

GuessR

Applicazioni e Servizi Web

Filasetta Angelo - 000725342 {angelo.filasetta@studio.unibo.it}

Sanchi Piero - 0001001367 {piero.sanchi@studio.unibo.it}

Talmi Alessandro - 0000979617 {alessandro.talmi@studio.unibo.it}

23 giugno 2021

Indice

1	Introduzione	4
1.1	Descrizione del sistema	4
2	Requisiti	5
2.1	Obiettivi	5
2.2	Personas	6
2.2.1	Jonathan	6
2.2.2	Jolyne	6
2.2.3	Joseph	7
3	Design	8
3.1	Struttura del Server	8
3.1.1	Gestione delle Route	8
3.1.2	Gestione delle Web Socket	8
3.1.3	Stato del Server	9
3.1.4	Gestione di una partita	10
3.1.4.1	Possibile implementazione alternativa semplificata e vantaggi derivanti da quella adottata	12
3.2	Struttura del Client	12
3.2.1	Componenti	13
3.2.2	Stato del Client	13
3.2.3	Gestione delle socket	14
3.3	Struttura del Database	14
3.4	Interfaccia Utente	15
3.4.1	Wireframe	15
3.4.2	Accessibilità	20
3.4.3	Schermate finali	21
3.4.4	Scelte estetiche e UI	26
4	Tecnologie	27
4.1	Tecnologie condivise tra client e server	27
4.1.1	Node Package Manager	27
4.1.2	Socket.io	28
4.2	Server-side	28
4.2.1	Node.js	28
4.2.2	Mongoose	29
4.3	Client-side	30
4.3.1	React.js	30
4.3.2	Redux	31
4.3.3	Bootstrap	32
4.4	Database-side	33
4.4.1	mongoDB	33

5 Codice	35
5.1 Server	35
5.1.1 Routes	35
5.1.2 Model su Mongoose	35
5.1.3 Socket Handlers	36
5.2 Client	36
5.2.1 Requests	36
5.2.2 Socket Handlers	37
6 Test	38
6.1 Redux DevTools	38
6.2 Axe DevTools	39
6.3 Postman	39
7 Deployment	41
7.1 Docker-free deployment	41
7.1.1 Prerequisiti, installazione di Software	41
7.1.2 Deployment Server	41
7.1.3 Deployment Client	42
7.2 Dockerized deployment	42
7.2.1 Docker	42
7.2.2 Docker compose	42
7.2.3 Deployment	43
8 Conclusioni	44
8.1 Note finali	44
8.1.1 Sviluppi futuri	44

Elenco delle figure

3.1	Diagramma delle classi del model del Server (in gioco)	10
3.2	Diagramma di sequenza per lo svolgimento di una partita	11
3.3	Diagramma delle classi della struttura del Database	14
3.4	Wireframe menu principale	16
3.5	Wireframe creazione lobby	17
3.6	Wireframe lobby	18
3.7	Wireframe gioco	19
3.8	Wireframe gioco mobile	20
3.9	Schermate di Login (Desktop e Mobile)	21
3.10	Schermata per la creazione di una nuova lobby	21
3.11	Schermata della Lobby (Desktop e Mobile)	22
3.12	Schermate Mobile di Chat e Giocatori	22
3.13	Schermata di sentence	23
3.14	Schermate di Disegno (Desktop e Mobile)	23
3.15	Schermata dei Report finale	24
3.16	Schermata dei report precedenti (Desktop e Mobile)	24
3.17	Schermate delle notifiche (Desktop e Mobile)	25
3.18	Schermata mobile contenente la navbar ed il menù a dropdown	25
3.19	Palette di colori utilizzata per i componenti del sito, il logo e lo sfondo	26
3.20	Logo di GuessR	26
4.1	Logo di npm [1]	27
4.2	Logo di Socket.io	28
4.3	Logo di Node.js [2]	29
4.4	Logo di Mongoose [3]	30
4.5	Logo di React.js [4]	31
4.6	Logo di Redux [5]	32
4.7	Logo di Bootstrap [6].	33
4.8	Logo di mongoDB [7].	34
6.1	Finestra di Redux Devtool	38
6.2	Logo di Redux DevTools	38
6.3	Logo di Axe DevTools	39
6.4	Schermata della Workspace Postman di GuessR	40
6.5	Logo di Postman	40
7.1	Logo di Docker	42

Capitolo 1

Introduzione

1.1 Descrizione del sistema

Il progetto nasce dall'idea di digitalizzare un gioco, una versione modificata di "sigaretta" [8] nato per caso dal gruppo di amici di uno dei nostri componenti, e renderlo digitale al fine di accorciare le distanze in questo periodo di forti limitazioni alla mobilità.

Il gioco, nella sua versione fisica, si svolge nel seguente modo: partecipano n giocatori (con n strettamente maggiore di 2, senza un limite superiore) ognuno dotato di un foglio di carta. Inizialmente ogni partecipante scrive una frase a piacimento su tale foglio e, senza farsi vedere dagli altri partecipanti, piega il foglio di modo che la frase non sia visibile.

