

Joking Hazard Online

Sistemi Distribuiti

Riccardo Sanità - 0001084309 {riccardo.sanita@unibo.it}

17 Agosto 2023

Indice

0.1	Abstract	2
0.2	Obiettivi	2
0.2.1	Scenarios	3
0.2.2	Politiche di autovalutazione	5
0.3	Requisiti	5
0.3.1	Requisiti funzionali	5
0.3.2	Requisiti non funzionali	6
0.4	Design	6
0.4.1	Design dell'architettura del sistema	6
0.5	Tecnologie	7
0.5.1	Socket.io	7
0.5.2	Axios	8
0.5.3	Mongoose	9
0.5.4	Express	10
0.5.5	Libreire di React	10
0.6	Struttura	11
0.7	Comportamento	12
0.8	Interazioni	13
0.8.1	Creazione nuova partita	14
0.8.2	Partecipazione ad una partita pubblica	15
0.8.3	"Round" di una partita	16
0.9	Validazione	17
0.10	Deployment	18
0.11	Esempi d'uso	19
0.12	Conclusioni	25
0.12.1	Riflessioni finali	25
0.12.2	Possibili miglioramenti	25
0.12.3	Che cosa ho imparato	25

0.1 Abstract

Il progetto è concepito con l'obiettivo di sviluppare una versione multiplayer online del noto gioco da tavolo Joking-Hazard. In ogni round, un giocatore designato come giudice seleziona una carta dal mazzo e la pone in vista di tutti. Ogni carta raffigura una vignetta di un fumetto, e ogni giocatore, attingendo dalle 7 carte in suo possesso, sceglie quella che ritiene possa completare la striscia fumettistica nel modo più divertente.

Le scelte dei giocatori vengono raccolte coperte, mescolate e presentate al giudice, il quale, senza sapere chi ha presentato quale carta, sceglie quella che ritiene sia la conclusione migliore per la striscia fumettistica. Il giocatore la cui carta viene scelta guadagna punti. Nei round successivi, il ruolo del giudice passa al giocatore successivo in senso orario. Il vincitore è il primo a raggiungere il punteggio massimo stabilito dai partecipanti.

Un elemento cruciale di questo progetto è l'interazione tra il front-end e il back-end. Grazie alla comunicazione con il server, l'interfaccia di ogni utente viene aggiornata in tempo reale con ogni modifica. La chat costituisce il canale di comunicazione tra i giocatori, in cui vengono pubblicate dagli utenti informazioni quali il vincitore del round, possibili spiegazioni relative alle decisioni prese, qual'è il punteggio massimo e molto altro.

0.2 Obiettivi

E' stato condotto un focus group con l'obiettivo di raccogliere informazioni dettagliate sui requisiti degli utenti per la creazione di una versione multiplayer online del popolare gioco da tavolo Joking-Hazard. Il focus group ha coinvolto un campione di dieci utenti, accuratamente selezionati per rappresentare una varietà di interessi, competenze e livelli di esperienza con i giochi online e da tavolo. Durante il focus group sono state poste una serie di domande strategiche ai partecipanti, in modo da avere un quadro più completo dei loro desideri e aspettative. Alcune di queste domande includono:

1. **Esperienza Precedente:** Qual è la tua esperienza con il gioco da tavolo originale, Joking Hazard? Hai mai giocato a versioni online di giochi da tavolo in precedenza?
2. **Funzionalità del Gioco:** Quali funzionalità ti aspetti di trovare in un gioco multiplayer online come questo? Ci sono delle caratteristiche specifiche del gioco da tavolo che ritieni siano importanti da includere nella versione online?
3. **Interazione tra Giocatori:** Come preferisci interagire con gli altri giocatori durante il gioco? Come ritieni che la chat del gioco possa migliorare la tua esperienza di gioco?
4. **Interfaccia e Usabilità:** Cosa ritieni importante in un'interfaccia utente per un gioco online? Quali sono le principali difficoltà che hai riscontrato

in passato nell'usare giochi online e come potremmo evitarle in questo progetto?

5. **Aggiornamenti e Monitoraggio del Progresso:** Come ti piacerebbe essere aggiornato sul tuo progresso nel gioco? Preferiresti avere un riepilogo a fine round o preferisci essere aggiornato in tempo reale?

Durante la discussione, sono emersi vari temi comuni che hanno aiutato a definire i requisiti principali del sistema:

- **Interazione in Tempo Reale:** Gli utenti hanno espresso il desiderio di avere un'interazione in tempo reale con il gioco e con gli altri giocatori. Hanno richiesto che le modifiche del gioco (come la scelta delle carte o il cambio del giudice) siano visualizzate in tempo reale per tutti i partecipanti.
- **Facilità d'Uso:** Un altro tema importante è stato la facilità d'uso. Gli utenti vogliono che l'interfaccia del gioco sia intuitiva e facile da navigare, indipendentemente dal dispositivo utilizzato per accedere al gioco.
- **Comunicazione tra Giocatori:** Il focus group ha espresso il bisogno di un metodo efficace di comunicazione tra i giocatori. Hanno suggerito l'inclusione di una chat di gioco, dove possono essere condivise informazioni come il vincitore del round, spiegazioni sulle decisioni prese e il punteggio massimo.
- **Aggiornamenti sul Progresso del Gioco:** Gli utenti hanno mostrato interesse per avere un resoconto dettagliato del progresso del gioco. Hanno richiesto la presenza di una dashboard dove poter monitorare i loro punteggi e il progresso verso il punteggio massimo.

