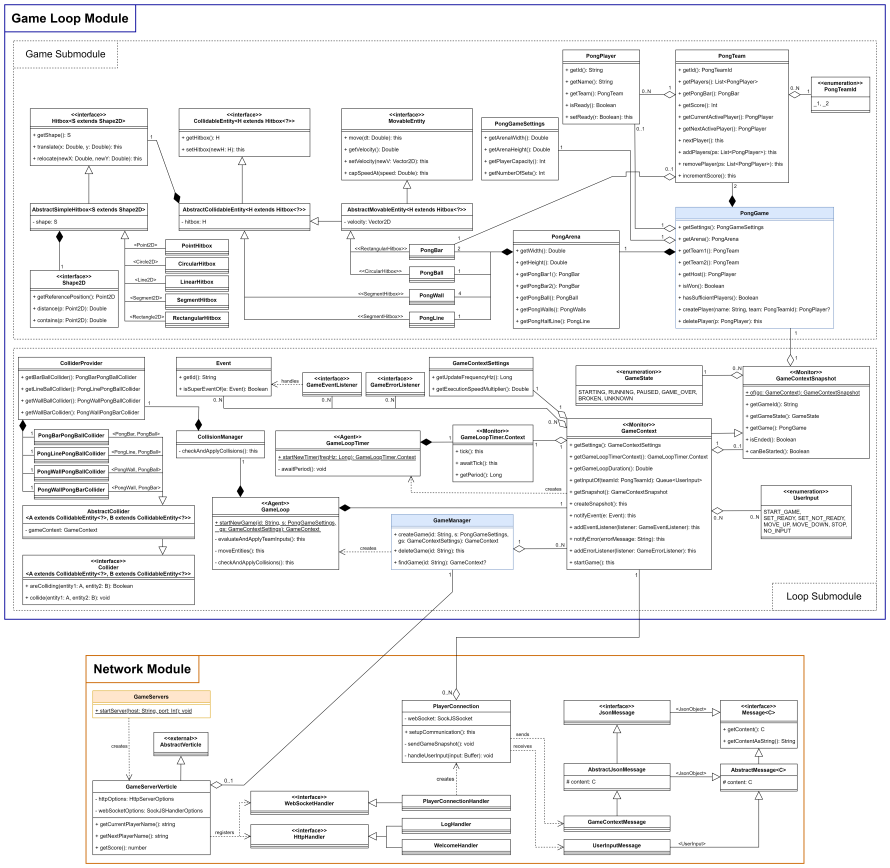


Figure 10: Diagramma delle classi che rappresenta i moduli di cui è composto il Game Server.

Come illustrato in Figura 10, il Game Server si compone di due moduli principali:

- **Game Loop Module**: contiene la definizione dei concetti e della logica necessari allo svolgimento di una partita. Il Game Loop Module si compone di due sottomoduli:
  - **Game Submodule**: contiene la modellazione delle entità presenti nel gioco;
  - **Loop Submodule**: contiene la definizione del game loop, quindi delle reazioni del gioco alle intenzioni espresse dai suoi giocatori e delle interazioni tra le entità del gioco.
- **Network Module**: contiene la definizione del server di gioco, che gestisce le connessioni dei giocatori al servizio, e dei messaggi che possono essere inviati o ricevuti da tale server.

4.1.1 Game Loop Module

In questa sezione, si illustreranno i due sottomoduli di cui è composto il Game Loop Module: il Game Submodule ed il Loop Submodule.

4.1.1.1 Game Submodule

Il Game Submodule, illustrato in Figura 11, è il componente del Game Server che contiene l'implementazione delle entità che interagiscono durante una partita di *Relay Pong*.

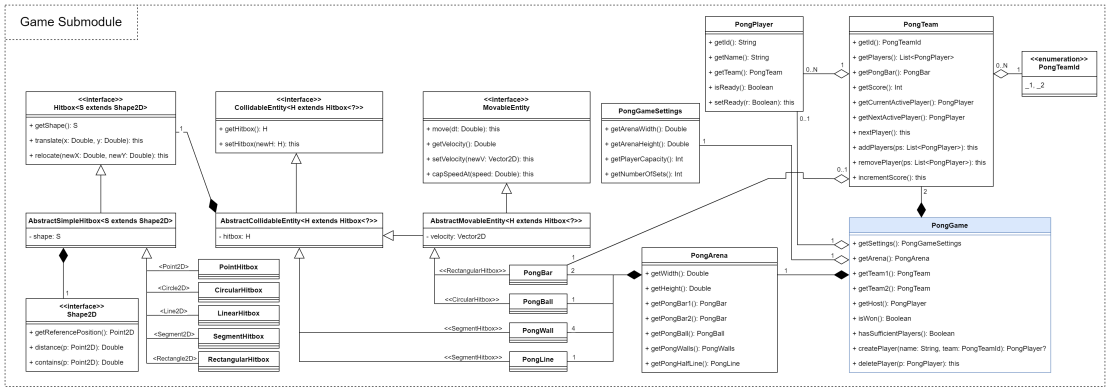


Figure 11: Diagramma delle classi che rappresenta il Game Submodule nel Game Server. Nel diagramma è evidenziato il componente principale del modulo: la classe `PongGame`.

Il modulo contiene la rappresentazione logica di una partita del gioco, modellata dalla classe `PongGame`. In particolare, un `PongGame` è composto da:

- *Un'arena di gioco*: rappresentata dalla classe `PongArena`. Una `PongArena` contiene gli elementi di gioco, classificabili in due categorie:

- `CollidableEntity`: un'entità che possiede una hitbox, modellata dalla classe `Hitbox`, con una certa forma bidimensionale, modellata dalla classe `Shape2D`;
- `MovingEntity`: un'entità che possiede una certa velocità ed è in grado di muoversi in uno spazio bidimensionale.

Nello specifico, una `PongArena` è composta da:

- *Una pallina*: rappresentata dalla classe `PongBall`. Una `PongBall` è sia una `MovableEntity` che una `CollidableEntity` con una hitbox circolare, modellata dalla classe `CircularHitbox`;
- *Due barre di gioco*: rappresentate dalla classe `PongBar`. Una `PongBar` è sia una `MovableEntity` che una `CollidableEntity` con una hitbox rettangolare, modellata dalla classe `RectangularHitbox`;
- *Un perimetro di gioco*: composto da quattro mura, modellate dalla classe `PongWall`. Un `PongWall` è una `CollidableEntity` con una hitbox lineare di una certa lunghezza, modellata dalla classe `SegmentHitbox`;
- *Una linea di metà campo*: modellata dalla classe `PongLine`. Una `PongLine` è una `CollidableEntity` con una hitbox lineare di una certa lunghezza, modellata dalla classe `SegmentHitbox`.

Attraverso la `PongArena`, è anche possibile resettare l'arena di gioco riportandola alle condizioni iniziali, assegnando alla `PongBall` una velocità con una direzione scelta in modo casuale.

- *Due squadre*: rappresentate dalla classe `PongTeam`. Un `PongTeam` è caratterizzato da un identificatore univoco all'interno della partita, modellato dall'enumerazione `PongTeamId`, una `PongBar` di cui ha il possesso e un certo numero di giocatori, modellati dalla classe `PongPlayer`. Un `PongPlayer` è caratterizzato da un identificatore univoco all'interno della partita, un nome ed uno stato, che indica se è pronto o meno a cominciare la partita. Un `PongTeam` tiene traccia anche dell'ordine in cui il controllo della barra della squadra è ceduto ai giocatori che vi appartengono;
- *Un host*: il `PongPlayer` che possiede il ruolo di host della partita, come descritto in fase di progettazione. Il ruolo di host è assunto dal primo giocatore che partecipa alla partita. Nel caso in cui l'host lasciasse la partita, il suo ruolo è ceduto al giocatore che ha identificatore minore in ordine alfabetico, che corrisponde a quello sta partecipando alla partita da più tempo.

Un `PongGame` gestisce la creazione e la rimozione dei giocatori in una partita, oltre all'assegnazione dinamica del ruolo di host a uno dei giocatori. Inoltre, definisce quando la partita ha abbastanza giocatori per cominciare e quando la partita è da considerarsi terminata. Un `PongGame` è anche configurabile attraverso delle impostazioni, modellate dalla classe `PongGameSettings`, che descrivono le *dimensioni logiche dell'arena di gioco*, il *numero massimo di giocatori* della partita ed il *numero di massimo di set* che possono essere giocati all'interno della partita.

Questo modulo si appoggia ad una libreria di utility per la gestione di alcune forme bidimensionali, implementata appositamente per questo progetto. La libreria definisce i concetti geometrici di:

- *Punto*: modellato dalla classe `Point2D`;
- *Vettore*: modellato dalla classe `Vector2D`;
- *Cerchio*: modellato dalla classe `Circle2D`;
- *Retta*: modellata dalla classe `Line2D`;
- *Segmento*: modellato dalla classe `Segment2D`;
- *Rettangolo*: modellato dalla classe `Rectangle2D`.

Nella libreria sono implementate molte relazioni matematiche tra poligoni in due dimensioni, che saranno utilizzate intensamente per il controllo delle collisioni durante lo svolgimento di una partita.

4.1.1.2 Loop Submodule

Il Loop Submodule, illustrato in figura 12, è il componente del Game Server che gestisce l'esecuzione del game loop e definisce come le entità del gioco possono interagire durante lo svolgimento di una partita.

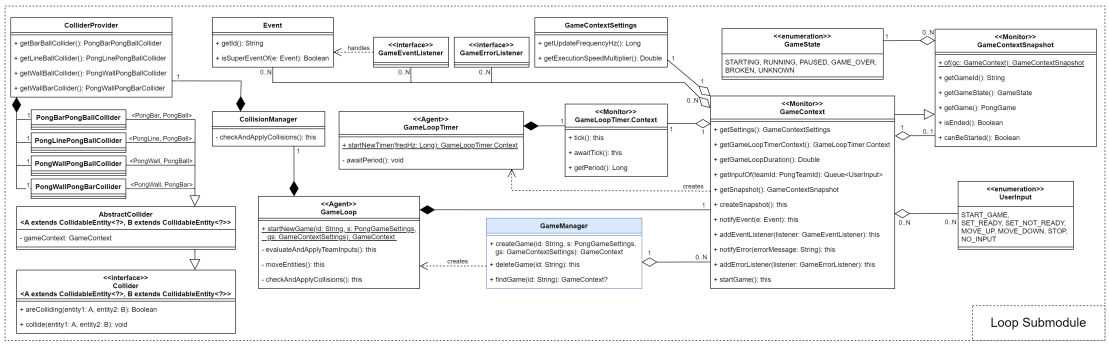


Figure 12: Diagramma delle classi che rappresenta il Loop Submodule nel Game Server. Nel diagramma è evidenziato il componente principale del modulo: la classe `GameManager`.