Una volta che ogni giocatore ha scritto la frase, passa il foglio al giocatore alla sua sinistra che, letta la frase, provvederà a fare un disegno che la rappresenti. Analogamente a prima poi i giocatori piegheranno il foglio di modo da nascondere il disegno e lo passeranno nuovamente al giocatore alla loro sinistra che invece stavolta (potendo vedere solo il disegno) scriverà una frase che lo descriva.

Viene seguito questo pattern finché, dipendentemente dalla scelta dei giocatori, si è raggiunto un numero di "passaggi" o il giro è terminato. Alla fine del gioco vengono mostrati a tutti i giocatori i fogli nella loro interezza che, necessariamente, avranno preso una deriva strana rispetto alla frase scritta inizialmente. Non vi è dunque un vincitore, il gioco infatti punta solo ad analizzare i risultati finali.

Capitolo 2

Requisiti

L'idea di progetto consiste nel creare un'architettura Client-Server che permetta lo svolgimento del gioco sopra descritto tramite un client web.

Il server coordina i giocatori, orchestrando gli stati, con l'idea di rendere il gioco scalabile.

Si vuole rendere fruibile il gioco tramite client web mobile, tablet e desktop, per permettere a più giocatori possibile di giocare garantendo totale libertà riguardo al device scelto; ne segue dunque che il client sarà realizzato seguendo una filosofia mobile first.

Sarà possibile registrarsi al servizio, anche al fine di fornire un sistema di notifiche riguardo ad alcune attività e recuperare dati riguardanti le partite precedentemente svolte.

Inoltre, verrà fornita una chat con la quale i giocatori possano comunicare durante la partita.

2.1 Obiettivi

Per la prioritizzazione dei nostri obiettivi utilizzeremo il metodo MoSCoW:

1. *MUST*:

- Modellazione di tutti i sistemi tramite rappresentazioni di Ingegneria del Software
- Gestione delle lobby
- Gestione degli stati di gioco
- Autentificazione
- Notifiche
- Test con utenti
- Responsiveness
- Accessibilità

2. *SHOULD*:

- Implementazione di un sistema di messaggistica
- Possibilità di consultare i report delle partite passate
- Feedback grafici (popup)

3. *WOULD*:

- Animazioni
- Feedback audio

4. *WON'T*:

- Possibilità di scaricare in locale i report delle partite passate

2.2 Personas

Le personas sono veri e propri identikit di utenti ideali interessati al servizio offerto. Sono una rappresentazione dei tratti caratterizzanti di ciascun utente e di quelli che li accomunano.

Per GuessR, sono state individuate 3 personas che identificano gli utenti target:

2.2.1 Jonathan



Profilo personale:

- Età: 14 anni
- Sesso: maschile
- Status: vive con i suoi genitori e, durante le sere infrasettimanali, si collega con i suoi amici tramite Discord per giocare a vari videogiochi

Occupazione/Formazione:

- Frequenta l'ultimo anno di scuole medie, dove si concentrano anche la maggior parte dei suoi amici, molto simili a lui. Non dispone di un reddito.

Comportamento online:

- Tempo su internet: almeno 20 ore a settimana
- Utilizza internet per: messaggistica istantanea, social network, videogiochi, cercare notizie calcistiche

Interessi personali:

- Videogiochi: predilige i giochi di gruppo, ma occasionalmente gioca anche a dei single-player
- Calcio: segue tutte le notizie della sua squadra e frequenta un'associazione calcistica locale per dilettanti

2.2.2 Jolyne



Profilo personale:

- Età: 22 anni
- Sesso: femminile
- Status: studentessa universitaria fuori sede, si riunisce spesso con i suoi amici per fare serata e, in quelle occasioni, necessita di giochi di gruppo semplici e divertenti

Occupazione/Formazione:

- Frequenta la triennale di psicologia, di tanto in tanto lavora in un ristorante a chiamata. Dispone di un reddito basso occasionale.

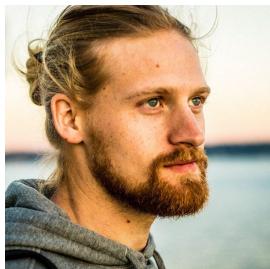
Comportamento online:

- Tempo su internet: 15 ore a settimana
- Utilizza internet per: messaggistica istantanea, social network, studio universitario

Interessi personali:

- Psiche umana: da sempre affascinata, sviluppa questa passione tramite il percorso di studi
- Arte: di tanto in tanto si cimenta in qualche illustrazione per svagarsi

2.2.3 Joseph



Profilo personale:

- Età: 27 anni
- Sesso: maschile
- Status: convive con la sua compagna in un appartamento, occasionalmente si connette tramite Skype con gli amici della sua città natale

Occupazione/Formazione:

- Dopo aver terminato le scuole superiori ha cominciato una carriera da impiegato d'ufficio e, una volta ricevuta una promozione, l'azienda l'ha trasferito fuori sede

Comportamento online:

- Tempo su internet: almeno 30 ore a settimana
- Utilizza internet per: messaggistica istantanea, social network, eseguire pratiche burocratiche a lavoro, riempire i momenti di vuoto lavorativo

Interessi personali:

- Film e serie TV: li utilizza la sera per rilassarsi
- Videogiochi: altra alternativa serale, videogiocatore dall'alba dei tempi

Capitolo 3

Design

3.1 Struttura del Server

Il server è strutturato principalmente in due macro-parti. Una parte del server si occupa di gestire tutte le richieste HTTP che provengono dal client, l'altra si occupa di gestire i messaggi relativi alle web-socket.