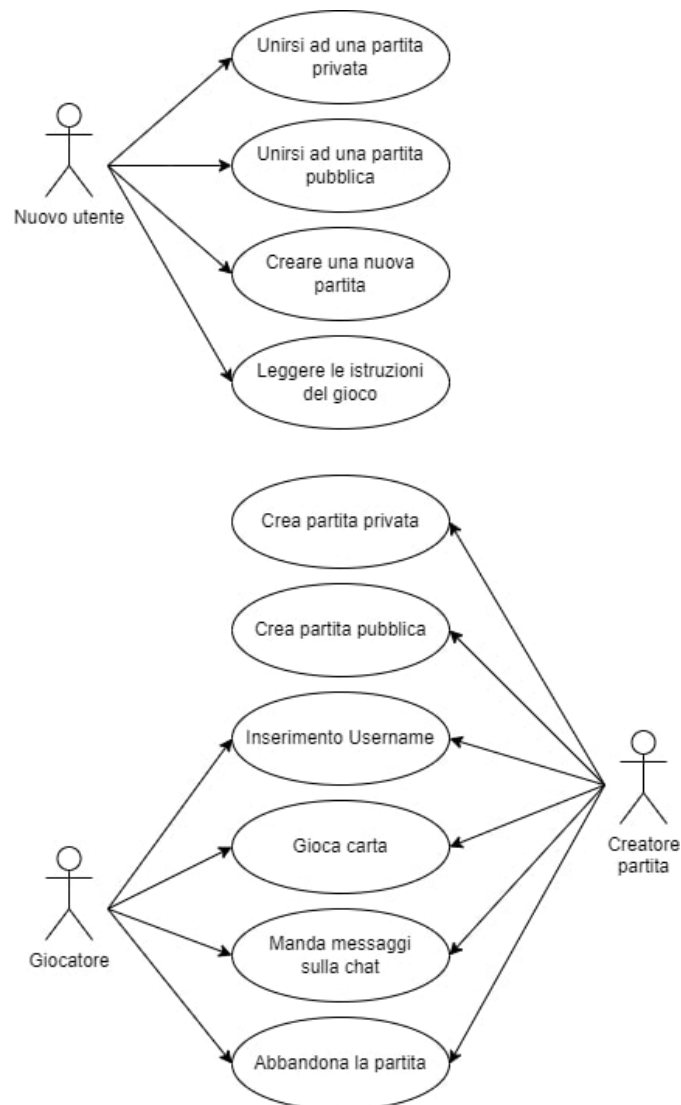
Le informazioni raccolte durante il focus group sono state fondamentali nella fase di progettazione e sviluppo del gioco.

0.2.1 Scenarios

Di seguito viene riportato il Diagramma dei Casi d'Uso (Figura 1) che rappresenta tutti gli scenari del gioco dal punto di vista dell'utente. In particolare abbiamo l'attore base (Utente) che può disconnettersi dal gioco in qualsiasi momento. Tutti gli altri attori raffigurano sempre l'utente in momenti diversi:

- All'avvio dell'applicazione l'Utente può scegliere se partecipare ad una partita privata, ad una pubblica, iniziare una nuova partita oppure leggere le istruzioni del gioco. Le partite pubbliche sono visibili in una tabella che mostra il numero dei giocatori e il codice necessario per partecipare a tale partita. Ad una partita privata invece è possibile accedervi solo se uno dei partecipanti ha condiviso il codice con il nuovo utente.

- Una volta entrato in una partita l'Utente deve inserire il suo nickname. L'utente, diventato a questo punto Giocatore, può copiare il link della partita per condividerlo con i suoi amici.
- Una volta in partita il Giocatore può procedere giocando le sue carte secondo le regole, può comunicare con gli altri giocatori attraverso la chat e infine può anche abbandonare la partita. Se ad abbandonare la partita è l'ultimo Giocatore rimasto allora la partita verrà eliminata.



0.2.2 Politiche di autovalutazione

Il sistema deve garantire la giocabilità della partita:

- Si devono seguire le regole per completare una sessione di gioco.
- Se un giocatore che non è l'ultimo rimasto si disconnette si disconnette la partita deve continuare. Nel caso in cui la partita fosse ancora in e il giocatore disconnesso si volesse riconnettere il server farà in modo di mostrare al giocatore le carte che aveva al momento della disconnessione
- Se l'ultimo giocatore si disconnette allora la partita deve terminare.
- L'elenco delle partite aperte/chiusa deve rimanere aggiornato. Inoltre, devono essere costantemente sincronizzati i client dei giocatori all'interno della stessa partita, in modo da garantire la coerenza del gioco.
- L'obiettivo principale del software sviluppato è garantire una solida capacità di adattamento alle dimensioni, una fluidità accettabile nell'esperienza di gioco e una sufficiente resilienza agli errori. In modo specifico, il servizio web necessita di gestire in modo efficace un grande numero di utenti simultaneamente, mentre l'esecuzione del gioco deve risultare scorrevole sul dispositivo dell'utente, evitando inconvenienti legati alle prestazioni sia grafiche che funzionali. Inoltre, è essenziale che il sistema non si blocchi in caso di eventuali problemi.

0.3 Requisiti

0.3.1 Requisiti funzionali

- **Gestione Utenti:** Il sistema deve essere in grado di gestire l'iscrizione e l'accesso e l'uscita degli utenti.
- **Creazione della Partita:** Il sistema deve permettere agli utenti di creare nuove partite, invitare amici, unirsi a partite esistenti o partecipare a partite pubbliche.
- **Gestione delle Partite:** Il sistema deve gestire le regole del gioco, monitorare il progresso della partita.
- **Interazione in Tempo Reale:** Il sistema deve fornire aggiornamenti in tempo reale sullo stato della partita a tutti i partecipanti, comprese le mosse dei giocatori e il cambio del mazzo di carte.
- **Chat Integrata:** Il sistema deve fornire una chat integrata per permettere ai giocatori di comunicare tra loro durante la partita.
- **Gestione del Mazzo di Carte:** Il sistema deve gestire la distribuzione delle carte.

- **Gestione dei Punteggi:** Il sistema deve tenere traccia dei punteggi dei giocatori durante la partita e aggiornare una classifica globale degli utenti.
- **Interfaccia di Gioco:** Il sistema deve fornire un'interfaccia utente intuitiva e reattiva che visualizza chiaramente lo stato del gioco, le carte dei giocatori e le mosse disponibili.
- **Tutorial e Regole:** Il sistema deve fornire un tutorial per i nuovi utenti e una sezione dedicata alle regole del gioco.

0.3.2 Requisiti non funzionali

- **Prestazioni:** Il sistema deve garantire tempi di risposta rapidi, indipendentemente dal numero di utenti che giocano contemporaneamente.
- **Scalabilità:** Il sistema deve essere in grado di gestire un aumento nel numero di utenti senza degradare le prestazioni o l'affidabilità.
- **Affidabilità:** Il sistema deve avere un basso tasso di errori e un'elevata tolleranza ai guasti. In caso di guasto, dovrebbe essere in grado di riprendersi rapidamente senza perdita di dati.
- **Usabilità:** Il sistema deve essere facile da usare, con un'interfaccia utente intuitiva che richiede il minimo di formazione per gli utenti.
- **Manutenibilità:** Il sistema deve essere facile da modificare e aggiornare. Deve essere ben commentato e progettato in modo da facilitare l'aggiunta di nuove funzionalità o l'adattamento a cambiamenti futuri.
- **Accessibilità:** Il sistema deve essere progettato in modo tale da essere utilizzabile da persone con varie abilità, rispettando i principi dell'accessibilità web.