Il componente principale del modulo è il `GameManager`, che gestisce un insieme di partite nel Game Server, permettendone la creazione, l'eliminazione o la ricerca. Alla richiesta di creazione di una partita, il `GameManager` inizializza un nuovo game loop, modellato dalla classe `GameLoop`.

Il `GameLoop` è un componente attivo del sistema che gestisce lo svolgimento di una partita. Alla sua creazione, inizializza e restituisce il suo contesto di esecuzione, modellato dalla classe `GameContext`. Il contesto di esecuzione di un agente è da intendersi come un componente passivo, che permette di interagire indirettamente con l'agente.

A sua volta, il `GameContext` inizializza un altro componente attivo, il `GameLoopTimer`, il cui scopo è scandire la frequenza di aggiornamento dello stato della partita gestita dal game loop. Analogamente al `GameLoop`, alla sua creazione, il `GameLoopTimer` inizializza e restituisce il suo contesto di esecuzione, modellato dalla classe `GameLoopTimer.Context`.

Un `GameContext` contiene lo stato delle entità di una partita, modellato dalla classe `PongGame`, contenuta nel Game Submodule. Inoltre, è caratterizzato da un identificatore univoco all'interno del `GameManager`, da un proprio stato, modellato dalla classe `GameState`, e da due code per gli input delle squadre, modellati dalla classe `UserInput`. Il `GameContext` contiene anche la configurazione riguardante l'esecuzione della sua partita, modellata dalla classe `GameContextSettings`, che descrive la *frequenza di aggiornamento dello stato della partita* e la *velocità di simulazione della partita*, ovvero quanto velocemente scorre il tempo virtuale del gioco, influenzando il movimento delle entità nell'arena.

Il `GameContext` di una partita espone anche molte funzionalità per la modifica dello stato della partita, come l'*inizio della partita*, oppure la *creazione e rimozione di giocatori*. Infatti, per non causare corse critiche, tutte le scritture sullo stato della partita devono essere mediate dal suo `GameContext`. Invece, è possibile eseguire letture concorrenti in ogni momento, accedendo allo snapshot corrente del `GameContext`, modellato dalla classe `GameContextSnapshot`. In ogni caso, quando la partita è cominciata, il suo stato dovrebbe essere modificato solo all'interno del `GameLoop`. In retrospettiva, per evitare queste complicanze, si sarebbe potuto adottare l'*event-loop* come modello d'esecuzione del game loop.

Attraverso il `GameContext` di una partita, è anche possibile registrare degli *event-handler*, modellati dalla classe `GameEventListener`, da eseguire in reazione a determinati eventi accaduti durante lo svolgimento della partita, modellati dalla classe `Event`. Gli eventi generabili all'interno di un `GameContext` sono contenuti nella classe `GameEvents`, che include eventi riguardanti l'aggiornamento dello stato della partita, la creazione o eliminazione dei giocatori nella partita, il reset dell'arena di gioco o l'aggiudicamento di un set da parte di una squadra. Analogamente, è possibile registrare degli *error-handler*, modellati dalla classe `GameErrorListener`, da eseguire in reazione a degli errori. L'esecuzione di tali handler è delegata a chi genera gli eventi o errori a cui devono reagire.

Quando creato, il `GameLoop` si trova nello stato `GameState.STARTING`, ovvero lo stato in cui i giocatori sono nella lobby della partita, in attesa che vi siano abbastanza partecipanti per cominciare a giocare. Quando tutti i partecipanti sono pronti, attraverso il `GameContext` è possibile cominciare la partita.

Durante l'esecuzione di una partita, il `GameLoop` si trova nello stato `GameState.RUNNING`, in cui esegue iterativamente la seguente sequenza di azioni:

1. *Elaborazione delle intenzioni dei giocatori*: per ogni squadra, si analizza il prossimo input ricevuto dalla squadra e si aggiornano le velocità delle barre di gioco di conseguenza. Gli input accettati dal `GameLoop` sono:
    - `UserInput.MOVE_UP`: imposta la velocità di una barra di gioco ad un valore positivo;

- `UserInput.MOVE_DOWN`: imposta la velocità di una barra di gioco ad un valore negativo;
- `UserInput.STOP`: imposta la velocità di una barra di gioco ad un valore nullo.

2. *Movimento delle entità del gioco*: si muovono la pallina e le barre delle due squadre in base alla loro velocità. Dopodiché, se la pallina non ha raggiunto la sua massima velocità, si aumenta la velocità della pallina. In questo modo, si rende il set in corso sempre più difficile, evitando che si prolunghi per troppo tempo;

3. *Gestione delle collisioni*: si controllano le collisioni tra le entità del gioco, riposizionandole ed aggiornando la loro velocità, eventualmente generando anche degli eventi particolari. Questo compito è delegato ad un componente del `GameLoop`, chiamato `CollisionManager`.

   La logica delle collisioni tra due entità del gioco è descritta da un `Collider`, che definisce quando due entità collidono e quali sono le conseguenze della loro collisione. Nel gioco esistono quattro tipi di collisione:

   - *Collisione tra una barra di gioco e la pallina*: gestita dal `PongBarPongBallCollider`, modifica la velocità della pallina in base al punto di collisione con la barra di gioco;
   - *Collisione tra la pallina e la linea di metà campo*: gestita dal `PongLinePongBallCollider`, cede il controllo dell'ultima barra di gioco che ha colpito la pallina al prossimo giocatore della squadra che la controlla, solo se la squadra possiede almeno due giocatori. Quando il controllo di una barra passa a un nuovo giocatore, la barra viene fermata;
   - *Collisione tra una barra di gioco e un muro dell'arena*: gestita dal `PongWallPongBarCollider`, ferma la barra di gioco, impedendole di uscire dal perimetro dell'arena;
   - *Collisione tra la pallina e un muro dell'arena*: gestita dal `PongWallPongBallCollider`, modifica la velocità della pallina, impedendole di uscire dal perimetro dell'arena. Se il muro è verticale, assegna un punto a una squadra, generando un evento opportuno, e resetta l'arena di gioco.

   Quando richiesto, il `GameManager` scorre tutte le entità dell'arena alla ricerca di collisioni, utilizzando i `Collider` forniti dalla classe `ColliderProvider`. Le collisioni vengono quindi applicate nell'ordine in cui sono state individuate.

4. *Controllo delle win-condition*: si controlla se la partita è terminata, verificando se una delle due squadre si è aggiudicata la maggioranza dei set della partita. In tal caso, il `GameLoop` entra nello stato `GameState.GAME_OVER`, prima di interrompersi.

5. *Attesa del prossimo tick*: si attende un tick del `GameLoopTimer`.

Alla fine di ogni iterazione, il `GameLoop` crea un nuovo snapshot del suo contesto di esecuzione e genera un evento corrispondente all'avvenuto aggiornamento dello stato della partita, in modo che dall'esterno sia possibile reagire al nuovo stato della partita.

Nel caso in cui accadessero degli errori durante l'esecuzione della partita che ne impediscano il proseguimento, il `GameLoop` entra nello stato `GameState.BROKEN`, prima di interrompersi. Questo può succedere, ad esempio, quando un giocatore si disconnette dalla partita ed era l'unico membro rimasto della sua squadra.

Al termine di una partita, che sia per suo naturale decorso o a causa di un errore, il `GameManager` la rimuove dall'insieme delle partite che gestisce. In questo modo, è possibile creare una nuova partita, eventualmente con lo stesso identificatore di quella eliminata.

4.1.2 Network Module

Il Network Module, illustrato in Figura 13, è il componente del Game Server che gestisce le connessioni a tale servizio.

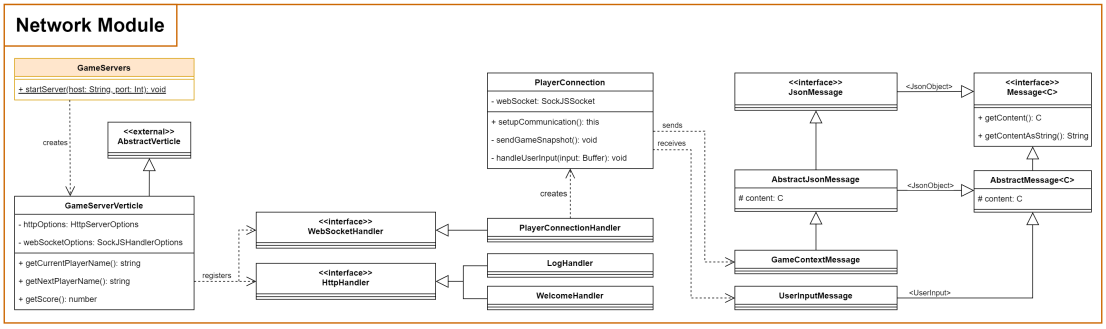


Figure 13: Diagramma delle classi che rappresenta il Network Module nel Game Server. Nel diagramma è evidenziato il componente principale del modulo: la classe `GameServers`.

Il modulo permette di creare l'istanza di un server di gioco attraverso la classe `GameServers`. I server di gioco sono stati implementati tramite la libreria Vert.x, quindi sono dei *verticle*, modellati dalla classe `GameServerVerticle`. Ogni `GameServerVerticle` si affida ad un `GameManager`, descritto nel Loop Submodule, per la gestione delle partite che ospita.

Un `GameServerVerticle` accetta connessioni di tipo HTTP e WebSocket, gestite rispettivamente da handler di tipo `HttpHandler` e `WebSocketHandler`. Tali handler sono registrati sul router del `GameServerVerticle`, che espone le seguenti funzionalità:

- `GET /`: gestita dalla classe `WelcomeHandler`, permette all'utente di verificare se il server è raggiungibile, ricevendo una risposta in formato *text/plain* nel caso in cui lo fosse. Questa funzionalità è stata implementata per essere utilizzata solo in fase di sviluppo;
- `WS /findOrHost/{gameId}`: gestita dalla classe `PlayerConnectionHandler`, permette la partecipazione di un giocatore alla partita specificata all'interno del server di gioco, come descritto in fase di progettazione.

Quando l'utente invia una richiesta a questo indirizzo, il `PlayerConnectionHandler` cerca la partita con l'identificatore specificato tra quelle gestite dal `GameManager` del `GameServerVerticle`. Se non la trova, ne crea una nuova con quell'identificatore.

Una volta trovata o creata la partita e ottenuto il suo `GameContext`, il `PlayerConnectionHandler` tenta di creare un nuovo giocatore nella partita, corrispondente all'utente. Se ci riesce, perchè la partita non è ancora cominciata e non ha raggiunto la capacità massima di giocatori, il `PlayerConnectionHandler` inizializza una `PlayerConnection`, che configura la comunicazione tra la partita e l'utente, altrimenti la connessione con l'utente viene interrotta.

Una `PlayerConnection` configura due tipi di comunicazione:

- *Comunicazione dal server verso il client*: registra degli event-handler nel `GameContext` della partita, in modo da inviare all'utente un messaggio contenente il suo ultimo snapshot, in reazione ad eventi riguardanti l'aggiornamento dello stato della partita, oppure la connessione o disconnessione dei giocatori dalla partita.