3.1.1 Gestione delle Route

Le route del server sono divise in delle categorie. La divisione è stata pensata solo per una separazione maggiore delle responsabilità, soprattutto per quanto riguarda la logica e il comportamento dei controller associati alle rotte. Le rotte denotate con (auth middleware) richiedono il token ottenuto durante la fase di login all'interno dell'header Authorization della richiesta.

- Auth: Contiene tutte le rotte relative all'autenticazione.
 - **POST - /auth/signup**: Crea un nuovo utente (se non già esistente) utilizzando username e password presenti nel body della richiesta.
 - **POST - /auth/login** : Controlla se username e password coincidono con un qualche utente sul DB. In caso di esito positivo si ritorna un JWT.
 - **GET - /profile (auth middleware)**: Ritorna le informazioni dell'utente a partire dal token.
- Notification:
 - **GET - /notification (auth middleware)**: Recupera le notifiche di un utente a partire dal suo token.
 - **POST - /notification (auth middleware)**: crea una nuova notifica per l'utente a cui appartiene il token presente nella richiesta.
 - **DELETE - /notification:id (auth middleware)**: cancella la notifica con l'id specificato nel parametro di query.
- Resources: Contiene le rotte relative alle risorse che offre il sistema.
 - **GET - /languages (auth middleware)**: Recupera le lingue disponibili per il gioco.
 - **GET - /dw/report:id (auth middleware)**: scarica il report con l'id specificato nel parametro di query.
 - **GET - /report (auth middleware)**: recupera tutte le informazioni sui report di tutti i giocatori che hanno giocato nelle stesse partite in cui ha giocato l'utente che fa la richiesta,

3.1.2 Gestione delle Web Socket

Quando il server riceve o invia un messaggio tramite websocket, esso è associato ad uno specifico canale.

- **createLobby**: È il canale su cui vengono inviate le richieste da parte del client per creare nuove lobby. Il server risponde inviando il codice al client in caso di successo;

- **joinLobby:** È il canale su cui vengono inviate le richieste da parte del client per entrare in una lobby. Il server aggiunge l'utente alla lobby ed invia come risposta tutte le informazioni relative alla lobby, come la lingua, il codice, la lista degli utenti etc;
- **startGame:** È il canale su cui vengono inviate le richieste da parte del client per iniziare una nuova partita. Solo l'admin della lobby può inviare un messaggio su questo canale. Il server si occupa di inviare un messaggio di broadcast a tutti gli utenti nella lobby, cambiando il loro stato interno e facendo iniziare la partita.
- **sentence:** È il canale su cui vengono inviate le frasi scritte dagli utenti durante una partita.
- **draw:** È il canale su cui vengono inviati i disegni fatti dagli utenti durante una partita.
- **forwardData:** È il canale utilizzato per gestire il sistema di ACK in modo da mantenere consistenza nello stato del sistema.
- **chat:** È il canale su cui vengono inviati i messaggi scritti fatti dagli utenti durante una partita.
- **endGame:** È il canale su cui vengono inviate le richieste da parte del client per iniziare terminare la partita ed uscire dalla fase di visualizzazione dei report. Solo un utente admin può inviare un messaggio su questo canale. Il server si occupa di inviare un messaggio di broadcast a tutti gli utenti nella lobby, cambiando il loro stato interno e spostandoli nella pagina di attesa della lobby.
- **disconnect:** È il canale su cui viene sempre inviato un messaggio se la socket viene chiusa (volontariamente o anche a causa di crash). Il canale è molto utile perché permette di mantenere consistenza in tutti i casi possibili, modificando lo stato e notificando in tempo reale la situazione anche ad eventuali altri utenti nella lobby.

3.1.3 Stato del Server

La gestione delle partite richiede l'utilizzo di alcune strutture dati che non hanno a che fare con i dati salvati all'interno del Database. Nello specifico è molto importante modellare il sistema delle Lobby, i giocatori che ne fanno parte, ed i report dei giocatori che si arricchiscono mano a mano che la partita si avvicina alla sua fine. Inoltre si vorrebbero gestire i messaggi testuali che è possibile mandare mentre ci si trova all'interno di una lobby.

Nello stato interno di Redux è presente solo una Mappa, le cui chiavi sono i codici univoci che identificano la lobby mentre i valori sono gli oggetti Lobby.