0.4 Design

0.4.1 Design dell'architettura del sistema

L'applicazione è stata sviluppata utilizzando lo stack MERN, il quale si compone delle seguenti tecnologie, come suggerito dal suo nome:

- MongoDB
- Express
- React
- Node.js

Lo stack MERN è una delle tante varianti dello stack MEAN, dove la tradizionale struttura frontend Angular.js è stata rimpiazzata da React.js. Questo ha fornito la possibilità di creare con facilità una struttura a tre livelli (frontend, backend, database) utilizzando esclusivamente Javascript e JSON. Diverse tecnologie sono state implementate nel progetto, con variazioni da frontend a backend. Per il lato frontend, sono stati utilizzati Axios e React, per il lato backend, si è fatto uso di Express e Moongose. Infine, come database è stato scelto di utilizzare un database MongoDB Atlas che ha facilitato notevolmente l'archiviazione, la manipolazione e la rappresentazione dei dati ad ogni livello dell'applicazione.

In questo progetto, è stata adoperata una caratteristica recente di React, conosciuta come Hook, disponibile dalla versione 16.8 in poi. L'idea dietro gli Hook è di rendere i componenti più gestibili, evitando la necessità di controllare il loro stato e i relativi effetti collaterali, il che contribuisce a prevenire l'insorgere di errori e inconsistenze. Gli Hook, in pratica, sono funzioni che permettono di "agganciarsi" all'ambiente di React e al flusso di vita dei componenti, senza la necessità di utilizzare classi. Nell'ambito di questo progetto, gli Hook che abbiamo più spesso utilizzato sono:

- `useState`: che dà la possibilità di associare uno stato a un componente React;
- `useEffect`: che consente di gestire gli effetti che agiscono sui componenti funzionali.

0.5 Tecnologie

0.5.1 Socket.io

SocketIO è una libreria JavaScript specializzata per le applicazioni web in tempo reale. Questa libreria facilita una comunicazione bidirezionale e sincrona tra il client e il server. È composta da due componenti che condividono la stessa API: una componente funziona sul lato client, mentre l'altra opera sul lato server come libreria per Node.js.

Questa libreria è particolarmente utile per gestire funzionalità di chat e instant messaging, oltre a permettere l'invio di notifiche push. Il principio fondamentale su cui si basa SocketIO è che, quando si verifica un evento, il server lo invia immediatamente ai client connessi che potrebbero essere interessati a tale evento.

L'architettura di alto livello di SocketIO può essere riassunta come segue: sia il server che il client utilizzano le WebSockets per comunicare inviando messaggi. Questi messaggi, o eventi, vengono poi intercettati dai listener registrati.

In questo progetto queste meccaniche sono state utilizzate per la comunicazione a tutti gli utenti di alcuni eventi come: il trascinarsi di una carta, la scelta di una carta, i messaggi nella chat, l'abbandono della partita da parte di un giocatore ecc. In questo modo nel momento in cui un giocatore compie

un'azione ogni altro giocatore della stessa partita riceve degli aggiornamenti in tempo reale. Di seguito è riportato una parte di codice che usa le funzionalità di Socket.io

```
1      this.socket.on(
2        'get initialCards for game',
3        ({ whiteCards = [], blackCards = [] }) => {
4          this.setState({
5            whiteCards,
6            blackCards,
7          });
8        }
9      );
10
11      this.socket.on('disconnect', () => {
12        window.location.reload();
13      });
14
15      // quando un giocatore cambia il proprio nome, aggiorna lo
16      // stato dei giocatori con il nuovo nome
17      this.socket.on('name change', (players) => {
18        console.log({ players });
19        this.setState({ players });
20      });
21
22      // quando un giocatore si disconnette, rimuovilo dallo stato
23      this.socket.on('user disconnected', (players) => {
24        this.setState({ players });
25      });
```

0.5.2 Axios

Axios è una libreria HTTP basata su promise per JavaScript che consente di eseguire richieste HTTP asincrone dal browser e dal server. È particolarmente utile nello stack MERN (MongoDB, Express.js, React.js, Node.js), dove Axios viene utilizzato per facilitare la comunicazione tra il front-end e il back-end.

Nel contesto di un'applicazione MERN, Axios viene utilizzato per inviare richieste HTTP dal front-end React.js al back-end Node.js. Queste richieste possono includere operazioni CRUD (Create, Read, Update, Delete) su dati memorizzati in un database MongoDB.

Quando Axios invia una richiesta, restituisce una promise. Questa promise rappresenta l'operazione HTTP in sospeso. Una volta che la richiesta HTTP è stata completata, la promise viene risolta con il valore della risposta HTTP. Se si verifica un errore durante la richiesta HTTP, la promise viene rifiutata con l'errore.

Questo modello di promise consente di scrivere codice asincrono che può essere gestito in modo simile al codice sincrono. In particolare, Axios fornisce metodi per gestire il successo (utilizzando il metodo `then`) e l'errore (utilizzando il metodo `catch`) di una promise. Di seguito è riportato un esempio dell'utilizzo di Axios

```

1 // conferma da parte del server che si   entrati nella stanza
   giusta
2   this.socket.on('joined a room', (theRoom) => {
3     console.log({ theRoom });
4
5     // se le carte sono gi   state impostate, non chiamare
       questo endpoint
6     if (!this.state.whiteCards.length && !this.state.blackCards
       .length) {
7       axios
8         .post(`${SERVER_URL}/api/getInitialCards`, {
9           roomId: this.roomId,
10         })
11         .then((res) => {
12           if (!res.data) {
13             return;
14           }
15
16           const {
17             blackCards: newBlackCards,
18             whiteCards: newWhiteCards,
19           } = res.data;
20
21           this.socket.emit('set initialCards for game', {
22             whiteCards: newWhiteCards,
23             blackCards: newBlackCards,
24           });
25         });
26
27     //controlla se la stanza   stata segnata come privata
       dal primo utente connesso
28     const isPrivate = queryString.parse(this.props.location.
       search).private;
29
30     if (isPrivate === '1') {
31       this.socket.emit('set game as private');
32     }
33   }
34 });

```

0.5.3 Mongoose

Mongoose è una libreria per Node.js che fornisce un'interfaccia di alto livello per interagire con MongoDB. È un Object Data Modeling (ODM) library che gestisce la connessione tra Node.js e MongoDB, facilitando la creazione, la lettura, l'aggiornamento e l'eliminazione di dati nel database.