  La serializzazione di tale messaggio è delegata alla classe `GameContextMessage`, che produce un messaggio conforme alle specifiche di formato individuate in fase di progettazione, a partire dallo snapshot di un `GameContext`;

- *Comunicazione dal client verso il server*: registra un handler nella connessione, in modo da gestire i messaggi ricevuti dall'utente. I messaggi accettati dall'handler sono le intenzioni dell'utente, descritte in fase di progettazione, modellate dall'enumerazione `UserInput`.

  La deserializzazione di tali messaggi è delegata alla classe `UserInputMessage`, che produce uno `UserInput`, a partire da un messaggio ricevuto dall'utente, contenente un numero corrispondente all'intenzione da lui espressa.

Durante la connessione, il server di gioco mantiene aggiornati i partecipanti alla partita, secondo un'architettura di tipo *push*, quindi evitando *polling* da parte dei giocatori. Dall'altra parte, i giocatori possono notificare il server delle proprie intenzioni, inviando messaggi contenenti uno `UserInput`. In particolare, il server di gioco interpreta le intenzioni di un utente nel modo seguente:

- `UserInput.START_GAME`: nella lobby della partita, se l'utente è l'host e la partita può cominciare, inizia la partita;
- `UserInput.SET_READY` oppure `UserInput.SET_NOT_READY`: nella lobby della partita, imposta lo stato dell'utente, memorizzando che è pronto o non è pronto a cominciare la partita;
- `UserInput.MOVE_UP`, `UserInput.MOVE_DOWN` oppure `UserInput.STOP`: durante lo svolgimento della partita, se l'utente ha il controllo della barra della propria squadra, inserisce l'input nella coda dei comandi della squadra.

Il `GameServerVerticle` tiene anche traccia delle richieste ricevute, mostrandole sulla console del servizio Game Server.

4.2 Backend

Per realizzare il servizio di Backend, è stata scelta un'architettura *Router-Controller*, tipica di Express, composta di due moduli principali:

- **Routes**: contiene la definizione del contratto del servizio, definito attraverso lo standard *REST*;
- **Controllers**: contiene l'implementazione delle funzionalità esposte dal servizio.

Nella versione prototipale del sistema, il servizio di Backend è talmente semplice che è stato implementato un unico router, il `MainRouter`. Il `MainRouter` espone le seguenti funzionalità del servizio:

- `GET /`: permette all'utente di verificare se il servizio è raggiungibile, ricevendo una risposta in formato *text/plain* nel caso in cui lo fosse. Questa funzionalità è stata implementata per essere utilizzata solo in fase di sviluppo;
- `GET /findOrHostGame/{gameId}`: reindirizza l'utente all'unico server di gioco presente nel sistema, come descritto in fase di progettazione.

Il `MainRouter` tiene anche traccia delle richieste ricevute, mostrandole sulla console del servizio di Backend.

4.3 Frontend

Per realizzare l'applicazione fornita dal servizio di Frontend, è stata scelta come architettura il **Model-View-ViewModel**, il quale prevede la definizione di un modello dei dati e della logica applicativa, detto *Model*, quindi la definizione di un'interfaccia in grado di rappresentare tale modello e di gestire le interazioni con l'utente, detta *View*, ed infine la definizione di un intermediario tra i due, che reagisce ai cambiamenti dell'interfaccia applicandoli automaticamente al modello e viceversa, chiamato *ViewModel*.

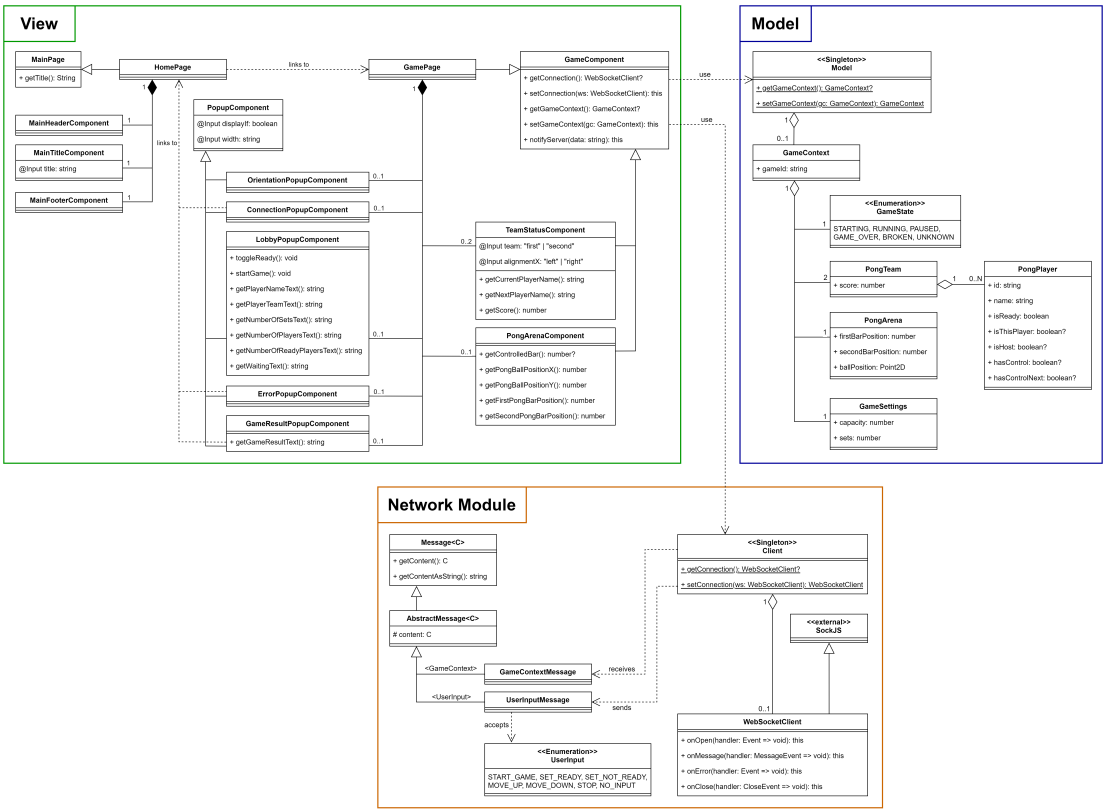


Figure 14: Diagramma delle classi che rappresenta i moduli di cui è composto il servizio di Frontend.

Come illustrato in Figura 14, il servizio di Frontend si compone di tre moduli principali:

- **Model**: contiene la definizione delle strutture dati necessarie a descrivere lo stato di una partita;
- **View**: contiene la definizione delle schermate dell'applicazione e dei componenti grafici di cui sono costituite. Inoltre, gestisce il flusso delle interazioni tra l'utente e l'applicazione;
- **Network Module**: gestisce la connessione a un server di gioco e definisce i messaggi che possono essere inviati o ricevuti dall'utente tramite tale connessione.

Utilizzando Angular come framework di sviluppo, non è necessario definire esplicitamente un modulo che rappresenti il **View-Model**. Infatti, Angular fornisce già alcune funzionalità per creare delle dipendenze tra l'interfaccia grafica dell'applicazione ed il suo modello, come il *data-binding* [6].

4.3.1 Model

Il Model, illustrato in Figura 15, è il componente che contiene l'implementazione della struttura dati che descrive lo stato di una partita.

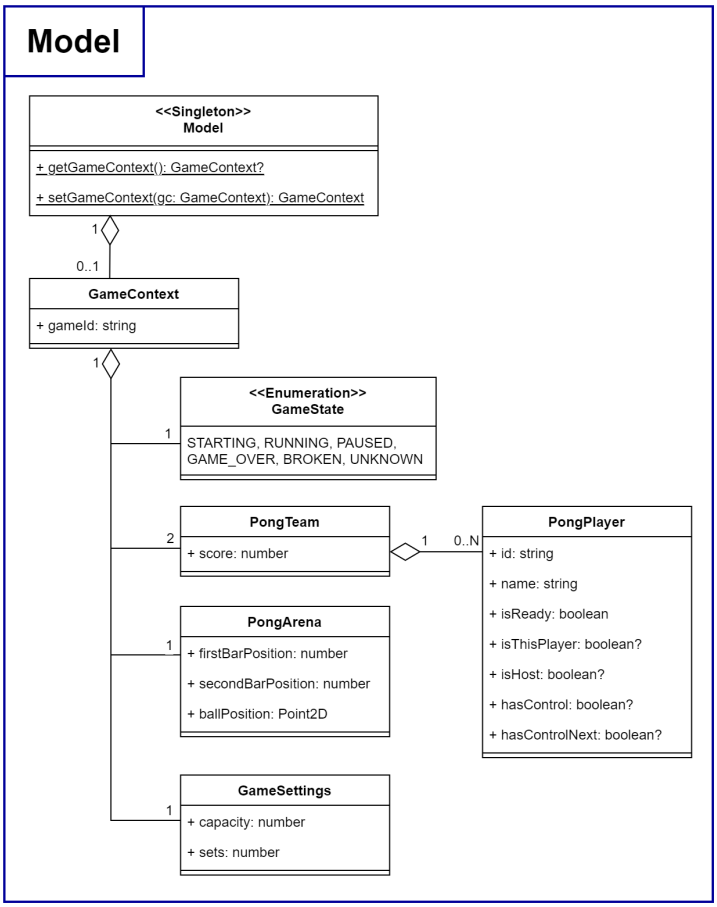


Figure 15: Diagramma delle classi che rappresenta il Model nell'applicazione fornita dal servizio di Frontend.

Considerando che un utente non può mai essere connesso a più partite contemporaneamente, si è scelto di utilizzare un *singleton*, modellato dalla classe `Model`, per accedere dall'interno dell'applicazione allo stato della partita a cui l'utente è attualmente connesso.

Lo stato della partita è modellato dalla classe `GameContext`, che contiene:

- *Uno stato*: rappresentato dalla classe `GameState`;
- *Due squadre*: rappresentate dalla classe `PongTeam`. Ogni squadra può eventualmente contenere dei giocatori, rappresentati dalla classe `PongPlayer`;
- *Un'arena di gioco*: rappresentata dalla classe `PongArena`;
- *Una configurazione*: rappresentata dalla classe `GameSettings`.

Questa struttura è stata progettata con l'intenzione di aderire all'interfaccia esposta dal servizio Game Server, permettendo all'applicazione di accedere alle informazioni ricevute dal server di gioco in modo più agevole.

4.3.2 Network Module

Il Network Module, illustrato in Figura 16, è il componente che gestisce la comunicazione tra l'applicazione ed il Game Server, attraverso un'implementazione del protocollo WebSocket.

