One vs All

Simone Venturi
simone.venturi5@studio.unibo.it

December 2022

One vs All è un gioco multiplayer online nel quale un giocatore si ritrova ad affrontare altri concorrenti in una serie di domande a scelta multipla. I partecipanti accedono al turno successivo rispondendo correttamente alla domanda del turno corrente. Ogni turno ha la durata di 30 secondi. Una volta terminati, il turno è considerato concluso e sarà elaborato il turno successivo. Il gioco termina quando il giocatore, rispondendo correttamente a tutte le domande, vince ed elimina tutti gli avversari oppure sbaglia una risposta venendo così sconfitto. Il sistema che offre il servizio One vs All ha caratteristiche di fault tolerance, scalabilità e replicabilità.

1 Obiettivo

Un utente che utilizza la piattaforma One vs All deve essere in grado di giocare ad un quiz game o in qualità di giocatore oppure in qualità di concorrente.

1.1 Domande per raffinare i requisiti

- Può essere possibile partecipare a più partite contemporaneamente?
  - Sì: il server deve essere in grado di gestire flussi di gioco di un client per diverse partite.
- C'è un numero minimo di partecipanti per iniziare il gioco?
  - Sì: oltre al giocatore deve essere presente almeno un concorrente, in questo caso la partita è un 1 vs 1
- Può essere possibile partecipare ad una partita già iniziata?
  - No, non è possibile partecipare ad una partita già iniziata. Non è prevista una modalità spettatore.
- Un utente come viene identificato?
  - L'utente è identificato tramite indirizzo email.
- Gli utenti all'interno di una partita hanno gli stessi diritti?
  - giocatore: può iniziare la partita e rispondere alle domande
  - concorrente: può rispondere alle domande

1.2 Scenari

Un utente che utilizza la piattaforma vuole giocare ad un quiz game contro altri giocatori. Perciò, in un primo momento, gli sarà richiesto di autenticarsi ed eventualmente registrarsi. In secondo luogo, potrà partecipare o creare una lobby di gioco con pochi click. Infine, giocherà al quiz game fino al termine della partita. Una volta visto l'esito, potrà tornare nella pagina iniziale e ripetere l'iter per partecipare o creare una nuova partita.

1.3 Politiche di autovalidazione

Il collaudo della piattaforma viene effettuato per mezzo di test automatici, sia unitari che di integrazione. I test comprendono lo scenario classico di utilizzo e la verifica delle principali operazioni di base che la piattaforma offre. Inoltre, il piano di sviluppo è gestito nel seguente modo:

- inizializzazione dell'architettura generale del sistema
- sviluppo delle funzionalità di autenticazione ed autorizzazione

- sviluppo gestione di lobby di gioco
- sviluppo flusso di gioco

Ad ogni punto raggiunto verrà rilasciata una versione del sistema sviluppato.

2 Analisi dei requisiti

Elaborando questi requisiti si ottengo queste funzionalità, come mostrato anche nei casi d'uso (figura 2):

- registrarsi
- effettuare il login
- gestione delle eventuali autorizzazioni
- creare una partita
- unirsi ad una partita come concorrente
- rispondere ad una domanda

