Federico Brandini (mat. 317271)
Federico Putamorsi (mat. 320041)

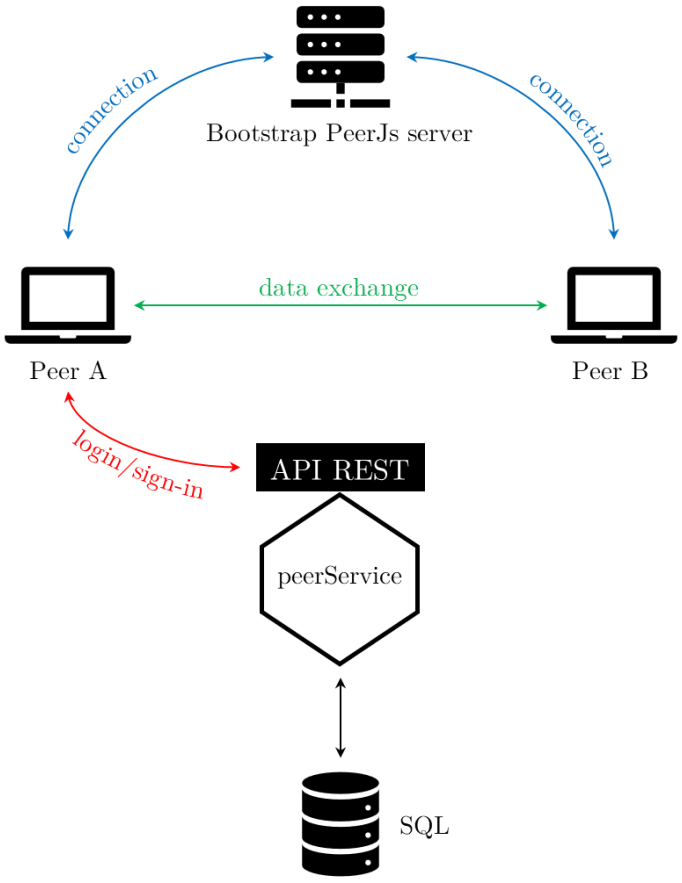
Forza4 P2P

1. Introduzione

Si desidera realizzare un'applicazione web peer-to-peer a singola pagina (sfruttando il framework React) che permetta ad un utente di sfidarne un altro a "Forza4". Il sistema deve inoltre consentire agli utenti in competizione di comunicare attraverso una semplice chat. Per poter giocare, in primo luogo, gli utenti devono essere registrati e aver eseguito l'accesso con una credenziale (nome utente e password). L'utente che ha eseguito l'accesso potrà inviare inviti di sfida ad altri giocatori registrati nel sistema (indicandone il nome utente) e attendere che tali inviti vengano accettati o rifiutati. Una volta accettato l'invito, i due utenti potranno giocare liberamente e interrompere la partita in qualsiasi momento (l'interruzione è obbligatoria al termine della partita).

2. Progettazione generale

La struttura generale del sistema può essere schematizzata come segue.



Per la gestione della comunicazione P2P è stata impiegata la libreria PeerJs di JavaScript, che semplifica lo sviluppo WebRTC, fornendo API semplici e complete. Il Bootstrap PeerJs server serve esclusivamente per aiutare i due peer a stabilire la connessione fra essi (come un server di inizializzazione o di bootstrap, appunto); infatti, i dati vengono poi scambiati direttamente fra i due peer, senza passare dal server. Un peer può indipendentemente invocare il microservizio di gestione del database delle credenziali per eseguire il login, oppure per registrarsi. Il codice identificativo di

ogni peer è infatti la chiave primaria della sola tabella contenuta nel database SQL: il nome utente del giocatore.

3. Protocollo overlay

Una volta stabilita la connessione, i due peer utilizzano un protocollo overlay per lo scambio dei messaggi. Per riconoscere le azioni da svolgere e mantenere coerente lo stato della partita e della chat, predisponiamo i seguenti tipi di messaggi:

- **chat**, per indicare che il contenuto del messaggio appartiene alla chat;
- **confirm**, per indicare se il peer remoto ha accettato o meno l'invito al gioco;
- **move**, che incapsula la mossa eseguita nel gioco;
- **end**, per indicare la fine della partita con il relativo esito.

4. Database

Il database SQL del sistema comprende un'unica tabella in cui vengono memorizzate le credenziali degli utenti registrati; risulta quindi inutile riportare lo schema concettuale (E/R) e quello logico (relazionale), in quanto comprenderebbero rispettivamente un'entità e una relazione. Per una gestione sicura delle password, all'interno della base di dati ne viene memorizzato l'hash con la funzione crittografica SHA3-512. Quando un utente si registra o esegue il login (che ha la sola funzione di autenticazione), viene immediatamente calcolato l'hash della password inserita (sul client); in questo modo la password in chiaro non entrerà mai in rete, ma solo la sua sintesi. Il sistema rimane comunque vulnerabile ad un attacco replay. Per la connessione tra il microservizio e la base di dati è stato impiegato il modulo *mysql*.

5. Microservizio peerService

Il microservizio peerService è stato implementato nel file *peerService.js* con il framework Express. Il suo scopo riguarda la gestione del database appena descritto. Le API RESTful messe a disposizione hanno le seguenti routes:

- / (richiesta GET), endpoint di benvenuto;
- */peer/username* (richiesta GET), per ottenere la rappresentazione JSON della credenziale dell'utente specificato;
- */newPeer* (richiesta POST), per la creazione di una nuova credenziale spedita nel body in formato JSON;
- */peers* (richiesta GET), per ottenere dieci username di giocatori dal database.

L'ultima route non viene mai richiamata all'interno dell'applicazione (client), ma potrebbe essere utile per una nuova versione futura della medesima, in cui è possibile scegliere lo sfidante fra un elenco iniziale di peer disponibili.