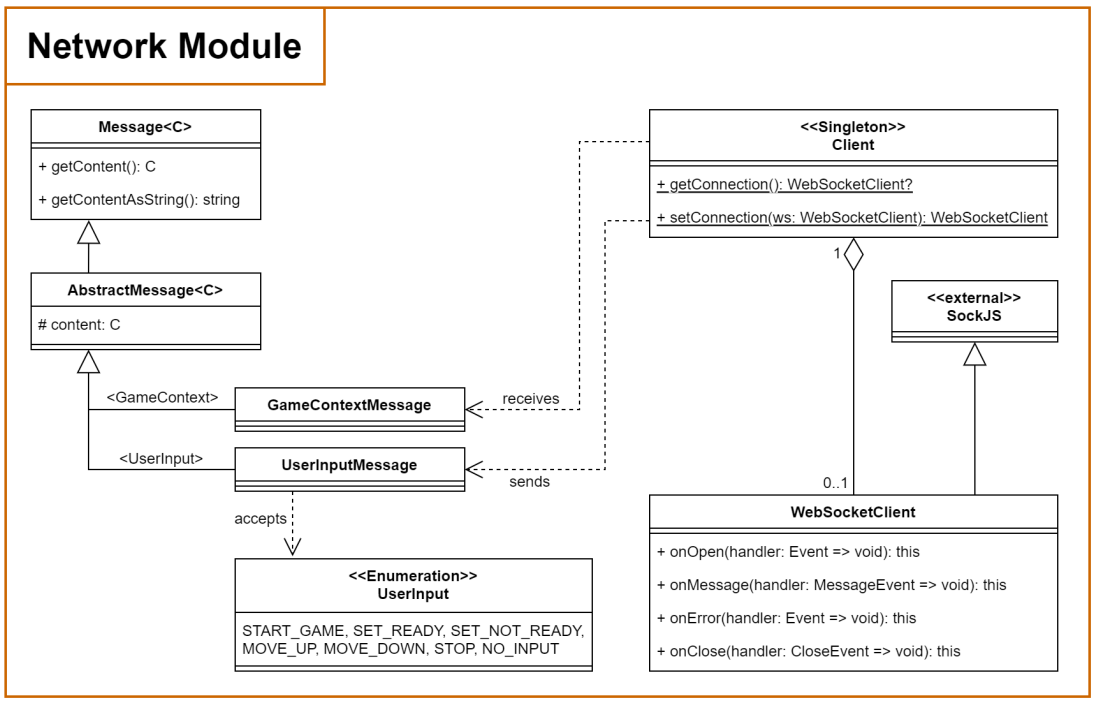


Figure 16: Diagramma delle classi che rappresenta il Network Module nell'applicazione fornita dal servizio di Frontend.

Analogamente a quanto detto per il Model, considerando che un utente non può mai essere connesso a più partite contemporaneamente, si è scelto di utilizzare un *singleton*, modellato dalla classe `Client`, per accedere dall'interno dell'applicazione alla connessione con il server di gioco che gestisce la partita a cui l'utente sta partecipando.

Il modulo permette di connettersi ad un server WebSocket attraverso la classe `WebSocketClient`, che è un *wrapper* fluente della libreria `SockJS`. Nel modulo sono anche contenuti i messaggi che l'applicazione può inviare o ricevere durante la comunicazione con un server di gioco. In particolare, si distinguono due tipologie di messaggi:

- `GameContextMessage`: gestisce la deserializzazione dello stato corrente della partita, inviato dal server di gioco verso l'applicazione;
- `UserInputMessage`: gestisce la serializzazione delle intenzioni dell'utente, inviate dall'applicazione verso il server di gioco. Le intenzioni che possono essere espresse dall'utente sono quelle individuate in fase di design, modellate dall'enumerazione `UserInput`.

4.3.3 View

All'interno dell'applicazione fornita dal servizio di Frontend, la View è il componente che contiene l'implementazione delle schermate dell'applicazione, individuate in fase di progettazione.

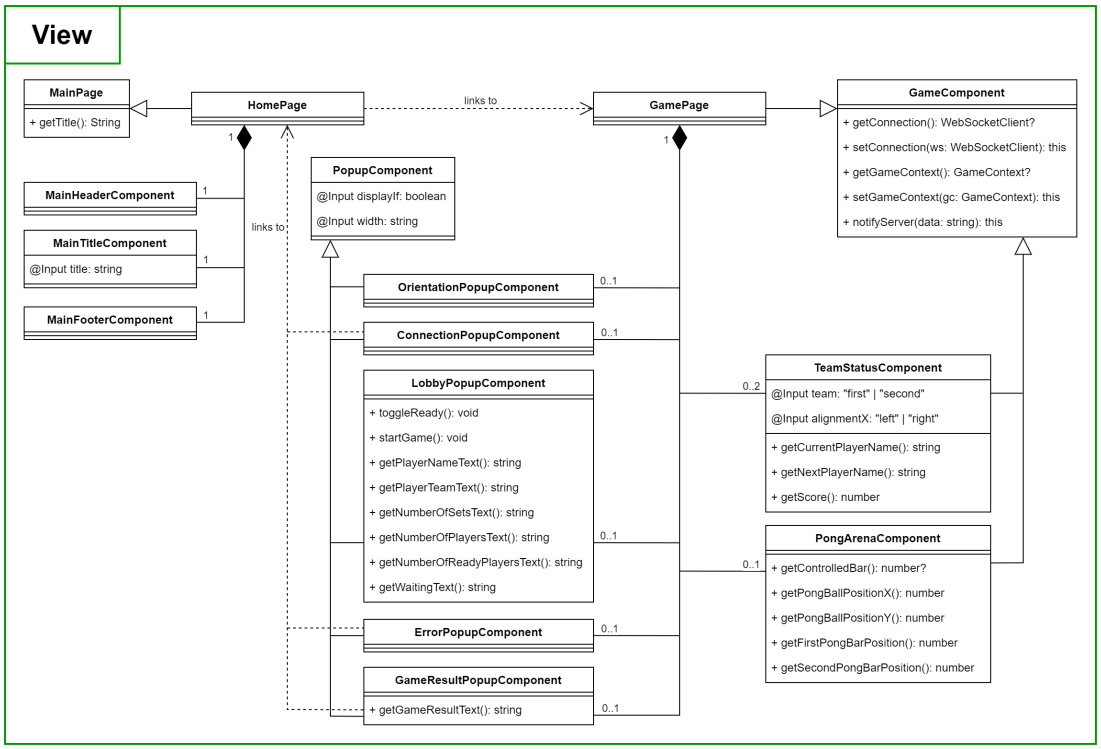


Figure 17: Diagramma delle classi che rappresenta la View nell'applicazione fornita dal servizio di Frontend.

Le schermate dell'applicazione sono state implementate seguendo i principi di modularità promossi dal framework Angular, quindi sono state scomposte in componenti

grafici più semplici e più facilmente manutenibili, illustrati in Figura 17. Tali componenti sono stati organizzati in due categorie principali:

- **Page**: componente che gestisce la rappresentazione grafica di una pagina web, corrispondente ad un certo indirizzo all'interno dell'applicazione;
- **Component**: componente che gestisce la rappresentazione grafica di uno o più concetti locali nell'applicazione.

Il primo componente che viene mostrato all'utente è la `HomePage`, che rappresenta la schermata principale dell'applicazione. La `HomePage` ha una struttura standard che si prevede possa essere riutilizzata in futuro per l'aggiunta di nuove pagine all'applicazione. Le pagine con tale struttura estendono il componente `MainPage`.

Ogni `MainPage` è caratterizzata da un'*intestazione*, modellata dal componente `MainHeaderComponent`, che contiene il logo dell'applicazione, dal *titolo della pagina*, modellato dal componente `MainTitleComponent`, e da un *piè di pagina*, modellato dal componente `MainFooterComponent`, oltre che dal suo contenuto specifico.

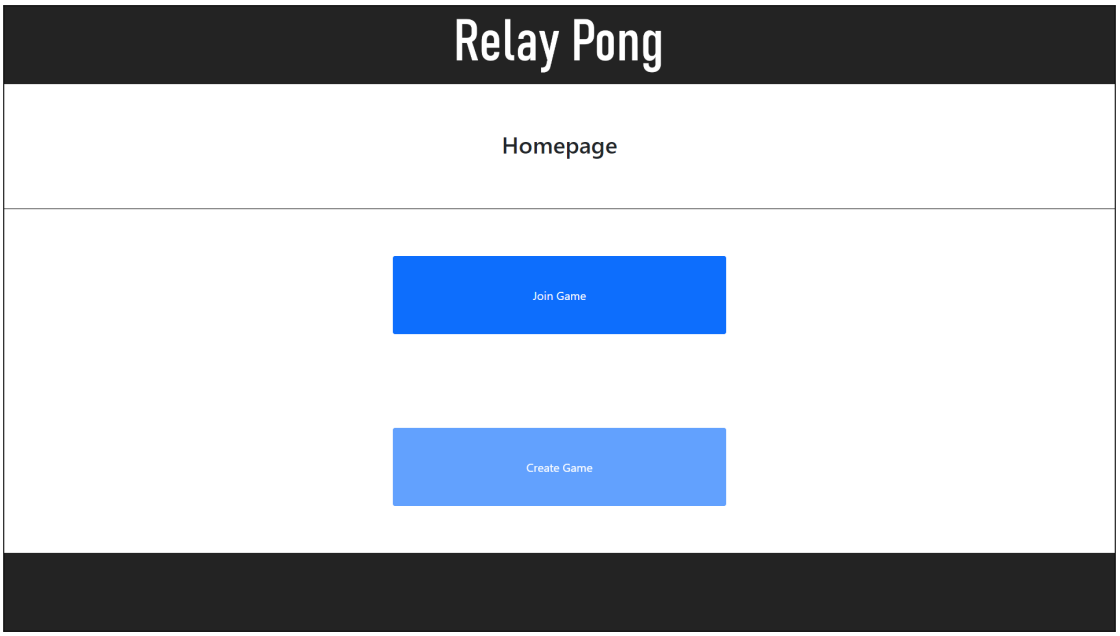


Figure 18: Screenshot della schermata principale dell'applicazione, modellata dal componente `HomePage`.

Come illustrato dalla Figura 18, la `HomePage` permette di accedere alle funzionalità principali dell'applicazione attraverso due pulsanti:

- *Join Game*: permette all'utente di partecipare a una partita. Per la versione prototipale del sistema, alla pressione di questo pulsante, l'utente viene reindirizzato alla schermata di gioco, tentando di connettersi all'unica partita gestita dal sistema;

- *Create Game*: permette all'utente di creare un server di gioco. Per la versione prototipale del sistema, questo pulsante è disabilitato, in quanto la sua funzione non è prevista dai requisiti definiti in fase di analisi.

La schermata di gioco, modellata dal componente `GamePage`, si occupa della rappresentazione grafica di una partita in tutto il suo ciclo di vita, oltre che delle interazioni tra l'utente ed il server di gioco. Quando caricata, l'applicazione richiede al servizio di Backend l'indirizzo a cui deve connettersi per permettere all'utente di partecipare all'unica partita gestita dal sistema. Infatti, in questo prototipo, non è permesso all'utente di scegliere a quale partita partecipare.