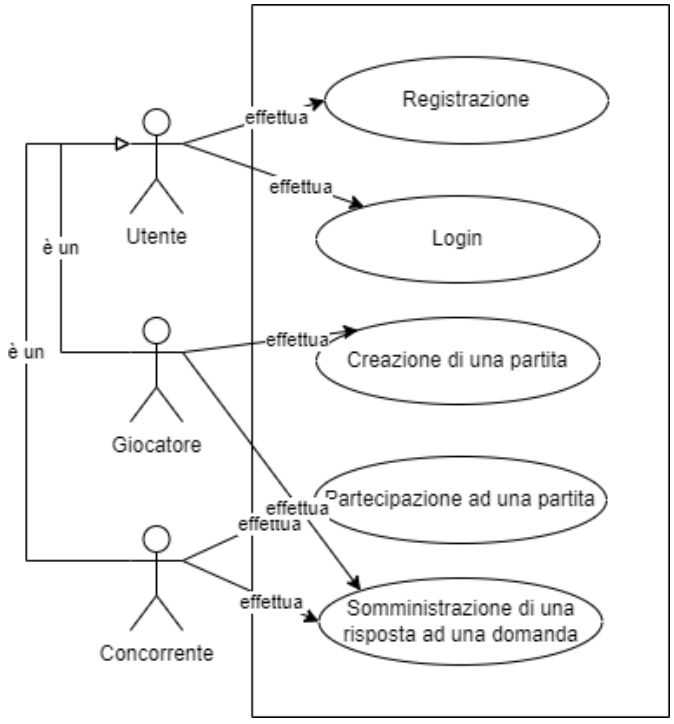

Figure 1: Casi d'uso di One vs All

Al fine di soddisfare i requisiti evidenziati, si è optato per la scelta dello stile architetturale ReST e l'utilizzo del pattern client-server.

3 Design

Il sistema viene disegnato come una piattaforma Web che utilizza lo stile architetturale ReST. Questo, inoltre, viene pensato a microservizi e distribuito, così da poter scalare, se opportunamente configurato, all'aumentare del traffico.

3.1 Design architetturale

Raffinando i requisiti emersi, sono state identificate tre componenti del sistema, come mostrato anche nel diagramma delle componenti (figura 3.1):

- frontend: un servizio che contiene un'applicazione che consente ad un utente di giocare a One vs All tramite un'interfaccia grafica.
- backend: un servizio che contiene un'applicazione con la quale i frontend comunicano e che gestisce la gestione dei dati ed il flusso del gioco.
- persistenza: un servizio che gestisce la persistenza dei dati.

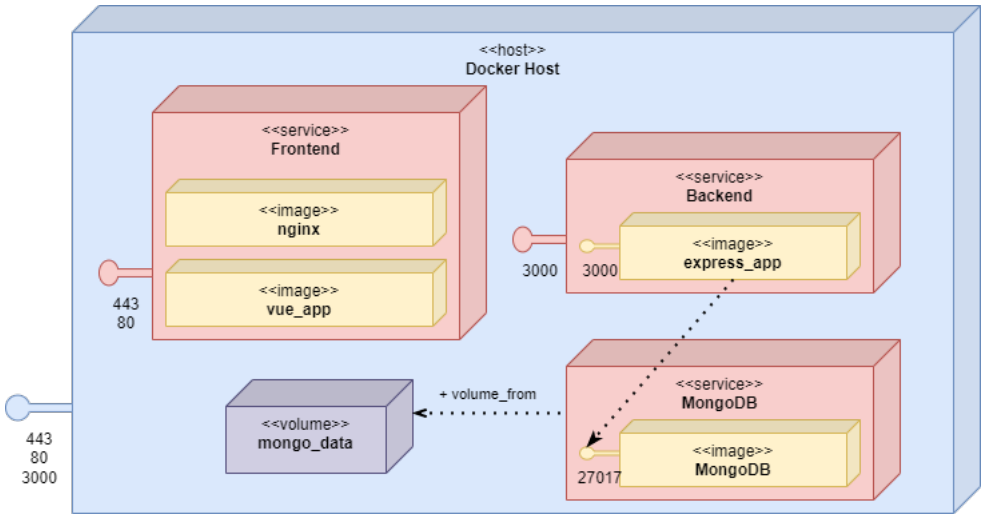

Figure 2: Diagramma dei componenti di One vs All

3.2 Struttura

Le entità del sistema sono rappresentate con il diagramma delle classi, mostrato in figura 3.2.