Mongoose fornisce un modo semplice e coerente per definire schemi per i dati. Uno schema Mongoose definisce la struttura dei documenti all'interno di una collezione MongoDB e definisce il tipo di dati per ciascun campo nel documento. Mongoose fornisce anche una serie di funzioni utili per validare i dati prima che vengano salvati nel database. Nell'immagine di seguito è riportato lo schema di una carta del gioco.

```

1 const mongoose = require('mongoose');

```

```

2
3 const cardSchema = new mongoose.Schema({
4     type: { type: String, required: true },
5     image: { type: String, required: true },
6 },
7 {
8     timestamps: true,
9 }
10 )
11
12 // Card Class
13 const Cards = mongoose.model('Cards', cardSchema);

```

Inoltre, Mongoose supporta le query MongoDB, permettendo di eseguire operazioni complesse sui dati come la ricerca, l'ordinamento e la paginazione. Mongoose traduce queste query in operazioni MongoDB native. Infine, Mongoose supporta le funzionalità di middleware, consentendo di eseguire funzioni prima o dopo che determinate operazioni vengono eseguite sul database.

0.5.4 Express

Express.js, o semplicemente Express, è un framework per applicazioni web Node.js che fornisce un insieme robusto di funzionalità per le applicazioni web e mobile. Nel contesto dello stack MERN (MongoDB, Express.js, React.js, Node.js), Express svolge un ruolo fondamentale nel gestire le richieste HTTP e nel mediare la comunicazione tra il front-end e il database.

Express semplifica lo sviluppo del lato server con una serie di funzionalità, tra cui:

- **Middleware:** Express utilizza il concetto di middleware per gestire le richieste e le risposte all'interno dell'applicazione. I middleware sono funzioni che hanno accesso all'oggetto di richiesta (req), all'oggetto di risposta (res) e alla funzione next nel ciclo di richiesta/risposta dell'applicazione.
- **Routing:** Express fornisce un modo semplice per definire le rotte (i percorsi URL) per la tua applicazione. Ogni rotta può avere una o più funzioni di gestione, che vengono eseguite quando la rotta viene abbinata.
- **Error Handling:** Express fornisce un meccanismo robusto per la gestione degli errori, con supporto per middleware di gestione degli errori personalizzati.

0.5.5 Libreria di React

React è una libreria JavaScript per la creazione di interfacce utente. Oltre alla libreria principale, ci sono molte librerie aggiuntive che possono essere utilizzate con React per estendere le sue funzionalità. Tra queste, ci sono react-dnd, react-dnd-touch-backend e styled-components.

- **react-dnd:** Questa è una libreria di trascinamento e rilascio (drag and drop) flessibile e di alto livello per React. Con react-dnd, è possibile creare componenti React complessi che possono essere trascinati e rilasciati, con un controllo completo su come i dati vengono gestiti e come l'interfaccia utente risponde a tali interazioni. react-dnd utilizza un modello basato su decoratori e un sistema di backend pluggable per gestire vari scenari di trascinamento e rilascio.
- **react-dnd-touch-backend:** Questo è un backend per react-dnd che consente il trascinamento e rilascio su dispositivi touch. Mentre react-dnd fornisce l'API di alto livello per il trascinamento e rilascio, react-dnd-touch-backend fornisce l'implementazione specifica per gestire gli eventi touch. Questo rende react-dnd utilizzabile su dispositivi mobili e touchscreen.
- **styled-components:** Questa è una libreria che consente di utilizzare i componenti stilizzati in React. Con styled-components, è possibile scrivere codice CSS reale nel JavaScript, eliminando il mapping tra componenti e stili. Questo consente di avere un codice più prevedibile e facile da gestire. Ogni componente stilizzato ha uno stile isolato, il che significa che i cambiamenti di stile a un componente non influenzeranno altri componenti.

0.6 Struttura

Il piano si divide in due parti principali: il "Client" e il "Server".

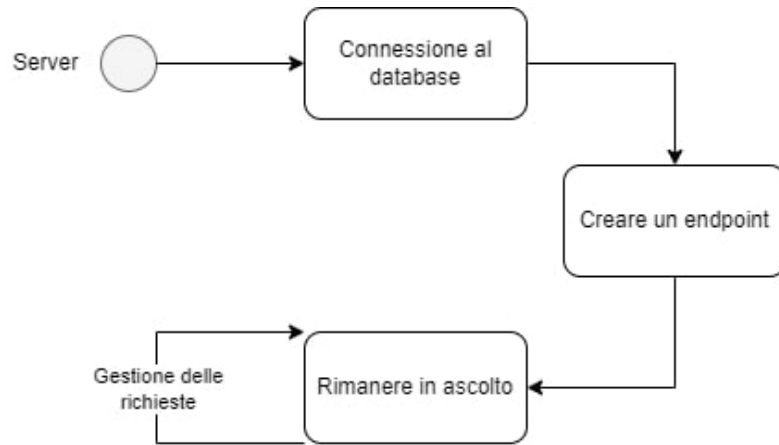
Il lato del Giocatore è dove si trova tutto ciò che riguarda l'uso dell'app e l'interazione con il gioco. Le parti diverse di questo lato includono: le risorse come immagini ed icone, i componenti da renderizzare, le variabili salvate localmente, le diverse route per visualizzare una pagina specifica e le funzioni utilizzate dal client.

Il lato del Server è dove viene fatta la parte dietro le quinte che gestisce i giocatori, le partite e le cose diverse che possono succedere durante il gioco. Le parti principali di questo lato sono: la connessione al server e l'avvio dell'applicazione e la gestione dei messaggi che i client inviano al server.