Durante il tentativo di connessione, all'utente viene mostrato un popup che permette di annullare il tentativo e ritornare alla schermata principale dell'applicazione, modellato dalla classe `ConnectionPopupComponent` (Figura 19).

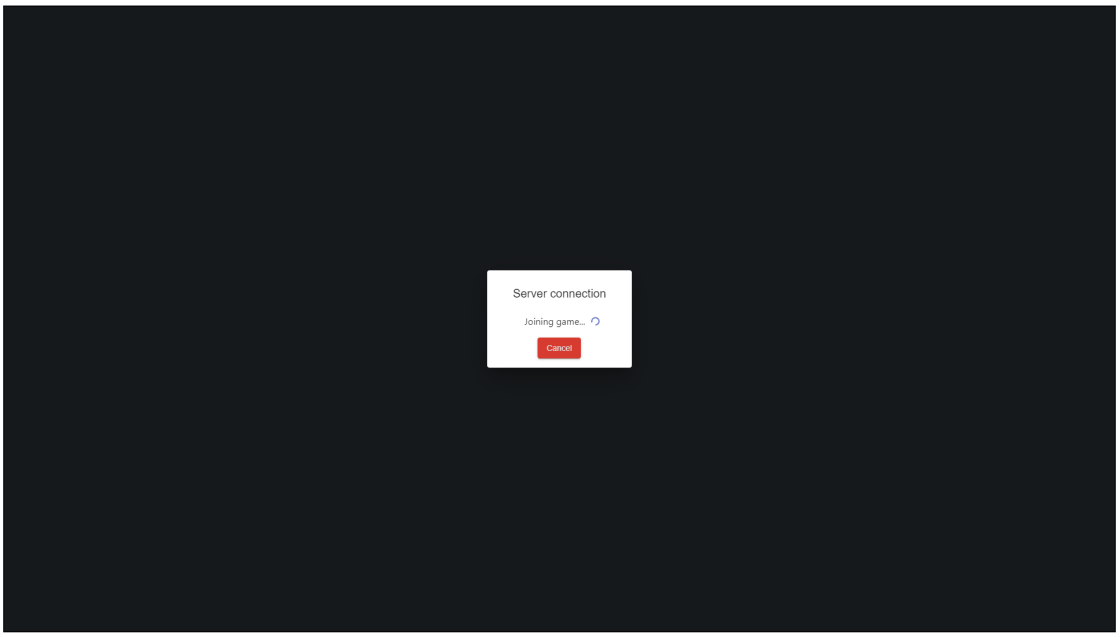


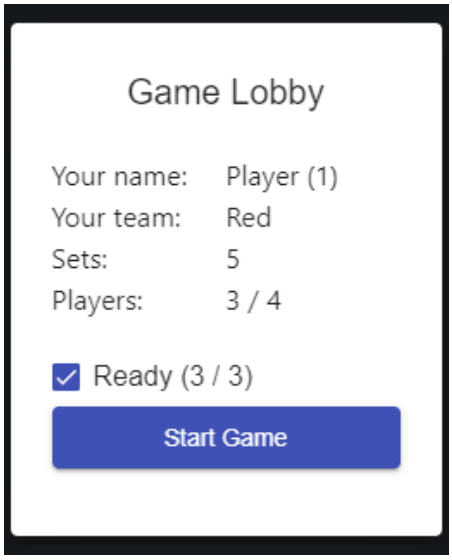
Figure 19: Screenshot della schermata di gioco dell'applicazione, modellata dal componente `GamePage`. Durante il tentativo di connessione dell'utente al server di gioco, all'utente viene mostrato il `ConnectionPopupComponent`.

Una volta connessa al server di gioco, l'applicazione comincia a ricevere messaggi dal server sullo stato aggiornato della partita e mostra un nuovo popup che comunica all'utente lo stato della lobby della partita a cui sta partecipando. Questo popup, modellato dalla classe `LobbyPopupComponent` (Figura 20), permette all'utente di indicare al server di essere o non essere pronto a cominciare la partita. Nel caso in cui l'utente sia l'host della partita e la partita può cominciare, il popup mostra anche un pulsante *Start Game*, che, quando premuto, comanda al server di gioco di iniziare la partita.

(a) Punto di vista dell'host, quando ha appena creato la partita. Ancora non ci sono abbastanza partecipanti per cominciare la partita.

(b) Punto di vista dell'host, quando ci sono abbastanza giocatori per cominciare la partita, ma non tutti i partecipanti sono pronti.

(c) Punto di vista di un utente ospite, in attesa che l'host cominci la partita.



(d) Punto di vista dell'host, quando può cominciare la partita.

Figure 20: Screenshot della schermata rappresentante la lobby di una partita, visualizzata attraverso il `LobbyPopupComponent`.

Alla pressione del pulsante *Start Game*, la schermata di gioco di tutti i partecipanti alla partita mostra l'arena di gioco, modellata dal componente `PongArenaComponent`,

illustrato in Figura 21. Il `PongArenaComponent` contiene la rappresentazione grafica dello spazio di gioco, della pallina e delle barre delle due squadre. Inoltre, mostra al giocatore di quale barra ha il controllo.

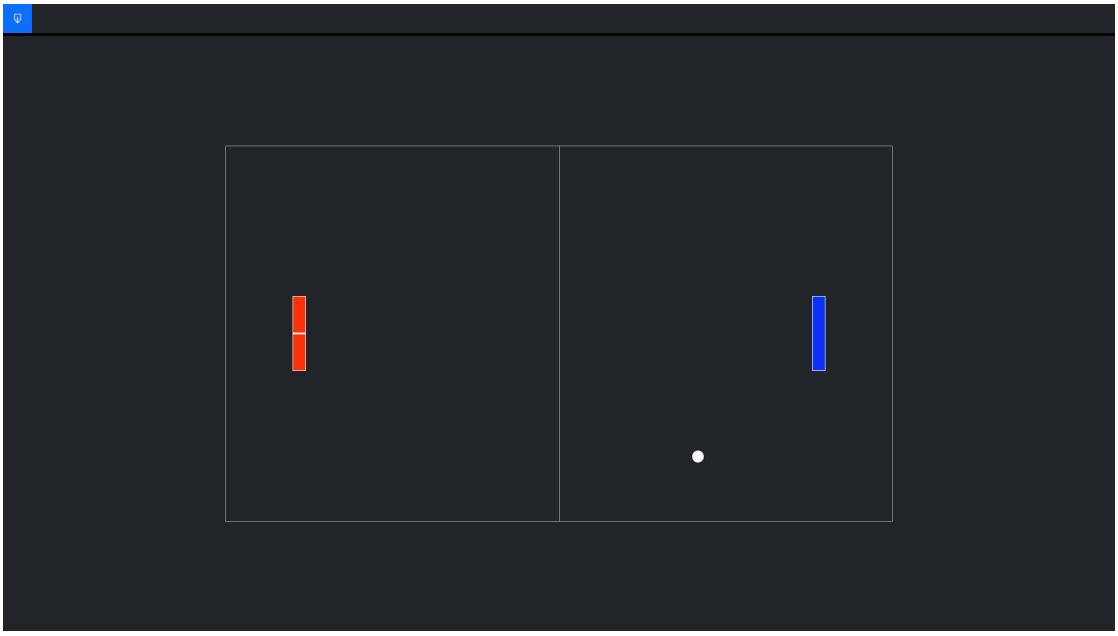
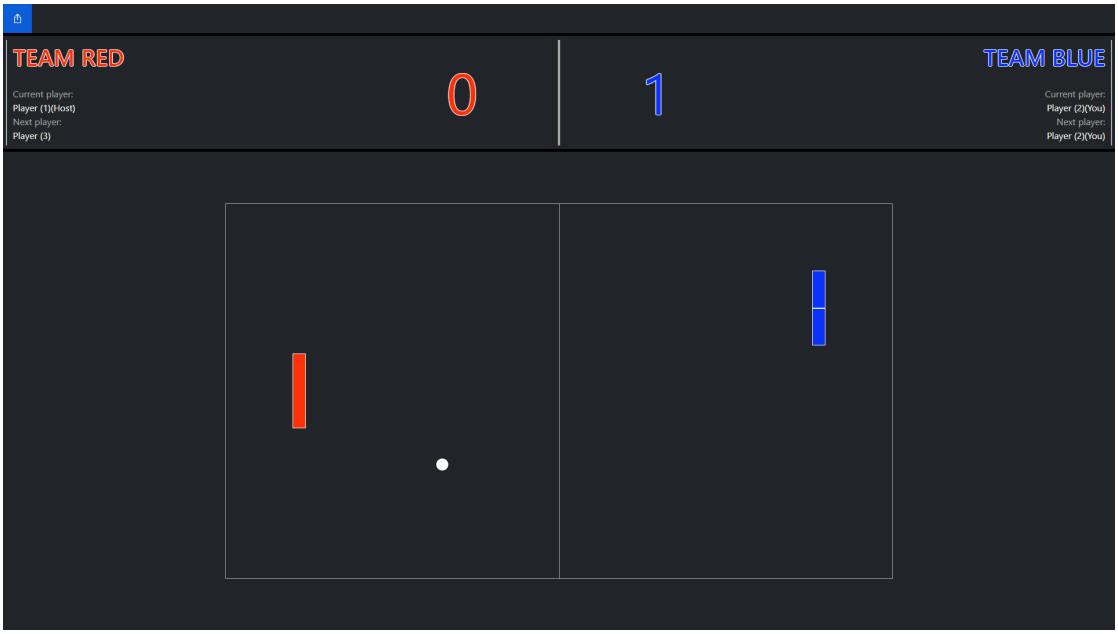


Figure 21: Screenshot della schermata di gioco dell'applicazione. Appena cominciata la partita, all'utente viene mostrata l'arena di gioco, modellata dal componente `PongArenaComponent`. In questo caso, il giocatore controlla la barra della squadra rossa, segnata da una linea bianca.

Quando il giocatore ha il controllo di una barra di gioco, l'applicazione accetta l'input dell'utente da tastiera, notificando il server di gioco delle sue intenzioni. In particolare, si accettano i seguenti input:

- Pressione dei tasti "`W`", "`D`" oppure "`ARROW_UP`": notifica il server di gioco dell'intenzione dell'utente di muovere la barra della propria squadra verso l'alto;
- Pressione dei tasti "`S`", "`A`" oppure "`ARROW_DOWN`": notifica il server di gioco dell'intenzione dell'utente di muovere la barra della propria squadra verso il basso;
- Pressione del tasto "`SPACE`": notifica il server di gioco dell'intenzione dell'utente di fermare la barra della propria squadra.

Oltre al `PongArenaComponent`, ora la schermata di gioco permette di visualizzare o nascondere lo stato delle due squadre, attraverso la pressione del pulsante nell'angolo in alto a sinistra della schermata. Lo stato di una squadra, rappresentato dalla classe `TeamStatusComponent`, contiene le informazioni relative al punteggio della squadra, al giocatore attualmente in controllo della barra della squadra e a quello a cui il controllo sarà ceduto successivamente, come illustrato in Figura 22.