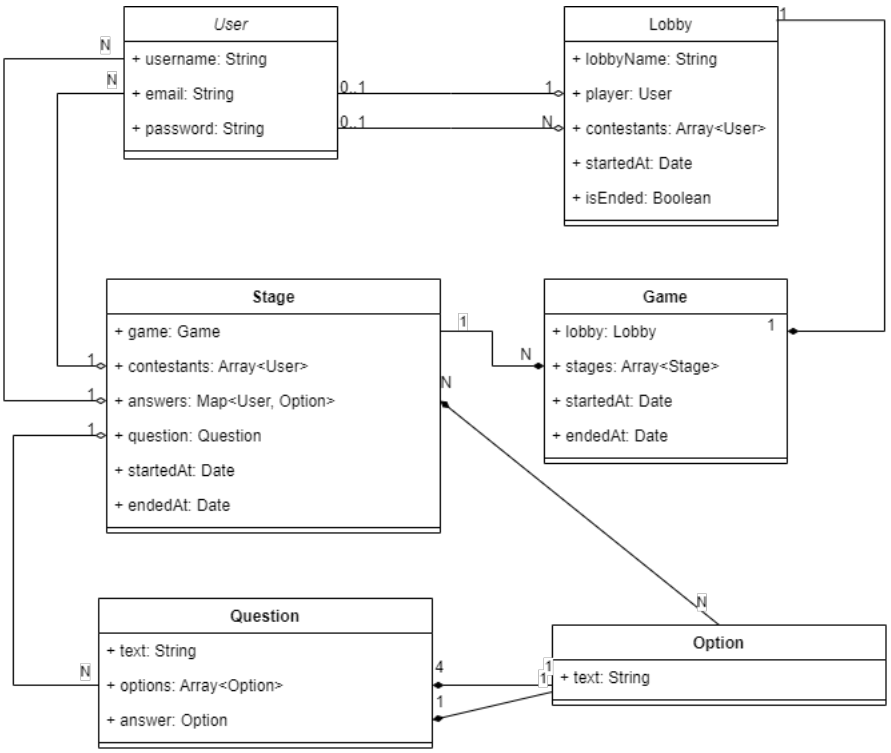

Figure 3: Diagramma delle classi del model di One vs All

L'entità **User** è la rappresentazione di un utente. Questo è caratterizzato da:

- username: il nickname dell'utente.
- email: l'indirizzo email dell'utente.
- password: la password dell'utente.

L'entità **Lobby** è la rappresentazione di una lobby di gioco. Questa è caratterizzata da:

- lobbyName: il nome della lobby.
- player: un riferimento a User. Questo campo rappresenta l'utente che partecipa alla lobby in qualità di giocatore.
- contestants: una lista di riferimenti a User. Questo campo rappresenta gli utenti che partecipano alla lobby in qualità di partecipanti.

- startedAt: la data di inizio della partita.
- isEnded: un valore booleano che contiene l'informazione se la partita è terminata o meno.

L'entità **Game** è la rappresentazione di una partita. Questa è caratterizzata da:

- lobby: il riferimento alla lobby che ha generato questa partita.
- stages: una lista di riferimenti a Stage. Questo campo rappresenta gli stage che formano la partita.
- startedAt: la data di inizio della partita.
- endedAt: la data di fine della partita.

L'entità **Stage** è la rappresentazione di una fase della partita. Questa è caratterizzata da:

- game: il riferimento al game al quale appartiene lo stage.
- contestants: una lista di riferimenti a User. Questo campo rappresenta gli utenti che sono abilitati a partecipare allo stage.
- answers: una mappa che associa un riferimento Utente ad un riferimento Option, ovvero la risposta che un utente ha somministrato per lo stage.
- question: il riferimento alla domanda estratta per lo stage.
- startedAt: la data di inizio dello stage.
- endedAt: la data di fine dello stage.

L'entità **Question** è la rappresentazione di una domanda. Questa è caratterizzata da:

- text: il testo della domanda.
- options: una lista di riferimenti a Option. Questo campo rappresenta le opzioni disponibili per la domanda.
- answer: il riferimento alla Option che rappresenta la risposta corretta.

L'entità **Option** è la rappresentazione di una opzione. Questa è caratterizzata da:

- text: testo dell'opzione.

3.3 Comportamento

Le entità che hanno un comportamento dinamico sono **Lobby**, **Game** e **Stage**.

In figura 3.3, viene mostrato il comportamento dinamico dello stato di una lobby dalla sua creazione fino al suo compimento, ovvero l'inizio del gioco.

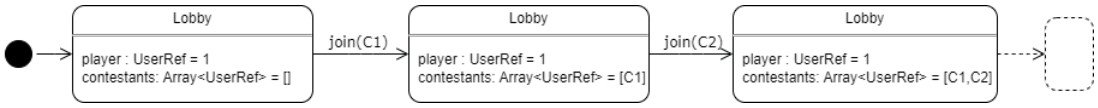

Figure 4: Diagramma di stato dell'entità Lobby

In figura 3.3, viene mostrato il comportamento dinamico dello stato durante il flusso di una partita.