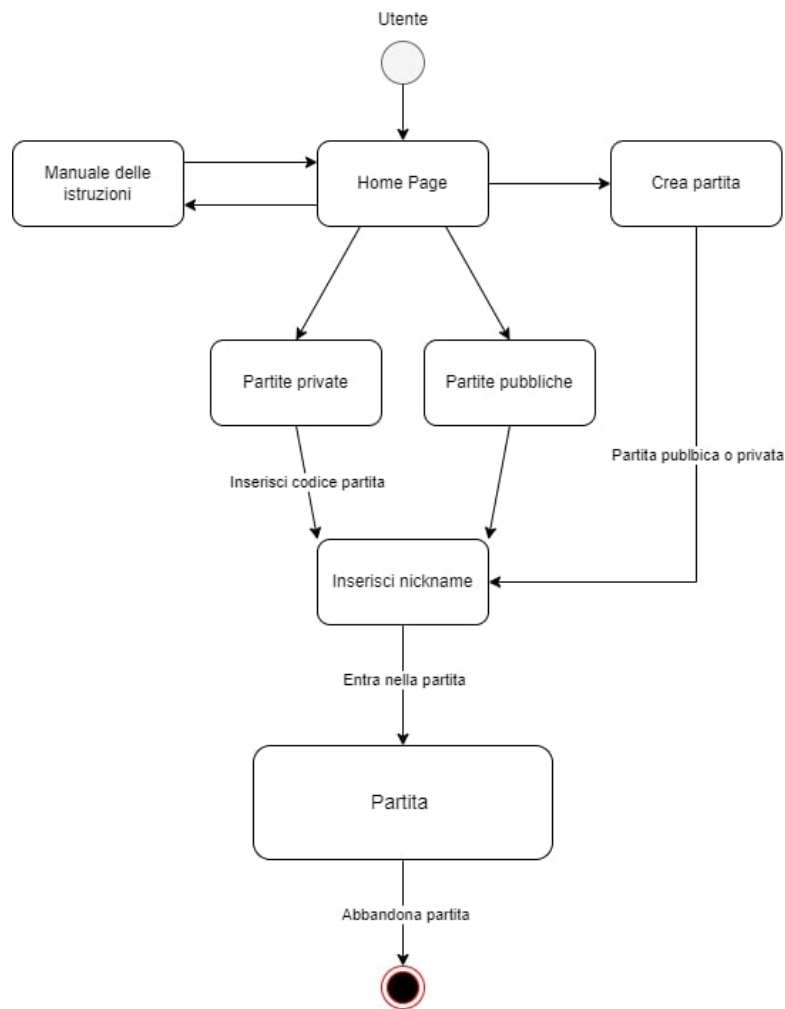
Per riassumere, il progetto è stato strutturato in modo coerente e suddiviso in parti distinte, il modulo Cliente si occupa della logica dell'applicazione mentre il modulo Server gestisce la logica delle partite. Questa divisione in parti ha semplificato lo sviluppo e la gestione del codice, assicurando l'alta qualità del progetto e la sua espandibilità.

0.7 Comportamento

Quello riportato qui sotto è il diagramma degli stati che schematizza il comportamento del server.



Di seguito invece, viene mostrato il diagramma degli stati dalla prospettiva di un utente, dal momento in cui avvia l'applicazione al momento in cui viene avviato un match. L'applicazione può terminare in qualsiasi momento nel momento in cui l'utente decide di chiudere la stessa.

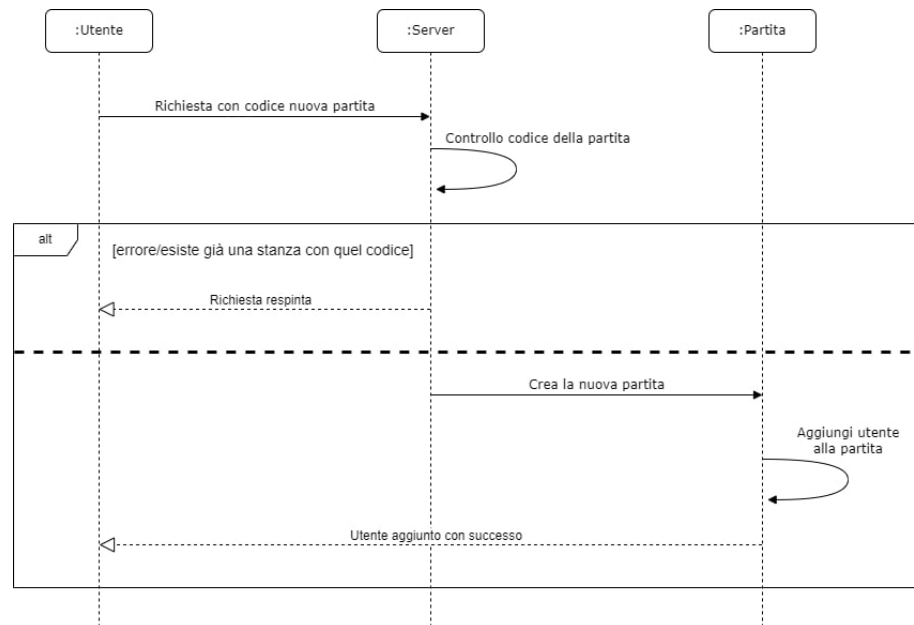


0.8 Interazioni

Le macro-interazioni principali all'interno del progetto sono: la creazione di una partita, la partecipazione ad una partita pubblica già esistente e un "round" di una partita. Per spiegare in modo semplice queste interazioni userò dei Diagrammi di Sequenza.

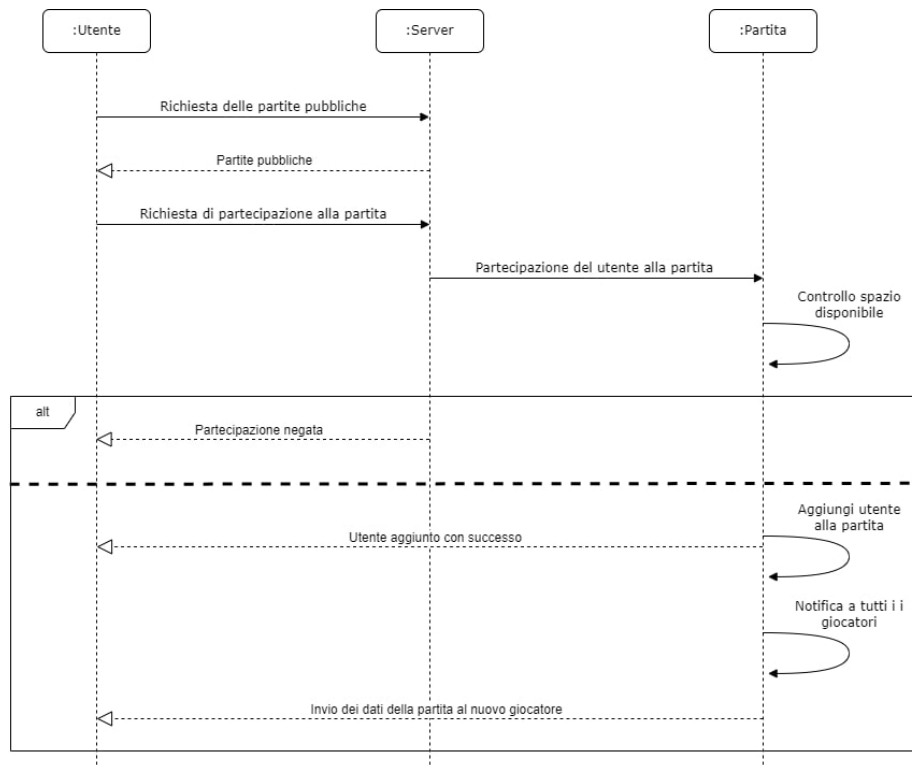
0.8.1 Creazione nuova partita

Nel momento della creazione di una nuova partita viene creato un codice, quando il server riceve una richiesta di creazione di una partita controlla che il codice della partita non sia già in uso. Se il codice della partita viene accettato allora il server crea una nuova partita e carica al suo interno le carte prendendole dal database.