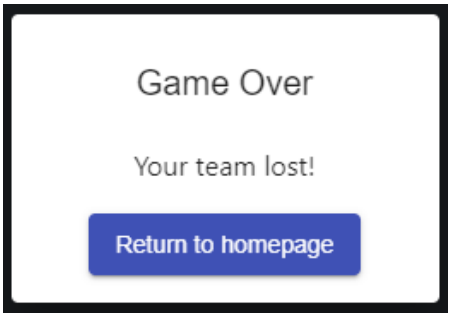
(a) Punto di vista del secondo partecipante, in controllo della barra della squadra blu.

(b) Punto di vista del terzo partecipante, in attesa di ricevere il controllo della barra della squadra rossa.

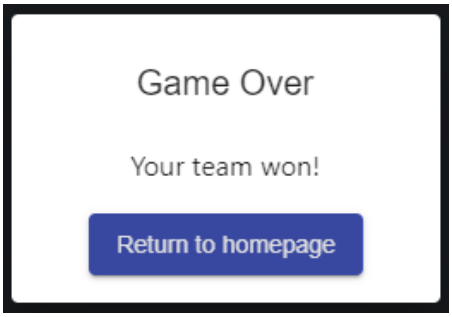
Figure 22: Screenshot della schermata di gioco dell'applicazione. Durante la partita, l'utente può visualizzare o nascondere lo stato delle due squadre.

Al termine della partita, la schermata di gioco mostra ai partecipanti un popup, contenente il risultato della partita, modellato dalla classe `GameResultPopupComponent`.

Come illustrato in Figura 23, il popup mostra all'utente se ha vinto, perso o pareggiato la partita. Inoltre, premendo il pulsante *Return to homepage*, permette di ritornare alla schermata principale dell'applicazione.



(a) Punto di vista di un partecipante che ha perso.



(b) Punto di vista di un partecipante vincitore.

Figure 23: Screenshot della schermata rappresentante il risultato di una partita, visualizzata attraverso il `GameResultPopupComponent`.

In ogni momento, all'utente potrebbe essere mostrato un popup che lo avverte di eventuali problemi di connessione al server di gioco, modellato dal componente `ErrorPopupComponent`, illustrato in Figura 24. Ciò potrebbe capitare se il server non è raggiungibile oppure se la partita a cui l'utente sta partecipando non può più proseguire, a causa della disconnessione degli altri partecipanti. Il popup permette all'utente di ritornare alla schermata principale dell'applicazione, premendo il pulsante *Return to homepage*.

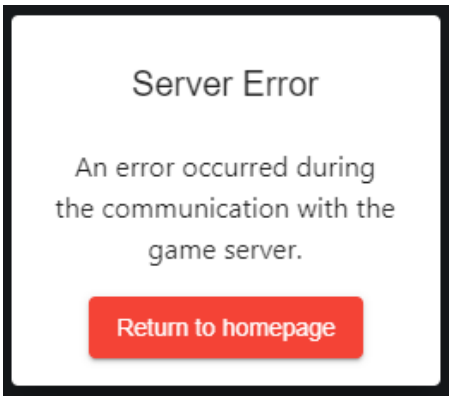


Figure 24: Screenshot della schermata di errore dell'applicazione, visualizzata attraverso l'`ErrorPopupComponent`.

Considerando i requisiti futuri del sistema, si è deciso di riadattare le schermate prodotte, rendendole visualizzabili anche tramite dispositivi mobili. A tal proposito, è stato implementato un nuovo popup per la schermata di gioco, illustrato in Figura 25. Questo popup, modellato dalla classe `OrientationPopup`, avverte l'utente di orientare

il dispositivo in *landscape mode* per visualizzare meglio la schermata di gioco. Si vuole specificare che l'applicazione non è ancora utilizzabile attraverso dispositivi mobili, ma si è solo preparata la sua interfaccia per sviluppi futuri.

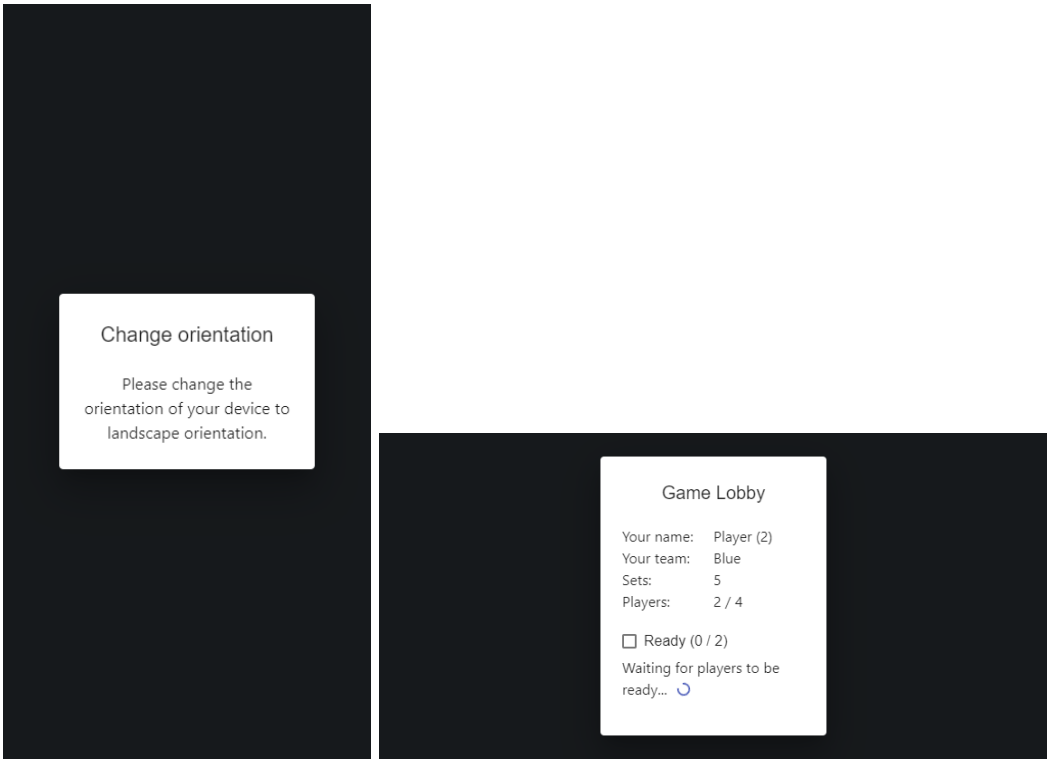


Figure 25: Screenshot della schermata di gioco dell'applicazione. In *portrait mode*, all'utente viene mostrato l'`OrientationPopup` che lo avverte di orientare diversamente il proprio dispositivo. In *landscape mode*, l'utente può correttamente visualizzare la schermata di gioco.

Passate in rassegna le varie schermate dell'applicazione, si vuole entrare nel dettaglio della comunicazione tra l'applicazione ed il server di gioco.

Al caricamento della schermata di una partita, quindi al caricamento del componente `GamePage`, l'applicazione si connette tramite Websocket al server di gioco e viene aggiornata la connessione attuale tra l'applicazione ed il server, contenuta nel singleton `Client` del Network Module. Durante tale connessione, l'applicazione riceve dal server costanti aggiornamenti sullo stato corrente della partita, che vengono salvati nel singleton `Model` del Model, contenente l'ultimo stato della partita conosciuto dall'applicazione.

In particolare, ogni volta che un messaggio del server viene ricevuto dall'applicazione, il messaggio viene deserializzato dalla classe `GameContextMessage` del Network Module, producendo una nuova istanza di `GameContext`, con la quale il `Model` viene aggiornato. In questo modo, lo stato della partita conosciuto dall'applicazione riflette quello conosciuto dal server.

Attraverso il data-binding di Angular, i componenti grafici della View che dipendono dal `Model` vengono aggiornati automaticamente ogni volta che il `Model` cambia. In questo modo, la rappresentazione grafica dello stato della partita riflette lo stato della partita conosciuto dall'applicazione, che, come appena detto, a sua volta riflette quello conosciuto dal server.

Ogni componente che gestisce l'interazione con il server di gioco e la visualizzazione dello stato corrente della partita deve estendere il componente `GameComponent`. Nello specifico, il `GamePage`, il `PongArenaComponent` ed il `TeamStatusComponent` sono tutti dei `GameComponent`. Un `GameComponent` può accedere alla connessione attuale dell'utente con il server di gioco, attraverso la classe `Client` del Network Module, e all'ultimo stato aggiornato della partita che l'utente ha ricevuto dal server, attraverso la classe `Model` del Model.

5 Validazione

La validazione del sistema è stata eseguita principalmente attraverso *test manuali* e *play-testing*, ma sono stati anche inclusi degli *test automatici* implementati tramite *JUnit* [1].

Per quanto riguarda i *test automatici*, nel Game Server sono stati realizzati alcuni test JUnit per verificare la correttezza della libreria per la gestione di forme bidimensionali, implementata appositamente per la gestione delle collisioni all'interno del gioco.

Per quanto riguarda i *test manuali*, nel Game Server sono stati realizzati alcuni test che permettono di interagire con una partita, senza che il servizio sia integrato con il Frontend ed il Backend. Questi test sono risultati molto utili nelle fasi iniziali dello sviluppo, in mancanza di un'interfaccia grafica che permettesse di eseguire del play-testing. In particolare, sono stati inclusi due test:

- `GameTester`: permette di avviare una partita, interagirvi e monitorare il suo stato tramite una console. Questo test è risultato particolarmente utile per verificare che le transizioni di stato di una partita avvenissero nelle giuste circostanze;

- `GameSimulatorTest`: permette di eseguire parallelamente un certo numero di partite. In ogni partita, partecipano due giocatori, simulati da degli agenti, modellati dalla classe `PlayerAgent`, i quali interagiscono con la partita inviando degli input casuali ad intervalli di tempo irregolari. Ad ogni iterazione del game loop, si verifica se lo stato della partita è consistente.

  Questo test ha permesso di verificare con una certa probabilità statistica che eventualmente tutte le partite terminano. Inoltre, ha permesso di osservare che, in determinate circostanze, è possibile che la pallina fuoriesca dal perimetro di gioco, anche se non è mai stato possibile riprodurre questo fenomeno durante il play-testing.

Durante le prime fasi di integrazione, si è voluto testare l'efficacia di una connessione WebSocket. Per questo è stata aggiunta una pagina di testing al servizio di Frontend, attraverso la quale è possibile valutare il throughput della connessione WebSocket con il server di gioco del sistema. Avviato il sistema, questa pagina è accessibile al seguente indirizzo del servizio di Frontend: `/test/websocket`.