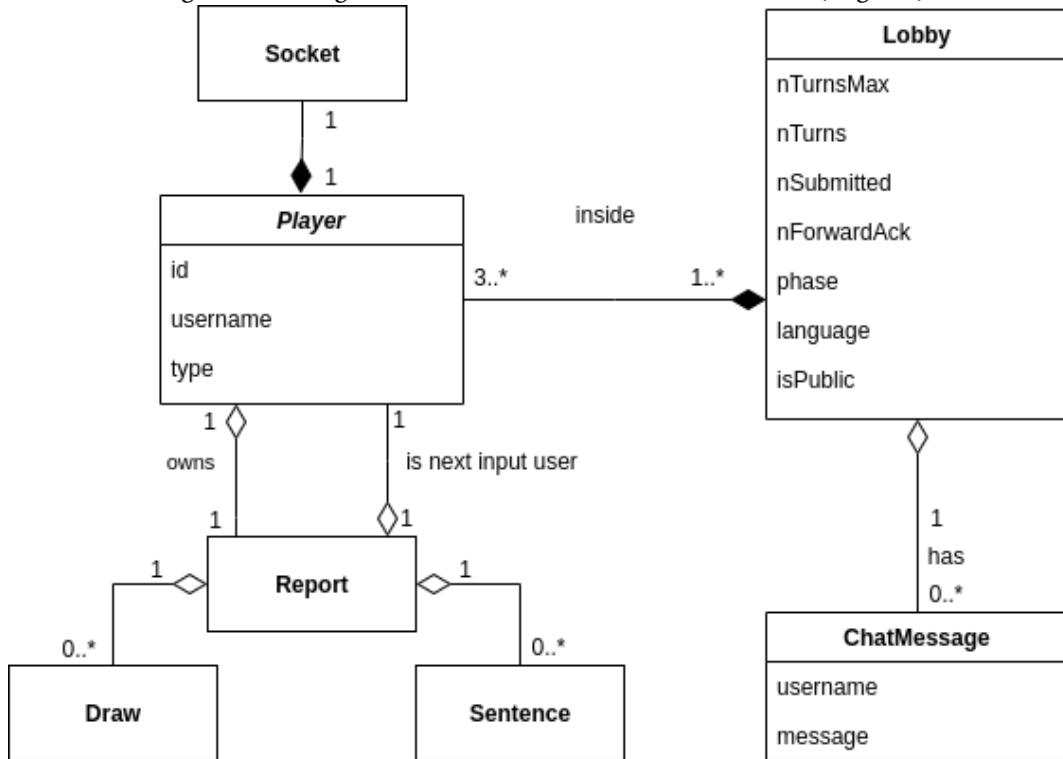
Ad ogni giocatore viene anche collegata la relativa socket in modo da poter inviare messaggi al client dell'utente una volta recuperata.

Lo stato delle lobby viene modificato solo attraverso gli handler delle socket, la gestione delle partite è infatti completamente gestita attraverso l'uso delle web-socket.

Di seguito una breve descrizione delle classi del model:

- **Lobby:** La classe principale del model che contiene tutte le altre. Contiene anche le informazioni generali della lobby come la lingua o se si tratta di una lobby pubblica o privata.
- **Player:** Rappresenta un giocatore all'interno della Lobby, il campo type denota se si tratta di un Admin, ovvero dell'utente che può far iniziare la partita.
- **ChatMessage:** Rappresenta un messaggio nella chat della Lobby.
- **Report:** Ad ogni utente è associato un Report, ovvero il foglio virtuale utilizzato nel gioco in cui vengono inserite le frasi e i disegni. Il report appartiene ad un determinato utente, ma viene anche passato ad altri utenti quando essi devono aggiungere contenuti.
- **Socket:** L'oggetto socket utilizzato dalla libreria socket.io associata all'utente.
- **Draw/Sentence:** Entrambe le classi sono in realtà stringhe. Draw è una stringa xml che rappresenta il disegno utilizzando lo standard SVG.

Figura 3.1: Diagramma delle classi del model del Server (in gioco)