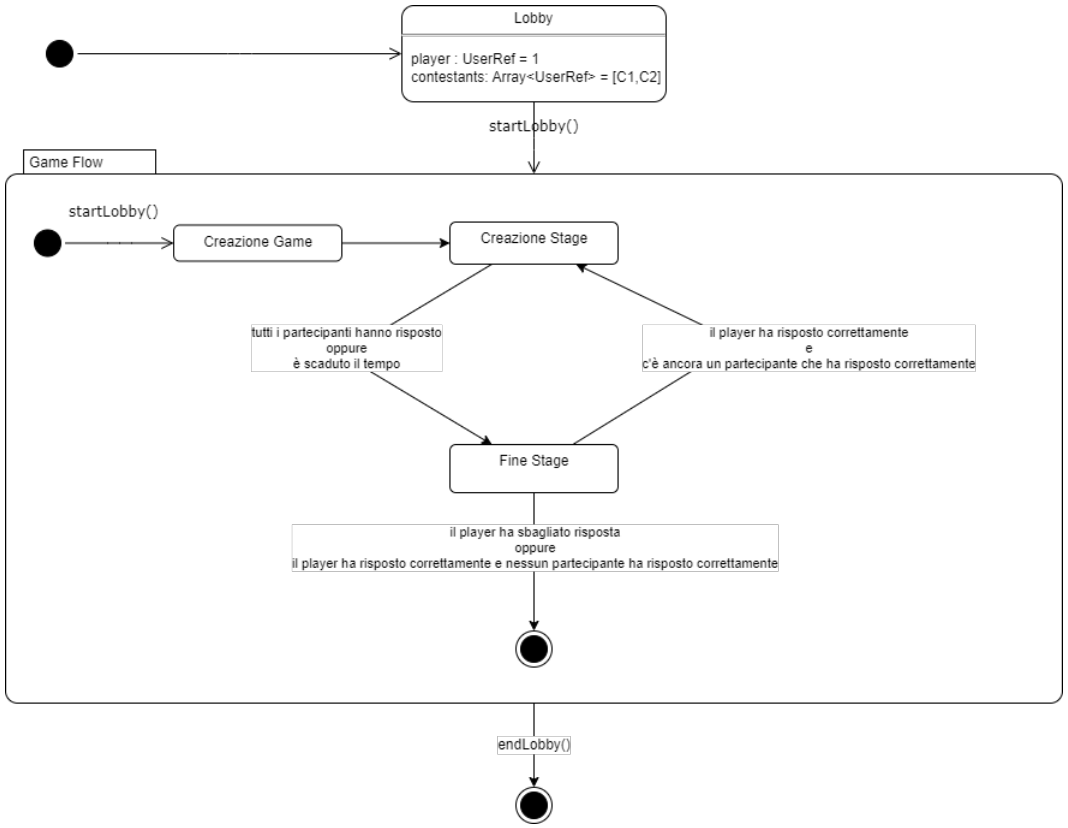

Figure 5: Diagramma di stato del flusso di gioco

3.4 Interazioni

Di seguito saranno illustrate le principali interazioni del sistema.

In figura 3.4, viene mostrato l'interazione delle componenti durante lo scenario di autenticazione.

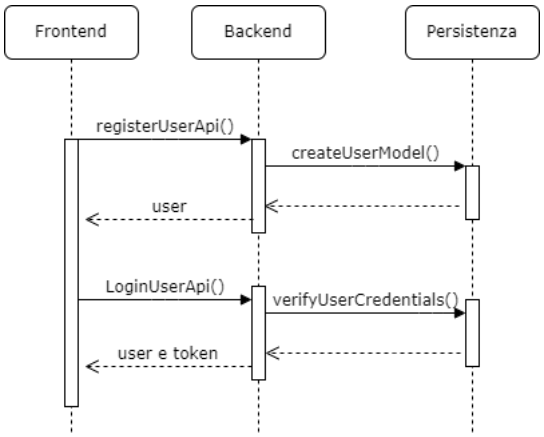

Figure 6: Diagramma di sequenza dell'autenticazione

Una volta autenticati, il frontend avrà un token con il quale è possibile accedere ad altre funzionalità del backend, senza il quale nessuna altra operazione è resa disponibile dal servizio di backend. In figura 3.4, viene mostrato l'interazione delle componenti durante lo scenario di creazione lobby e join di una lobby.