Una volta integrato completamente il sistema, si è testata la sua correttezza attraverso diverse ore di *play-testing*. Durante il play-testing, si è spesso cercato di simulare situazioni particolari, come un diverso numero di partecipanti alla partita o disconnessioni frequenti, per verificare la robustezza del sistema. Si vuole però precisare che il sistema è stato testato solo in locale.

6 Deployment

Per il deployment del sistema, è necessaria una macchina in cui siano installati *git* [16], *Docker* [2] ed eventualmente *npm* [12].

Per poter eseguire il sistema è necessario:

1. *Scaricare il sorgente del sistema*, reperibile al seguente repository. Per clonare il repository è necessario avere installato git sulla propria macchina.
2. *Navigare all'interno della cartella in cui si è clonato il repository*;
3. *Avviare il sistema*, eseguendo il seguente comando:

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

   Nel caso in cui npm fosse installato sulla propria macchina, è anche possibile eseguire il sistema tramite il seguente comando:

   ```
   npm start
   ```

   Per eseguire il sistema è comunque necessario avere installato ed avviato Docker sulla propria macchina. Al primo avvio, il sistema viene anche installato, il che potrebbe richiedere dai 2 ai 5 minuti.

   Maggiori dettagli sono contenuti nel `README.md` del repository.

All'avvio del sistema, i servizi diventano disponibili ai seguenti indirizzi di default:

- Frontend: `http://127.0.0.1:4200`
- Backend: `http://127.0.0.1:4201`
- Game Server: `http://127.0.0.1:4202`

In ogni caso, è possibile configurare il sistema modificando il file `.env` all'interno della cartella in cui si è clonato il repository. Dalla configurazione è possibile modificare le porte su cui vengono esposti i servizi nella macchina ospitante e la rete in cui il servizio è raggiungibile. Infatti, è possibile esporre il servizio a una rete, assegnando alla variabile d'ambiente `SERVICE_HOSTNAME` l'indirizzo ip o l'hostname che la macchina ospitante ha in quella rete.

7 Esempi di utilizzo

Una volta avviato il sistema, gli utenti possono connettersi al servizio di Frontend e cominciare a giocare. Di seguito, si riporta un possibile scenario di utilizzo dell'applicazione.

Nello scenario preso in esempio, quattro utenti tentano di partecipare alla stessa partita di *Relay Pong*, ognuno utilizzando un browser diverso tra *Microsoft Edge*, *Chrome*, *Opera* e *Firefox*:

1. Per prima cosa, si connettono al servizio di Frontend (Figura 26);
2. Dopodiché alcuni di loro decidono di partecipare alla partita gestita dal sistema, cliccando il pulsante *Join Game* della schermata principale dell'applicazione. In questo modo, visualizzano la lobby della partita (Figura 27);
3. Ad un certo punto, i quattro utenti si trovano a partecipare tutti alla stessa partita. Dalla lobby, indicano di essere pronti a cominciare la partita e attendono che l'host la cominci (Figura 28);
4. Quando l'host preme il pulsante *Start Game*, la partita comincia e gli utenti possono giocare, controllando la barra della propria squadra a turno. Ognuno vede lo stato della partita dalla propria prospettiva (Figura 29);
5. Al termine della partita, agli utenti viene mostrato il risultato del gioco (Figura 30).

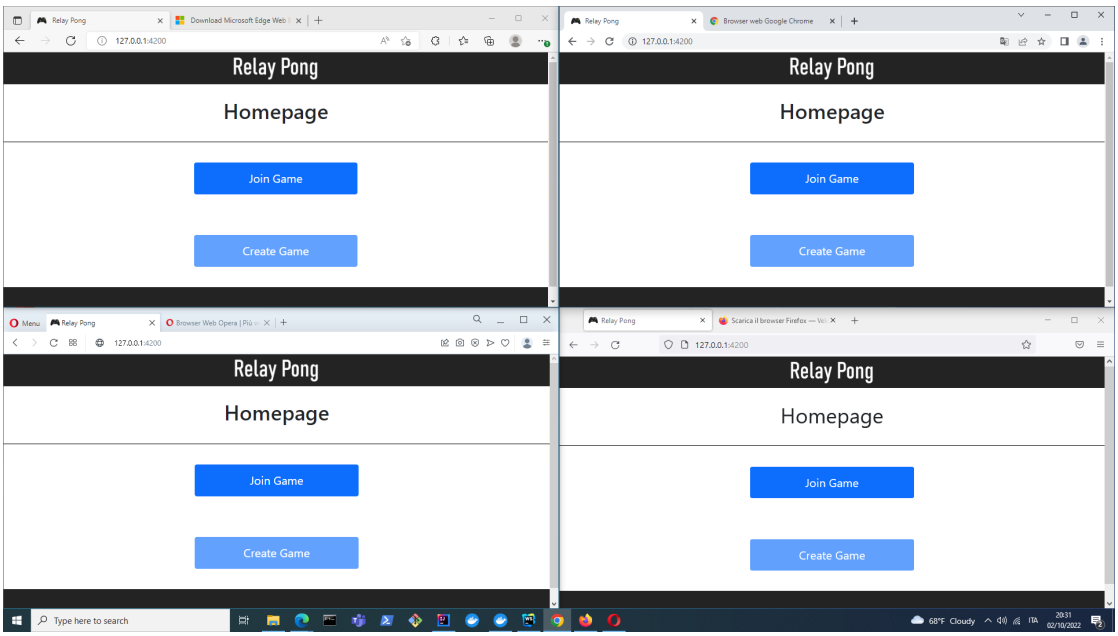


Figure 26: Quattro utenti accedono alla schermata principale dell'applicazione utilizzando browser diversi.

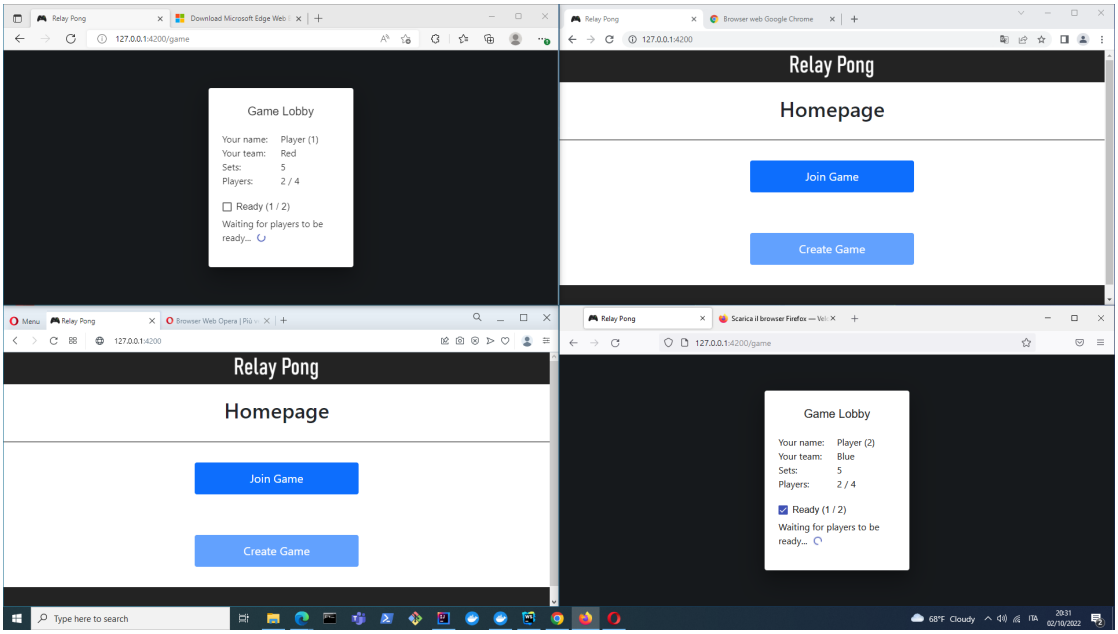


Figure 27: L'utente in alto a sinistra partecipa per primo alla partita gestita dal sistema, diventandone l'host. Dopo un po' di tempo, anche l'utente in basso a destra decide di partecipare alla stessa partita.

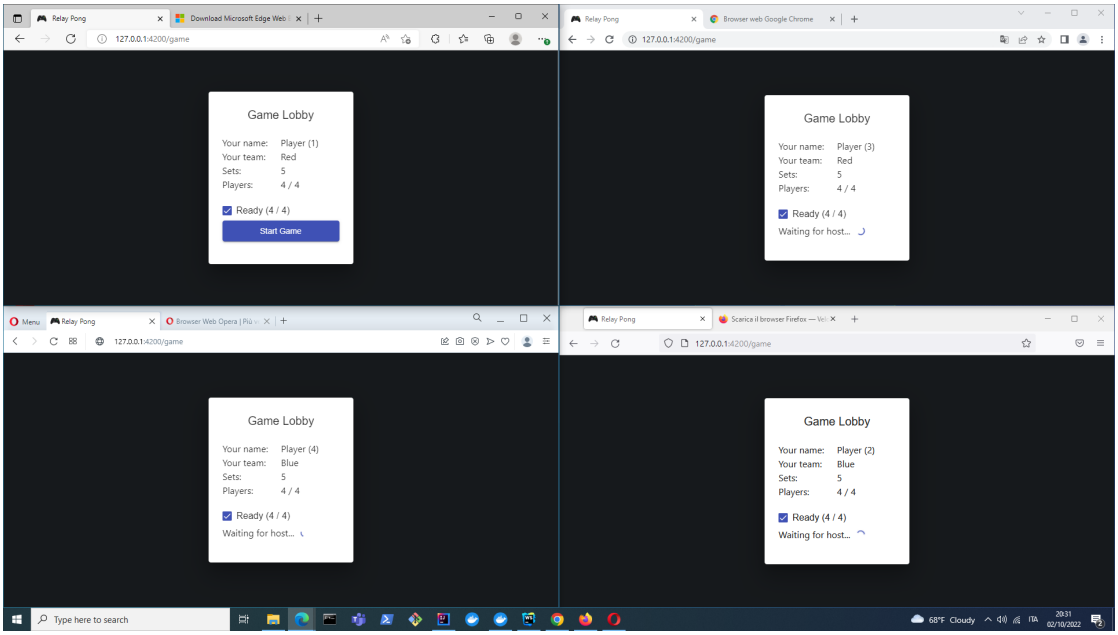


Figure 28: Ad un certo punto, tutti e quattro gli utenti sono pronti a cominciare la partita e attendono che l'host premi il pulsante *Start Game*.