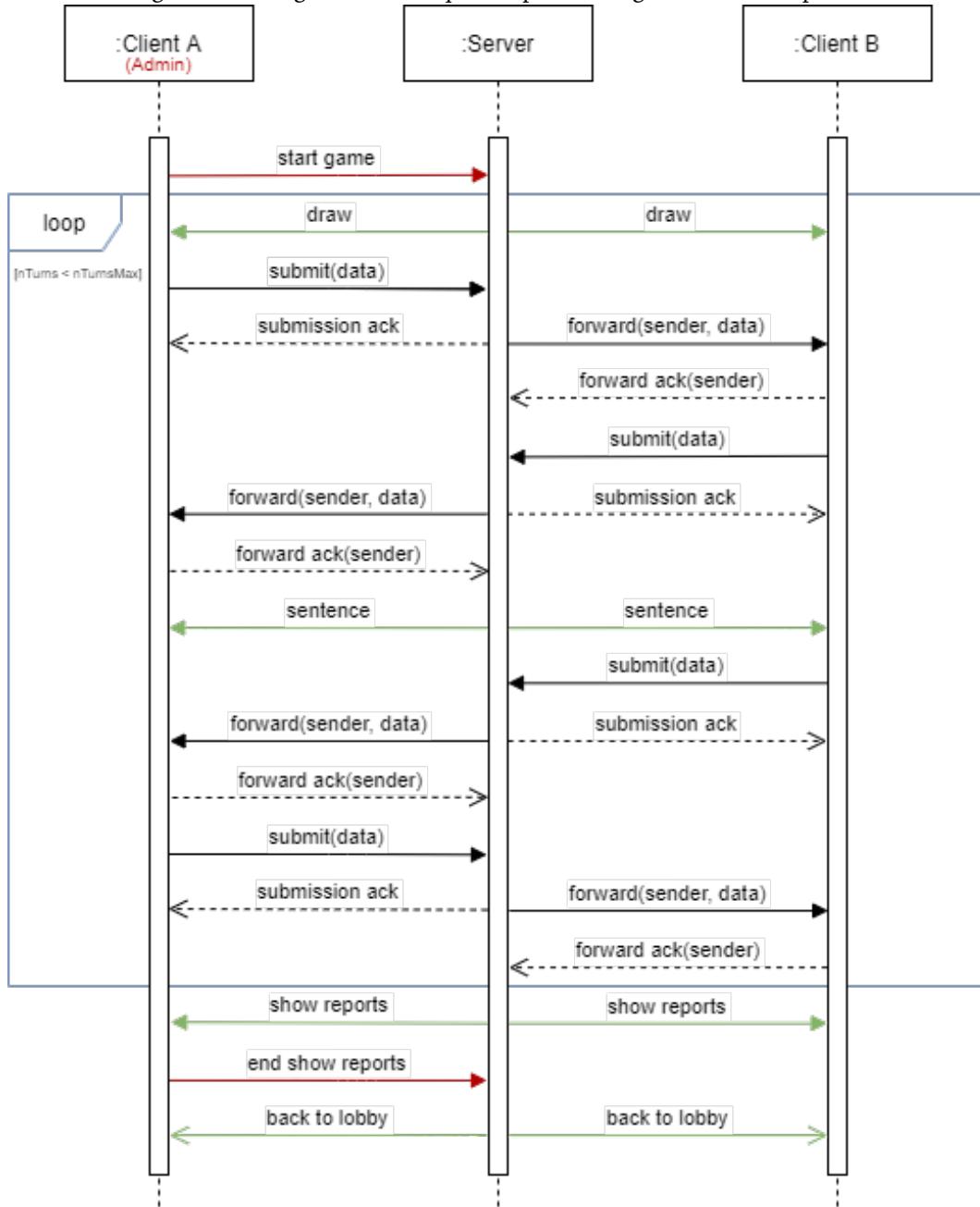
3.1.4 Gestione di una partita

Per esplicare la gestione della comunicazione tra il server ed i vari client all'interno di una partita, verrà usato un diagramma di sequenza; in tale diagramma sono coinvolti il server, un client A che svolge il ruolo di Admin della lobby ed infine un secondo client B.

Il gioco richiede in realtà un numero minimo di giocatori pari a 3, ma per semplificare la rappresentazione l'esempio coinvolge solo 2 giocatori.

In rosso troviamo i messaggi inviati dall'Admin volti a modificare attivamente lo stato della lobby.
 In verde invece sono rappresentati i messaggi inviati dal server tramite i quali i vari client modificano le proprie UI, facendo progredire il gioco.
 In nero infine, sono presenti i messaggi dedicati allo scambio di frasi e disegni, input dei giocatori.

Figura 3.2: Diagramma di sequenza per lo svolgimento di una partita



Si presuppone che A e B si trovino all'interno di una lobby; durante la creazione di quest'ultima A ha impostato un numero di turni, ad esempio 3. Tale numero di turni verrà salvato nella variabile *nTurnsMax*.

A premerà dunque il pulsante start game che scatenerà l'invio al server dell'omonimo messaggio; viene impostata a zero una variabile *nTurns* la quale contiene il numero di turni trascorsi.

Il server dunque esegue una broadcast inviando sul namespace *draw* un messaggio. Alla ricezione del messaggio i client modificano la UI permettendo ai giocatori di scrivere la loro prima frase.

In questo esempio A scrive ed invia una frase prima di B; egli esegue dunque una submit inviando i suoi dati (la frase) al server che risponde gli con un *submission ack*. Contemporaneamente il server invia inoltre, tramite *forward*, i dati a tutti i membri della lobby TRANNE il mittente (A). Mentre B continua a scrivere la sua frase, il client in background salva i dati ricevuti ed invia un *forward ack* al server.

Ad un certo punto anche B invierà la sua frase, ripetendo messaggi analoghi a quelli sopra descritti.