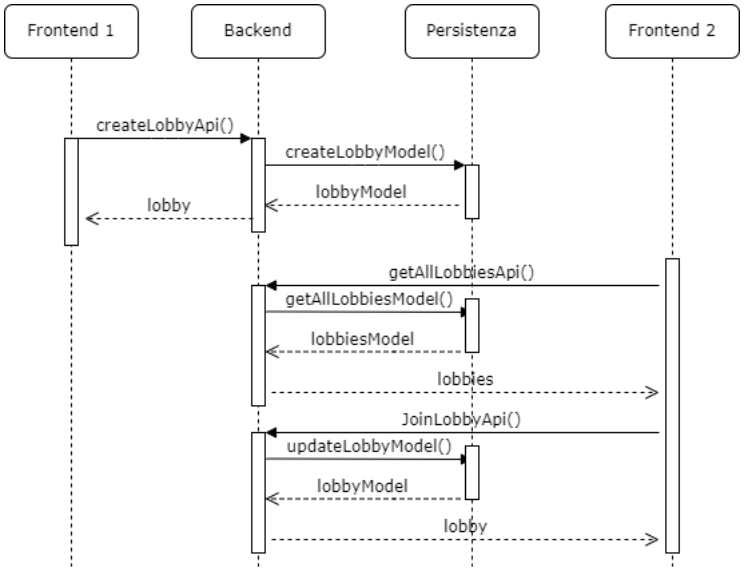

Figure 7: Diagramma di sequenza della gestione di una lobby

Una volta creata una lobby è possibile iniziare una partita. Il flusso della gestione di un turno e la somministrazione di una giocata è mostrato in figura 3.4. In quest'ultima

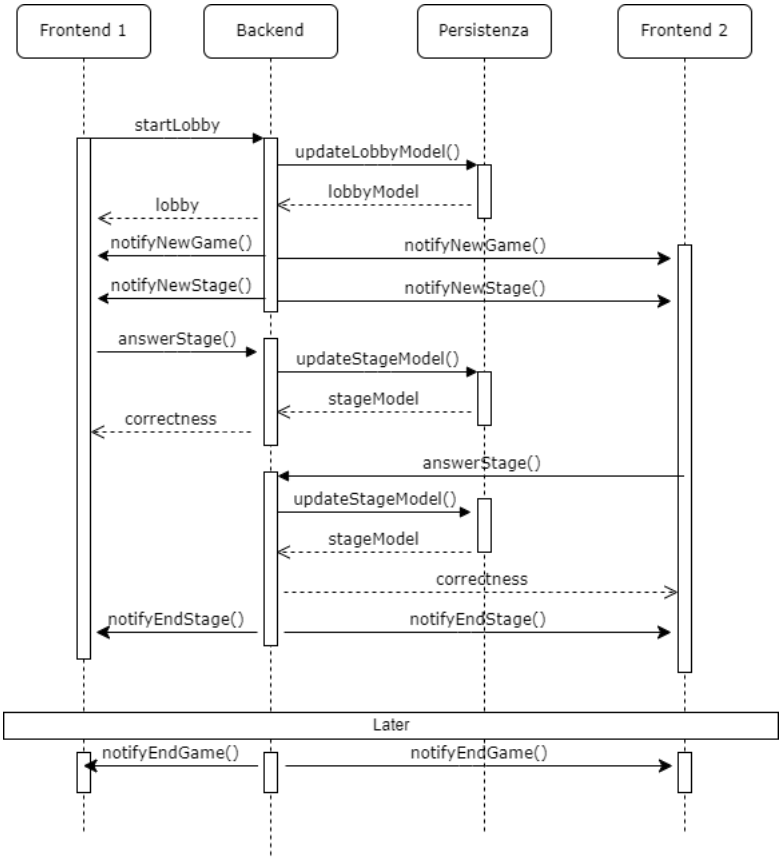

Figure 8: Diagramma di sequenza del flusso di una partita

interazione emerge la necessità di mantenere allineati più frontend che partecipano alla stessa partita.