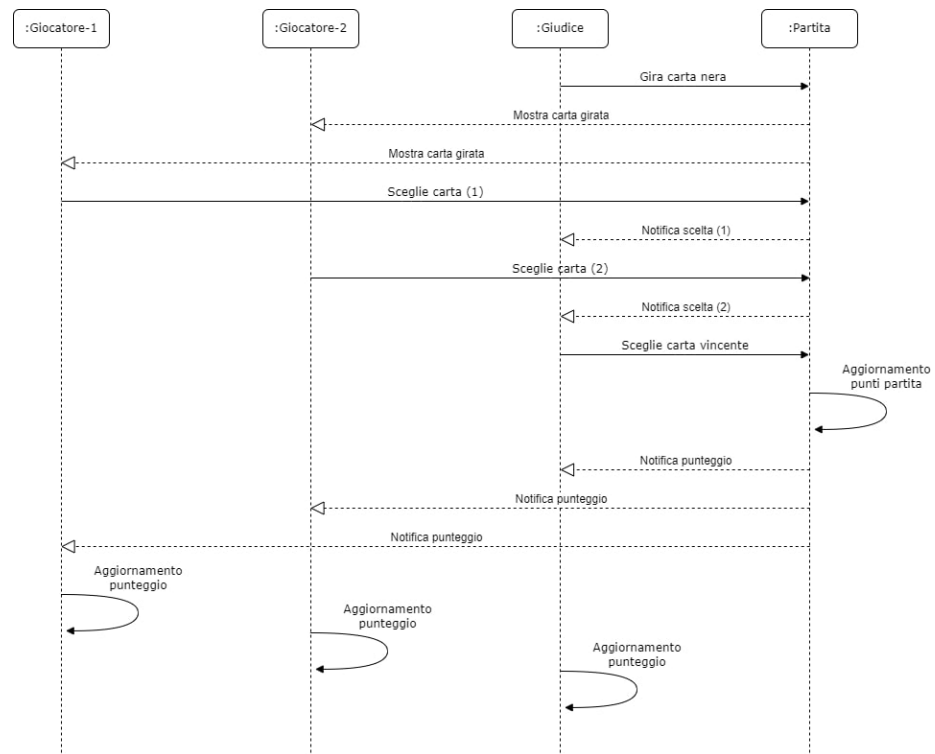
0.8.2 Partecipazione ad una partita pubblica

Nella sezione Partite Pubbliche del menù è possibile vedere quali partite sono attive in questo momento, essendo partite pubbliche vi si può accedere senza avere un codice. Se il numero degli utenti già presenti nella partita lo consente, quando un utente viene aggiunto alla partita tutti i partecipanti vengono aggiornati.



0.8.3 "Round" di una partita

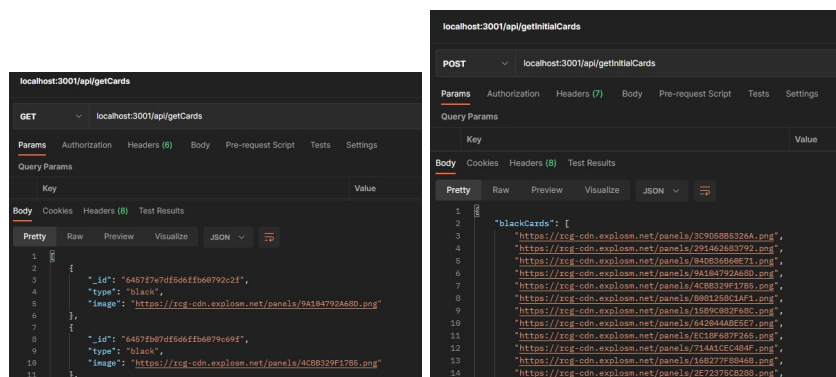
Quando comincia un round ogni giocatore deve avere 7 carte, a questo punto il giudice, che in questa mano non giocherà nessuna carta, gira la carta nera in modo che tutti i partecipanti possano vederla. I giocatori scelgono la carta da giocare e il giudice una volta ricevute tutte le carte sceglie qual è secondo lui la carta vincente attribuendo successivamente il punteggio a chi ha giocato quella carta.



0.9 Validazione

La validazione della funzionalità di questo progetto è stata ottenuta seguendo più metodologie e strumenti diversi.

Il testing delle rotte ho usato Postman, un'applicazione che permette di creare e inviare richieste HTTP a un server. Postman mi ha permesso di simulare le diverse rotte che il codice esporta, come `/api/checkAvailableRooms`, `/api/getCards`, `/api/getInitialCards`, `/api/getActiveRooms` e `/api/getPublicRooms`. Per ogni rotta, ho impostato il metodo HTTP appropriato (GET o POST), i parametri necessari (se presenti), e ho inviato la richiesta al server. Postman mi ha mostrato la risposta del server, che ho confrontato con quella attesa. In questo modo, ho potuto verificare se il codice funzionava correttamente e se gestiva gli errori in modo adeguato



Per testare le funzionalità dei miei componenti React, ho deciso di coinvolgere i componenti del focus group che avevo usato per ideare l'interfaccia della mia web app. Ho ritenuto che fosse più efficace usare 10 utenti reali che svolgono numerosi test reali e non simulati, piuttosto che usare strumenti automatizzati o simulazioni. Ho chiesto ai componenti del focus group di interagire con la mia web app, provando le diverse funzioni e valutando l'usabilità, l'estetica e la soddisfazione. Ho raccolto i loro feedback tramite questionari, interviste e osservazioni. In questo modo, ho potuto verificare se i miei componenti React funzionavano come previsto e se rispondevano alle esigenze e alle aspettative degli utenti.

Infine, per testare la parte di comunicazione tra client e server del mio software, ho usato il modulo `socket.io-client`, che mi ha permesso di creare un socket fittizio che si connette al server e invia e riceve eventi. Ho usato il framework di testing Mocha per scrivere i test, e le librerie Chai e Assert per verificare le asserzioni. Per ogni test, ho impostato il socket del client prima di eseguire il test, e l'ho disconnesso dopo averlo completato. Ho usato le funzioni `emit` e `on` del socket per inviare e ascoltare gli eventi, e ho controllato se la risposta del

server era quella attesa. In questo modo, ho potuto verificare se il server gestiva correttamente le richieste dei client e se inviava i dati corretti.