Dato N come il numero di giocatori, per verificare che tutti i giocatori abbiano inviato la loro frase e che ogni client disponga di una copia della frase di tutti gli altri giocatori, il server verifica di aver ricevuto un numero di forward ack pari ad $N^*(N-1)$.

Il secondo termine nella moltiplicazione è minore di 1 rispetto al primo in quanto A salverà in locale la propria frase e dunque non riceverà un forward per la frase da lui inviata e, conseguentemente, non invierà un forward ack (in un esempio con 3 client A, B e C il server avrebbe ricevuto due ack per la frase di A, due per la frase di B ed altri 2 per la frase di C).

Verificata tale condizione, il server invia in broadcast a tutti i giocatori un messaggio sul namespace *sentence*, sentenziando la fine della fase dove i giocatori scrivono una frase.

I client, ricevuto il messaggio, modificano nuovamente la loro UI per offrire la possibilità ad i giocatori di creare un disegno.

Analogamente a prima, i giocatori inviano i loro dati (questa volta disegni) e, una volta ricevuto il giusto numero di forward ack, il server incrementa nTurns di 1; se nTurns < nTurnsMax, il server effettuerà una broadcast sul canale draw, sentenziando la fine della fase di disegno. I client modificheranno dunque nuovamente le loro UI per permettere l'input di una frase, ricominciando il ciclo.

Nel caso invece in cui sia stato raggiunto il numero di turni impostati, il server eseguirà una broadcast sul namespace *show reports*, segnalando ai client di mostrare a schermo i report di tutti gli utenti costruiti passo per passo tramite le precedenti forward.

Quando l'Admin A lo riterrà più opportuno (solo egli avrà questa possibilità), segnalerà al server la sua volontà di terminare la visione dei report. Il server dunque eseguirà una broadcast sul namespace *back to lobby*, riportando tutti i client alla lobby iniziale, dove l'Admin potrà nuovamente iniziare una nuova partita.

3.1.4.1 Possibile implementazione alternativa semplificata e vantaggi derivanti da quella adottata

In un'implementazione alternativa, sarebbe stato possibile inviare ad ogni client solo il dato (frase o disegno) necessario per la prossima fase, inviando in broadcast solamente alla fine tutti i dati di tutti i report. Seppur tale implementazione sarebbe stata più semplice, ne avrebbe perso la qualità dell'esperienza utente, incrementando i tempi di attesa.

Nella soluzione adottata, per ridurre i tempi derivanti dal download dei dati (specialmente dei disegni, che risultano molto più pesanti delle frasi), tale operazione viene effettuata in background sia mentre i giocatori più veloci nella creazione di frasi/disegni attendono quelli più lenti, sia mentre questi ultimi stanno per l'appunto completando il loro input.

Nell'implementazione più semplice, invece, tutti i giocatori sarebbero stati costretti ad un'attesa più lunga dovuta ad una grande quantità di informazioni inviata ad ogni client. Per garantire la sincronia tra tutti i client, inoltre, anche i giocatori con una connessione veloce avrebbero dovuto attendere che quelli con una connessione lenta effettuassero il download di tutte le informazioni necessarie, vincolando l'attesa al client con la connessione più lenta.

La soluzione adottata dunque elimina quasi totalmente il problema spalmando il download nel tempo e sfruttando i tempi morti che naturalmente si vengono a creare durante il gioco nell'attesa che tutti i giocatori abbiano terminato di creare il loro input.

3.2 Struttura del Client

Il client di GuessR è stato realizzato utilizzando il framework React.

Il client è strutturato principalmente in tre macro-parti: Componenti, Stato ed Handler delle Socket.

3.2.1 Componenti

La struttura delle pagine dell'applicazione si trova dentro i componenti di React. Per distinguere un componente da un normale file di Javascript basta guardare la lettera iniziale nel nome del file. Se è maiuscola si tratta di un componente React, altrimenti di un semplice modulo javascript.

I componenti di React sono riutilizzabili a loro volta in altri componenti, permettendo di applicare il principio DRY anche in aspetti di strutturazione e non solo di logica.

React nasce come framework per single page application. Per aggiungere altre route al sistema è necessaria la libreria *React Router*, la quale mette a disposizione anche strumenti per effettuare il redirect utilizzando la logica base di React, andando a preservare lo stato interno del client.

La suddivisione dei componenti è stata effettuata in base alla pagina in cui essi vengono utilizzati. I componenti comuni si trovano in una cartella denominata *common*. I componenti sono divisi per pagina.

3.2.2 Stato del Client

Lo stato dell'applicazione all'interno del client è stato incapsulato quasi completamente all'interno di Redux.

Gli stati di React sono comunque stati utilizzati in maniera minore per gestire comportamenti della GUI o altri dettagli minori.