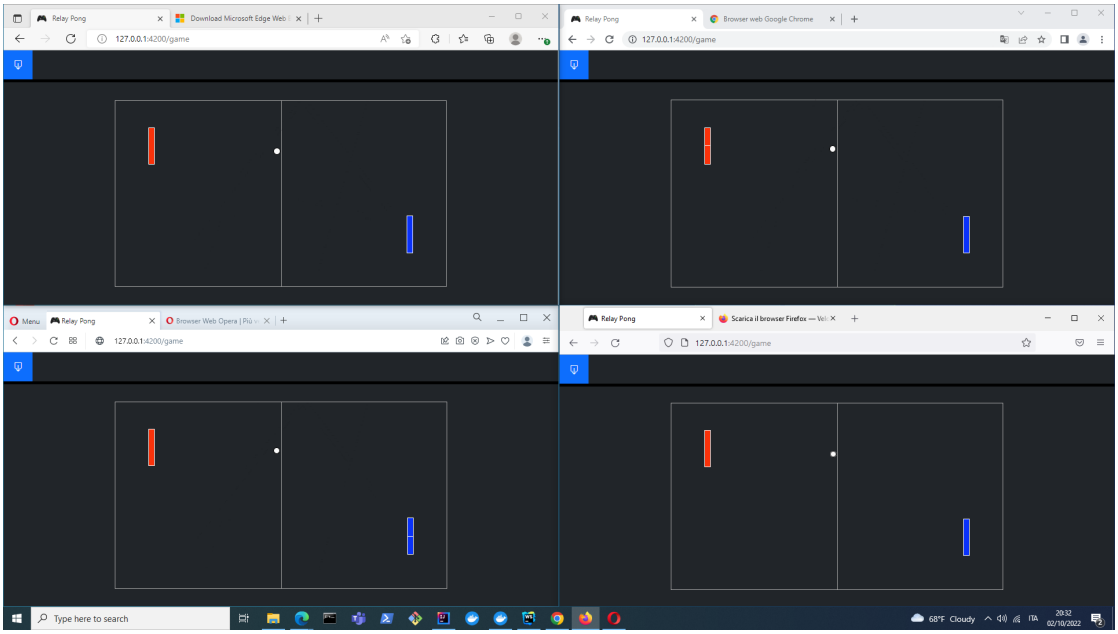


Figure 29: Durante la partita, gli utenti giocano controllando a turno la barra della propria squadra. L'utente in alto a destra controlla la barra della squadra rossa, mentre l'utente in basso a sinistra controlla quella della squadra blu.

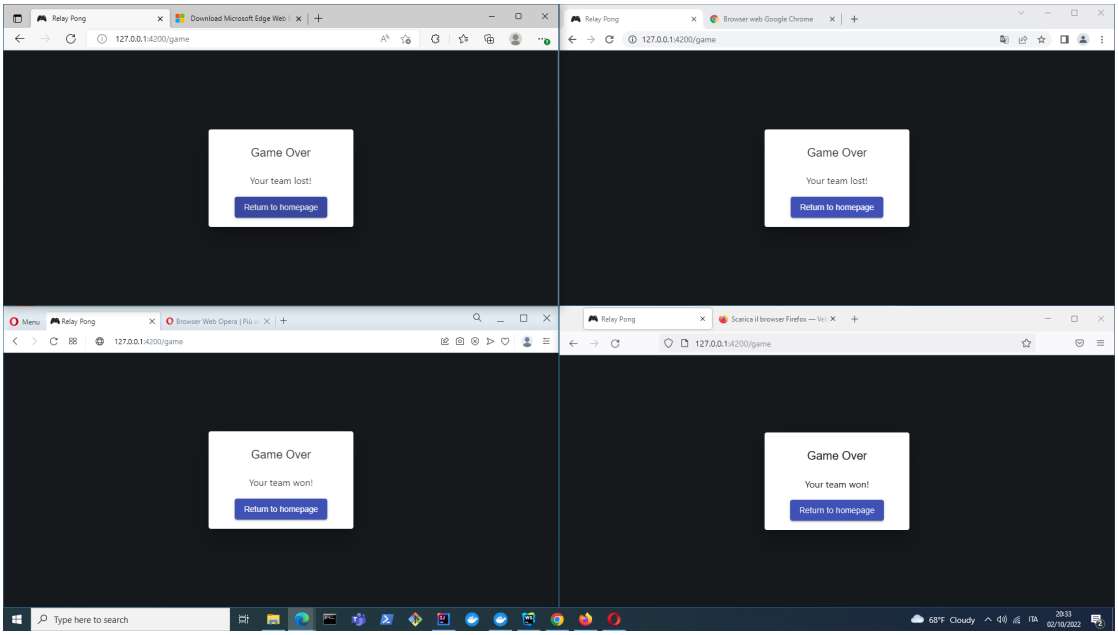


Figure 30: Al termine della partita, viene a mostrato ai partecipanti il risultato del gioco. In questo esempio, la squadra blu è risultata vincitrice.

8 Conclusioni

In questo progetto, è stato realizzato un sistema prototipale che permette di accedere al gioco *Relay Pong* attraverso un'applicazione web.

Il sistema ha soddisfatto per la maggior parte i requisiti prototipali, ma potrebbe ancora essere migliorato. In particolare, è necessario migliorare la precisione delle collisioni, perché, anche se non è stato possibile riprodurre il fenomeno durante il play-testing, dai test manuali risulta possibile che la pallina fuoriesca dal perimetro di gioco.

Inoltre, riguardo la fluidità dell'applicazione, di tanto in tanto si può notare il fenomeno di *stuttering*, per cui la visualizzazione della partita scatteggia. Questo potrebbe essere dovuto a un problema legato alla gestione della visualizzazione della partita, dell'esecuzione della partita da parte del server di gioco, oppure della comunicazione tra i giocatori ed il server di gioco. Chiaramente, per la realizzazione del prototipo, non sono state considerate molte ottimizzazioni che si sarebbe potuto mettere in pratica a questo proposito e che saranno tema di sviluppi futuri.

Infine, si ricorda che i test sul prototipo sono stati eseguiti solo in locale, il che non è sufficiente per valutare le prestazioni di un'applicazione web real-time. In futuro, sarà necessario provare ad esporre il sistema su un dominio pubblico e ad accedere all'applicazione web da remoto.

Per il resto, il prototipo ha prodotto risultati soddisfacenti, mostrando che, in linea di principio, è possibile realizzare un gioco real-time come applicazione web tramite il protocollo WebSocket. Inoltre, nel prototipo si intravede anche una buona base di partenza per lo sviluppo del sistema ultimato.

8.1 Estensioni future

Oltre ai requisiti futuri che il sistema dovrà soddisfare una volta ultimato, già descritti in fase di analisi, sarà necessario eseguire una ricerca più approfondita riguardo la comunicazione real-time tramite WebSocket. Infatti, in questo sistema, la comunicazione è stata realizzata secondo un'architettura push, in cui il server di gioco invia i dati direttamente ai giocatori. In questo modo, i giocatori non devono eseguire polling sul server e la comunicazione risulta più efficiente.

Tuttavia, non sono stati ancora presi in considerazione alcuni accorgimenti, come l'utilizzo di buffer per i messaggi lato client, per monitorare lo stato della comunicazione, o meccanismi di *back-pressure*, per evitare che il server invii messaggi ai client più velocemente di quanto i client riescano a consumarli. Questi meccanismi saranno di fondamentale importanza per lo sviluppo del sistema ultimato.

8.2 Conoscenze acquisite

Durante la realizzazione di questo progetto, è stata acquisita una certa competenza nell'utilizzo del framework *Angular* e del framework *Vert.x*. Inoltre, si è potuto sperimentare con l'integrazione di servizi diversi tramite *Docker*. Sono state anche acquisite conoscenze sull'utilizzo di *npm* e *Gradle* come strumenti di build automation. Oltre

a ciò, si è potuto sperimentare con la progettazione delle interfacce esposte da servizi che utilizzano protocolli diversi, con riferimento ad *HTTP*, in particolare *Express*, e *WebSocket*, in particolare *SockJS*.

Note di stile

Durante la stesura di questo progetto si è dato il seguente significato agli stili testuali adottati:

- *ITALIC*: evidenzia un concetto interessante;
- **BOLD**: evidenzia un concetto importante;
- `TYPEWRITER`: evidenzia un concetto che ha una corrispondenza uno a uno con un componente del sistema (es.: una classe, l'indirizzo di una risorsa...).

Riferimenti

[1] Kent Beck and Erich Gamma. JUnit 5.
URL: https://junit.org/junit5/ .
[Ultimo accesso: 02-10-2022].

[2] Inc. Docker. Docker.
URL: https://www.docker.com/ .
[Ultimo accesso: 02-10-2022].

[3] Eclipse Foundation. Vert.x Web.
URL: https://vertx.io/docs/vertx-web/java .
[Ultimo accesso: 17-09-2022].

[4] OpenJS Foundation. Express js.
URL: http://expressjs.com .
[Ultimo accesso: 17-09-2022].

[5] Google. Angular.
URL: https://angular.io/ .
[Ultimo accesso: 17-09-2022].

[6] Google. Angular - Data Binding.
URL: https://angular.io/guide/binding-syntax .
[Ultimo accesso: 24-09-2022].

[7] Google. Google Chrome Homepage.
URL: https://www.google.it/intl/it/chrome/ .
[Ultimo accesso: 17-09-2022].

[8] Red Hat. API REST e RESTful.
URL: https://www.redhat.com/it/topics/api/what-is-a-rest-api .
[Ultimo accesso: 18-09-2022].

[9] Red Hat. Stateful vs Stateless.
URL: https://www.redhat.com/it/topics/cloud-native-apps/stateful-vs-stateless .
[Ultimo accesso: 17-09-2022].

[10] ECMA International. JSON - JavaScript Object Notation.
URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-404 .
[Ultimo accesso: 19-09-2022].

[11] Microsoft. Typescript.
URL: https://www.typescriptlang.org .
[Ultimo accesso: 17-09-2022].

[12] npm Inc. npm.
URL: https://www.npmjs.com/ .
[Ultimo accesso: 02-10-2022].

[13] Robert Nystrom. Game Programming Patterns - Game Loop.
URL: https://gameprogrammingpatterns.com/game-loop.html .
[Ultimo accesso: 01-10-2022].

[14] Openbase. SockJS.
URL: https://openbase.com/js/sockjs/documentation .
[Ultimo accesso: 17-09-2022].

[15] Oracle. Java.
URL: https://www.java.com/it .
[Ultimo accesso: 17-09-2022].

[16] Linus Torvalds and Junio Hamano. Git - Version Control System.
URL: https://git-scm.com/ .
[Ultimo accesso: 02-10-2022].

[17] Wikipedia. Model View View-Model.
URL: https://en.wikipedia.org/wiki/Model-view-viewmodel .
[Ultimo accesso: 17-09-2022].

[18] Wikipedia. Pong.
URL: https://it.wikipedia.org/wiki/Pong .
[Ultimo accesso: 16-09-2022].

[19] Wikipedia. Single Page Application.
URL: https://it.wikipedia.org/wiki/Single-page_application .
[Ultimo accesso: 17-09-2022].

[20] Wireshark. HTTP Protocol.
URL: https://wiki.wireshark.org/Hyper_Text_Transfer_Protocol .
[Ultimo accesso: 17-09-2022].

[21] Wireshark. WebSocket Protocol.
URL: https://wiki.wireshark.org/WebSocket .
[Ultimo accesso: 17-09-2022].