0.10 Deployment

Per provare l'applicazione è necessario seguire i seguenti passaggi:

1. **Clonazione delle directories:** Joking-Hazard_Frontend & Joking-Hazard_Backend
 2. **Frontend:** sarà necessario installare yarn attraverso il comando
- ```
1 yarn install
```
- e automaticamente tutte le librerie necessarie verranno scaricate.
3. **Funzionamento:** per prima cosa è necessario far partire il server semplicemente avviando il file server.js, il quale mostrerà in console un messaggio per avvisare della riuscita connessione al database.

```
Node API server started on port 3001
Connected to MongoDB...
```

Ora per avviare il frontend basta scrivere in console

```
1 npm start
```

oppure

```
1 yarn start
```

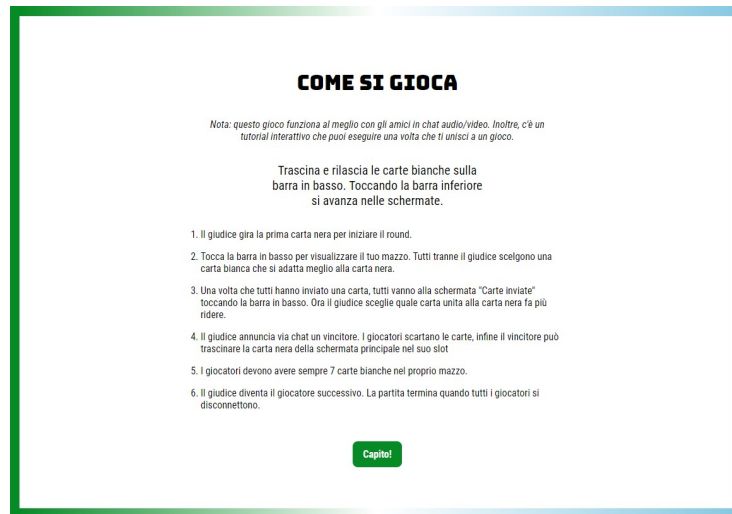
Ora è possibile collegarsi alla piattaforma tramite un browser con indirizzo **http://localhost:3000**, per vedere il funzionamento a più giocatori basta aprire un'altra pagina sempre su **http://localhost:3000**

## 0.11 Esempi d'uso



Una volta avviata un'istanza del client di gioco (con il server pronto in ascolto) comparirà la schermata del menù del gioco dal quale potranno essere effettuate 4 scelte:

- Leggere le istruzioni.
- Creare una partita.
- Vedere le partite pubbliche.
- Unirsi ad una partita privata tramite il codice della partita.



In questa pagina è possibile vedere quali sono le regole del gioco. durante la partita è presente un ulteriore tutorial per indicare passo per passo cosa fare.

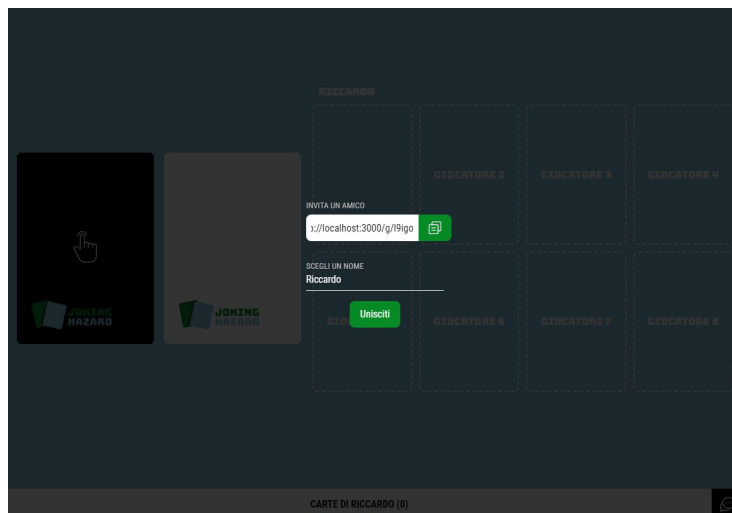
Nella pagina di creazione della partita si può scegliere se rendere la partita pubblica o privata



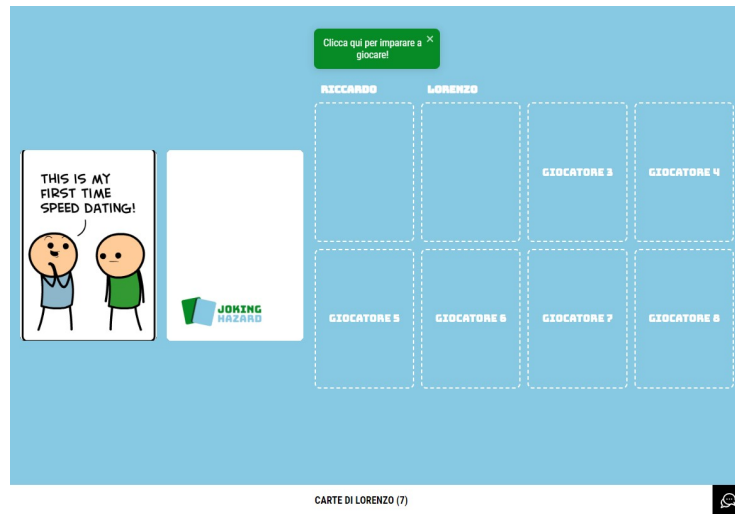


Nella pagina delle Partite Pubbliche si possono vedere quali sono le partite in corso, il loro codice e quante persone sono già presenti.

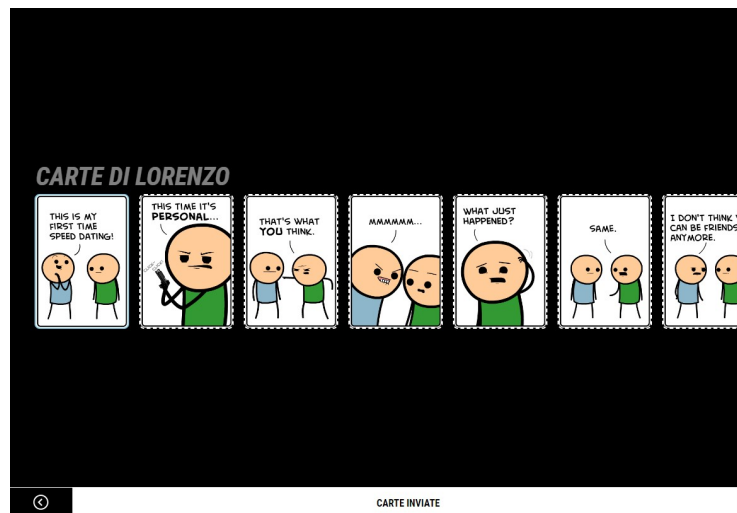
In qualunque modo si scelga di entrare in una partita è necessario inserire il proprio nickname.