All'interno dello stato di Redux del client sono incapsulati quattro oggetti:

- **userInfo**: contiene tutte le informazioni dell'utente che ha effettuato l'accesso, in particolare:
 - **id**: l'Id univoco dell'utente;
 - **username**: il nome utente dell'utente;
 - **token**: il web token dell'utente fornito dal server durante la fase di autenticazione;
 - **notifications**: tutte le notifiche dell'utente;
- **util**: contiene oggetti di utilità generale, in particolare:
 - **languages**: l'elenco delle lingue disponibili per la creazione di una lobby;
 - **socket**: la socket (oggetto della libreria socket.io) associata all'utente;
 - **isLoading**: un campo booleano utile per far comparire una rotellina di caricamento a schermo in caso di attesa.
- **lobby**:
 - **settings**: contiene le impostazioni della lobby impostabili nel momento della creazione, in particolare:
 - * **isPublic**: true se la lobby è pubblica, false altrimenti;
 - * **nTurns**: il numero di turni che durerà la partita.
 - * **language**: la lingua che usata all'interno della lobby;
 - **info**: contiene alcune informazioni della lobby, in particolare:
 - * **code**: il codice univoco della lobby;
 - * **users**: la lista degli utenti all'interno della lobby.
 - * **isMyRoleAdmin**: true se l'utente è admin della lobby, false altrimenti.
 - **status**: la fase in cui si trova la lobby.
 - **waitingAllSubmit**: true se durante il gioco si sta ancora aspettando che almeno un utente invii la sua frase o il suo disegno.
 - **messages**: l'elenco di messaggi presenti sulla chat.
 - **receivedData**: la frase o il disegno ricevuto e su cui si baserà la risposta da inviare.
 - **reports**: la lista di tutti i report che vengono popolati durante le fasi di gioco.
- **previousReports**: contiene la lista dei report visionati nelle partite già giocate in precedenza.

3.2.3 Gestione delle socket

Quando il client riceve o invia un messaggio tramite websocket, esso è associato ad uno specifico canale.

- **joined**: il canale su cui il server invia tutte le informazioni della lobby se l'utente è riuscito ad accedervi.
- **players**: il canale su cui il server invia tutte le informazioni riguardanti la connessione o la disconnessione da parte di altri giocatori all'interno della lobby.
- **chat**: il canale su cui il server invia tutte le informazioni riguardanti nuovi messaggi che vengono scritti all'interno della chat.
- **sentence**: il canale su cui il server invia la frase che l'utente ricevente dovrà disegnare.
- **draw**: il canale su cui il server invia il disegno che l'utente ricevente dovrà interpretare e descrivere con una frase.
- **forwardData**: il canale utilizzato per ricevere dati da parte del server riguardanti i report che vengono aggiornati.
- **showReport**: il canale su cui il server invia un messaggio quando la partita è finita e si possono visualizzare i risultati finali.
- **backToLobby**: il canale su cui il server invia la conferma per poter tornare nella fase di attesa all'interno della lobby una volta cambiato lo stato interno.

3.3 Struttura del Database

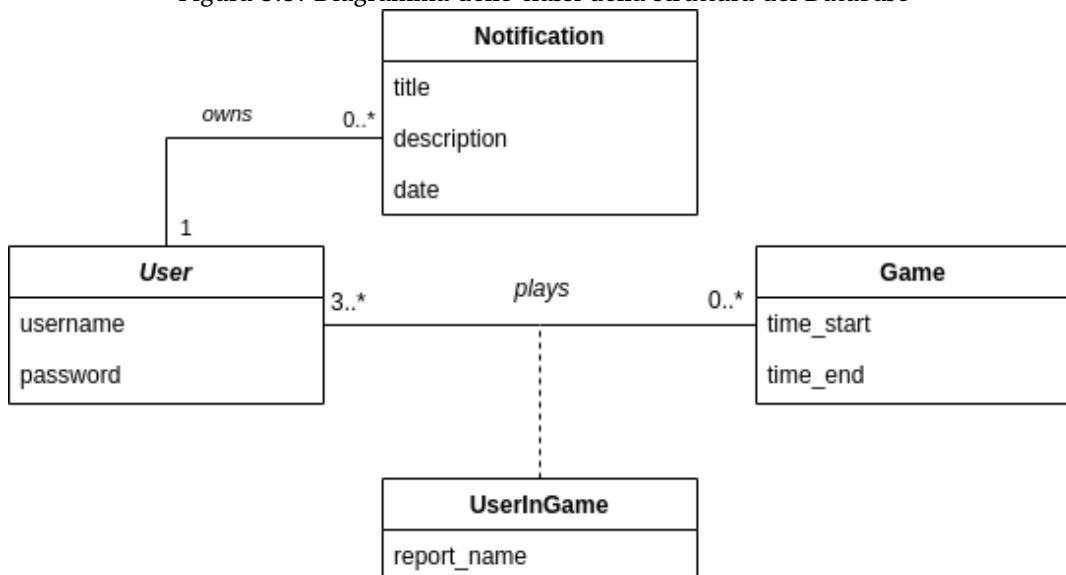
GuessR non richiede un Database particolarmente articolato.

Una classe *User* è necessaria per archiviare gli utenti e permettere la creazione di un sistema di autenticazione.