4 Implementazione

In questa sezione saranno riassunti i dettagli implementativi adottati al fine di realizzare il sistema e finalizzare gli obiettivi prefissati, suddivisi nei tre servizi delineati durante la fase di design.

4.1 Persistenza

Al fine di garantire la persistenza ai dati generati e consentire una corretta esecuzione delle partite, è risultato necessario creare una base di dati nella quale conservare le informazioni. A questo fine si è scelto di adottare una DBMS non relazionale, MongoDB mon [a]. Questa scelta permette al modello delineato in fase design (figura 3.2) di essere conservato tale e quale. Inoltre, la componente di persistenza, come mostrato in figura 3.1 utilizza un volume di appoggio nel quale sono conservati tutti i dati per il ripristino dello stato della base di dati in caso di errore del database e/o della piattaforma, aumentando la resilienza del sistema. In ultima analisi, il servizio di persistenza non espone nessuna porta all'esterno del sistema ed è consultabile solamente dal servizio di backend.

4.2 Backend

Il servizio di backend è il principale responsabile della gestione delle funzionalità delineate in fase di analisi dei requisiti e rappresenta il motore del sistema. Lo sviluppo del backend è stato realizzato utilizzando Javascript come linguaggio di programmazione eseguito su Node.js. Il backend fa uso del framework Express.js exp, uno standard de facto per lo sviluppo di ReST API in Javascript. Al fine di rendere più fluido e veloce lo sviluppo della parte di persistenza e di modellazione delle classi, si è deciso di optare per l'utilizzo di un ODM, in questo caso Mongoose mon [b]. Questa scelta ha come effetto collaterale quello di legarsi alla scelta tecnologica del database; non risulta il migliore scenario qualora, in un futuro, si volesse cambiare il servizio di persistenza. Tutte le rotte per l'autenticazione e la gestione delle lobby sono state documentate con l'ausilio di Swagger e sono consultabili raggiungendo il percorso /docs di questo servizio come visibile in figura 4.2. Come descritto nei requisiti funzionali, il sistema deve prevedere un layer di autorizzazione. A tal fine, si è scelto di utilizzare JSON Web Token jwt come tecnologia per conservare le informazioni necessarie al riconoscimento dell'utente una volta autenticato all'interno della piattaforma. Questo token deve essere fornito in tutte le rotte descritte all'interno dell'header in un campo denominato x-ovsa-access-token.
Alcune rotte fanno uso di alcuni middleware sviluppati al fine di validare le richieste ed eventualmente intercettare quelle erronee restituendo i messaggi di errore prestabiliti. Ogni messaggio in uscita del backend è creato da un modulo sviluppato appositamente al fine di conservare la formattazione dei messaggi separata dal livello del modello sviluppato. Per testare il sistema, durante lo sviluppo sono stati progressivamente sviluppati sia dei test di unità che di integrazione tramite l'ausilio del framework Jest jes.