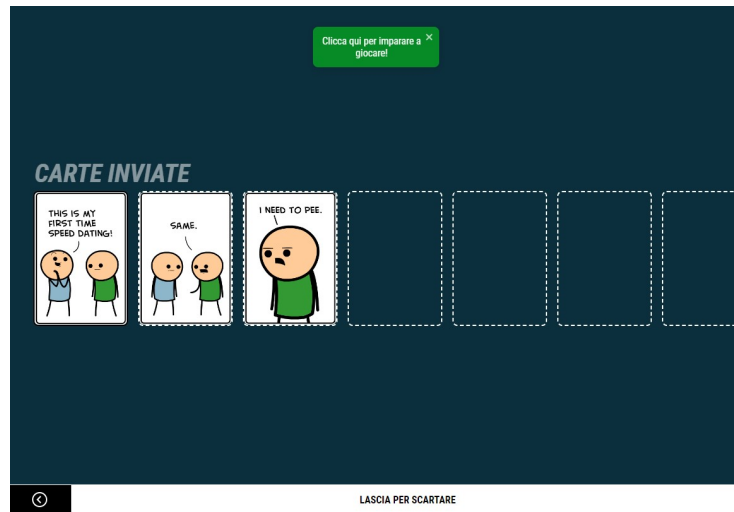
Il gioco si articola su tre schermate. La prima rappresenta il "tavolo" ovvero il luogo comune in cui è presentata la carta nera da completare.



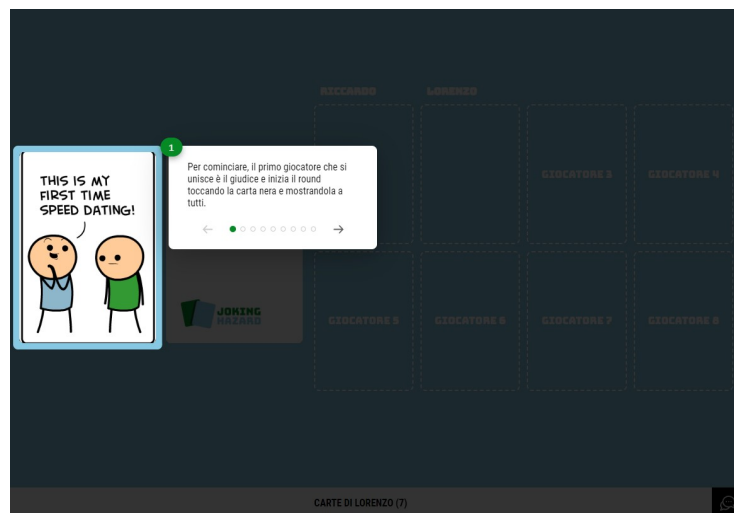
La seconda schermata rappresenta il mazzo di ogni giocatore.



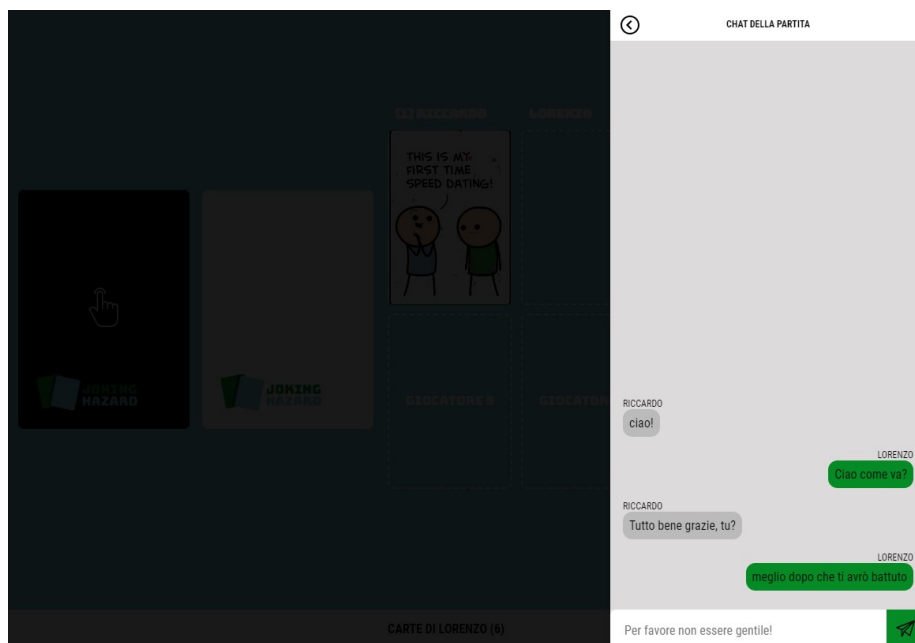
Infine l'ultima schermata è il punto in cui il giudice rivela le carte scelte dai giocatori e decreta il vincitore di questo "round".



Nel gioco è presente anche un tutorial che spiega passo per passo il funzionamento del gioco



E' presente anche una Chat che permette ai giocatori di comunicare in tempo reale.



## **0.12 Conclusioni**

### **0.12.1 Riflessioni finali**

In conclusione, il progetto del gioco di carte online ha dimostrato un notevole successo, grazie alla sua architettura ben progettata e all'uso efficace di tecnologie moderne. L'architettura adottata ha reso l'aggiunta di funzionalità aggiuntive un processo relativamente semplice, offrendo una grande flessibilità per futuri sviluppi.

### **0.12.2 Possibili miglioramenti**

Tuttavia, alcune funzionalità potenziali, come l'aggiunta di carte personalizzate da parte degli utenti, non sono state implementate. Questo è dovuto principalmente alla difficoltà nel reperire le immagini delle carte del gioco reale su internet. Nonostante questa limitazione, il progetto ha dimostrato di essere robusto e versatile, offrendo un'esperienza di gioco coinvolgente e interattiva. Guardando al futuro, con ulteriori sviluppi e miglioramenti, il gioco di carte online ha il potenziale per diventare ancora più coinvolgente e personalizzabile, offrendo agli utenti un'esperienza di gioco ancora più ricca e gratificante.

### **0.12.3 Che cosa ho imparato**

Questo progetto mi ha permesso di approfondire la conoscenza e l'uso delle socket, nonché dei rapporti tra client e server nella programmazione distribuita.