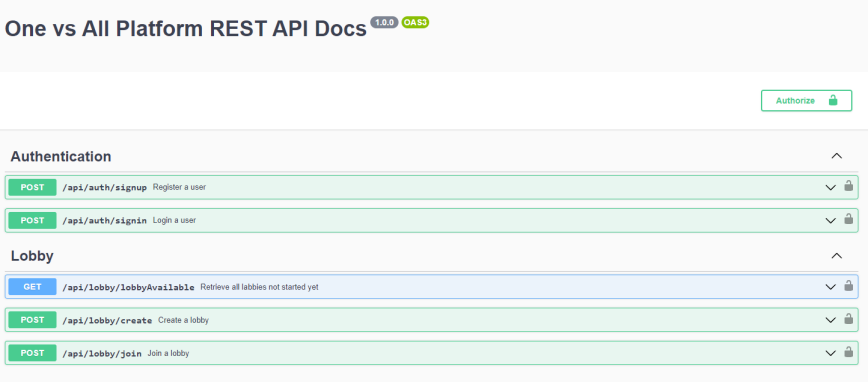

Figure 9: Swagger API docs

Come è emerso durante la modellazione dell'interazione del flusso di gioco di una partita, in alcuni momenti è necessario che il server riesca a comunicare ad ogni client la modifica dello stato della partita perché un altro utente ha avuto un'interazione. Per risolvere questa esigenza si è deciso di utilizzare le WebSocket per instaurare un canale full duplex tra client e server. La server socket sviluppata gestisce tutte le interazioni necessarie per il flusso corretto di una partita: dal momento dell'accesso dell'utente all'interno di una lobby sino alla fine del gioco, come mostrato nella figura 3.4. Inoltre, la socket in ascolto utilizza lo stesso meccanismo di autorizzazione delle API. Il servizio di backend espone di default la porta 3000.

4.3 Frontend

Il servizio di frontend ha il principale compito di accompagnare un utente durante la sua esperienza all'interno del quiz game. Questo servizio è sviluppato tramite una Single Page Application con l'utilizzo del framework Vue.js vue [a]. I percorsi sviluppati sono:

- "/register" : path nel quale è possibile registrarsi
- "/login" : path nel quale è possibile effettuare il login
- "/" : path dove sono raggruppate tutte le funzionalità per creare una lobby, aggiungersi ed entrarci.
- "/lobby/:id" : path dentro la lobby di gioco nella quale si svolge tutto il quiz game.

Il servizio fa uso di uno store (tramite il pacchetto vuex vue [b]) che centralizza tutte le informazioni del client in un unico punto ed espone le informazioni tramite dei getters. Lo store inoltre fa uso dei servizi interni sviluppati per comunicare in maniera conforme con le specifiche definite nella documentazione Swagger 4.2. Il frontend implementa anche la socket client necessaria per comunicare con il backend durante la partita. Il servizio di frontend è costruito con l'utilizzo di Nginx ed espone di default la porta 80.

5 Auto valutazione

Il sistema è provvisto di test di unità e di integrazione sviluppati parallelamente allo sviluppo del sistema. Questi test verificato nella completezza il modello delineato mentre per quanto riguarda l'integrazione sono coperti tutti gli endpoints dell'API ma non la parte di flusso di gioco gestito dalle WebSocket. Questi sono verificabili seguendo le istruzioni presenti nel README.md del repository di progetto alla voce *Esecuzione del sistema per testing* e successivamente la voce *Esecuzione dei test* il cui output è mostrato in figura 5

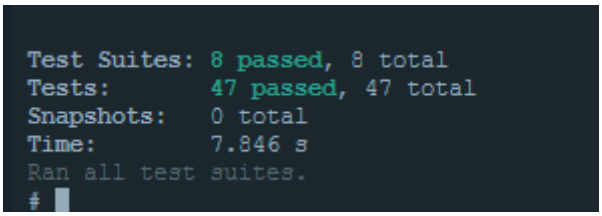

Figure 10: Esito esecuzione test

Inoltre sempre tramite lo strumento di versioning è possibile consultare l'evoluzione del progetto ed i derivables che sono stati prodotti durante il piano di sviluppo. In particolare, verificando i tag prodotti:

- v0.1: inizializzazione dell'architettura generale del sistema
- v0.2: sviluppo delle funzionalità di autenticazione ed autorizzazione
- v0.3: sviluppo gestione di lobby di gioco
- v0.4: sviluppo flusso di gioco
- v1.0: consegna One vs All

6 Istruzioni per il deploy

Il sistema è provvisto di tutto l'occorrente per la messa in produzioni. Tramite l'utilizzo della containerizzazione è possibile con pochi comandi scaricare i sorgenti e lanciarli in un ambiente vergine seguendo le istruzioni presenti nel README.md del repository di progetto alla voce *Esecuzione del sistema.*
Infine, per rendere possibile l'esecuzione del sistema, è stato creato un comando npm tramite il quale verranno generati i record di domande e relative opzioni nella base di dati. Per questa funzionalità occorre seguire le istruzioni presenti nel README.md del repository di progetto alla voce *Esecuzione dei seeder per la popolazione iniziale del database.*

7 Esempi d'uso

Ogni scenario proposto è testabile tramite l'uso del servizio di frontend. In alternativa, si può testare la soluzione per mezzo del front-end di documentazione Swagger all'indirizzo *http://localhost:3000/docs/*. Cliccando su un'operazione è possibile compilare il modello della richiesta oppure utilizzare l'esempio proposto. Seguendo gli scenari proposti:

- registrarsi: POST /api/auth/signup
- effettuare il login: POST /api/auth/signin
- gestione delle eventuali autorizzazioni: uso dell'autorizzazione di Swagger
- creare una partita: POST /api/lobby/create
- unirsi ad una partita come concorrente: POST /api/lobby/join

L'unico scenario non testabile direttamente dalla documentazione di Swagger è la somministrazione di una risposta ad una domanda poiché fa uso delle WebSocket. Nella figura 7 si può osservare la possibilità da parte di un utente di inviare al server la propria risposta durante uno stage.

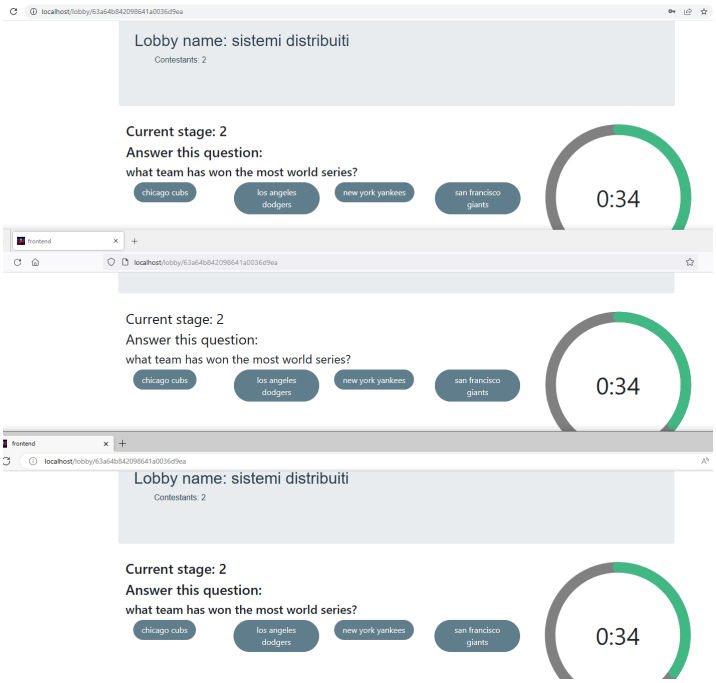

Figure 11: GUI di una partita

8 Conclusioni

Seguendo queste scelte di design e di implementazione è stato prodotto un sistema per la gestione di un quiz game online multiplayer. La scelta architetturale di disegnare il sistema come una composizione di microservizi permette a questa di essere più facilmente scalabile, replicabile su ogni ambiente con sufficienti risorse hardware e resiliente ad errori poiché ogni container ha una sua strategia di recovery.

8.1 Sviluppi futuri

Il sistema nella sua interezza è migliorabile. A livello architetturale si potrebbero inserire delle repliche per ogni servizio in modo tale da poter gestire meglio il carico qualora ci fossero un numero alto di richieste e partite. A livello implementativo, il frontend non ha ricevuto il sufficiente focus per avere una User Experience degna di nota. Inoltre, si potrebbero inserire dei test di integrazione che riguardino la logica di gioco che usufruisce delle WebSocket.

8.2 Concetti appresi

Questo progetto mi ha permesso di mettere in pratica aspetti e tecnologie viste durante il corso di Sistemi Distribuiti quali il design di un sistema distribuito, l'implementazione di una soluzione replicabile tramite strumenti di containerizzazione come Docker, la scrittura di interfacce per la rappresentazione delle risorse e i concetti di resilienza e tolleranza ai guasti.
Inoltre, ho potuto approfondire lo sviluppo di software tramite lo stack MEVN e utilizzare la tecnologia delle WebSocket.

References

Mongodb, a. URL https://www.mongodb.com/.

Express.js. URL https://expressjs.com/.

Mongoose, b. URL https://mongoosejs.com/.

Json web token. URL https://www.npmjs.com/package/jsonwebtoken.

Jest.js. URL https://jestjs.io/.

Vue.js, a. URL https://vuejs.org/.

Vuex, b. URL https://vuex.vuejs.org/.