ATTENZIONE: il campo password di User non contiene la password in chiaro, ma l'hash della password generata dal server.

Figura 3.3: Diagramma delle classi della struttura del Database



Per aggiungere un sistema di notifiche all'applicazione è necessaria la creazione di una classe *Notification*, legata a *User* con cardinalità uno a molti, in quanto un utente può avere un numero qualsiasi di notifiche,

ma la notifica è specifica per il singolo utente.

La classe *Game* rappresenta la partita a cui almeno tre utenti hanno preso parte. La cardinalità tra *User* e *Game* è molti a molti, in quanto ad una partita possono partecipare molti giocatori (almeno 3), mentre i giocatori possono aver giocato innumerevoli partite, o anche nessuna.

La classe associativa *UserInGame* rappresenta lo specifico utente che ha preso parte alla specifica partita. Il campo *report_name* è il nome del documento (situato nel server, è possibile scaricarlo attraverso il client) contenente il report finale.

3.4 Interfaccia Utente

3.4.1 Wireframe

In questa sezione verranno illustrati i Wireframe dell'applicazione, realizzati durante la fase di progettazione. Tutte le schermate dell'applicazione sono state progettate con un'ottica mobile-first, ovvero puntando all'ottimizzazione del design su dispositivi mobili.

In tutte le schermate compare una NavBar in cui compare il Logo dell'applicazione, una Label che indica il nome della pagina in cui ci si trova e un'icona rappresentante operazioni possibili per l'utente (per esempio il Logout).

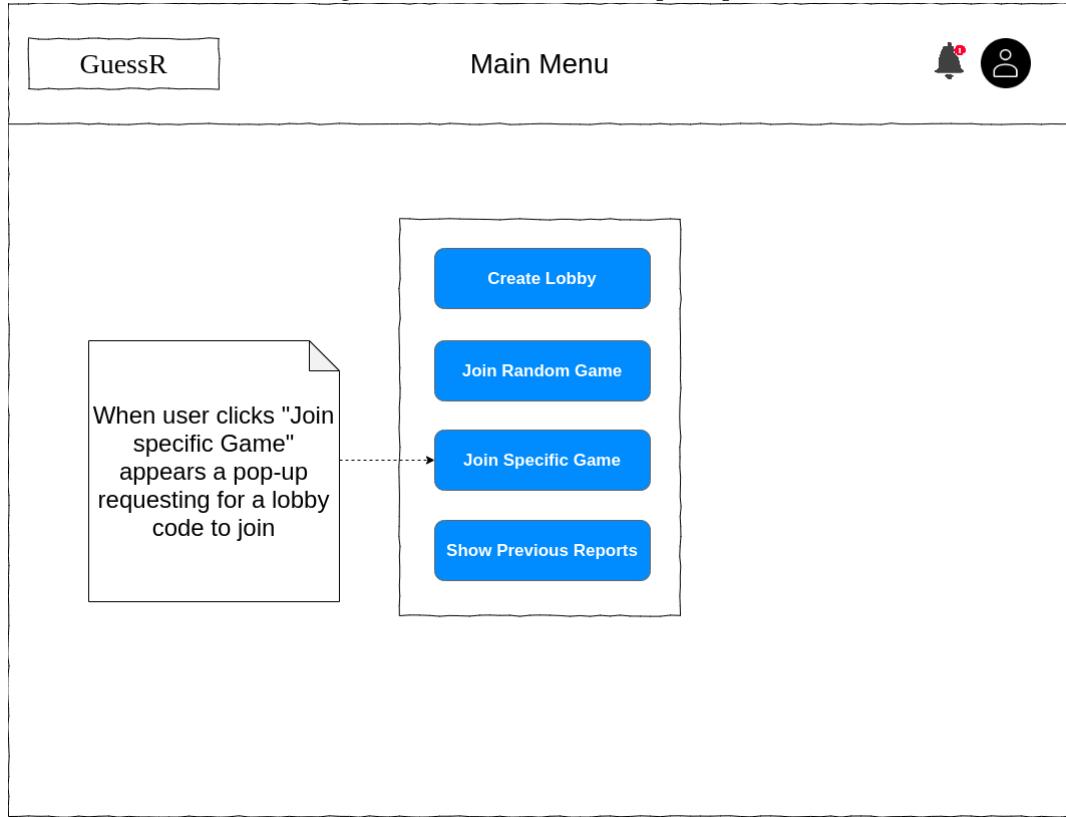
La prima pagina che compare quando si accede all'applicazione è la schermata di Autenticazione (Wireframe mancante). In questa schermata è possibile effettuare la registrazione di un nuovo account o il login. Qualora si effettuasse una registrazione verrà effettuato in automatico anche il Login. Inserendo le proprie credenziali in modo corretto e cliccando sul pulsante Login si potrà accedere al menù principale.

In caso di credenziali sbagliate l'utente viene avvertito e viene invitato a provare di nuovo.

Nella schermata del menù Principale l'utente può cliccare su quattro pulsanti.

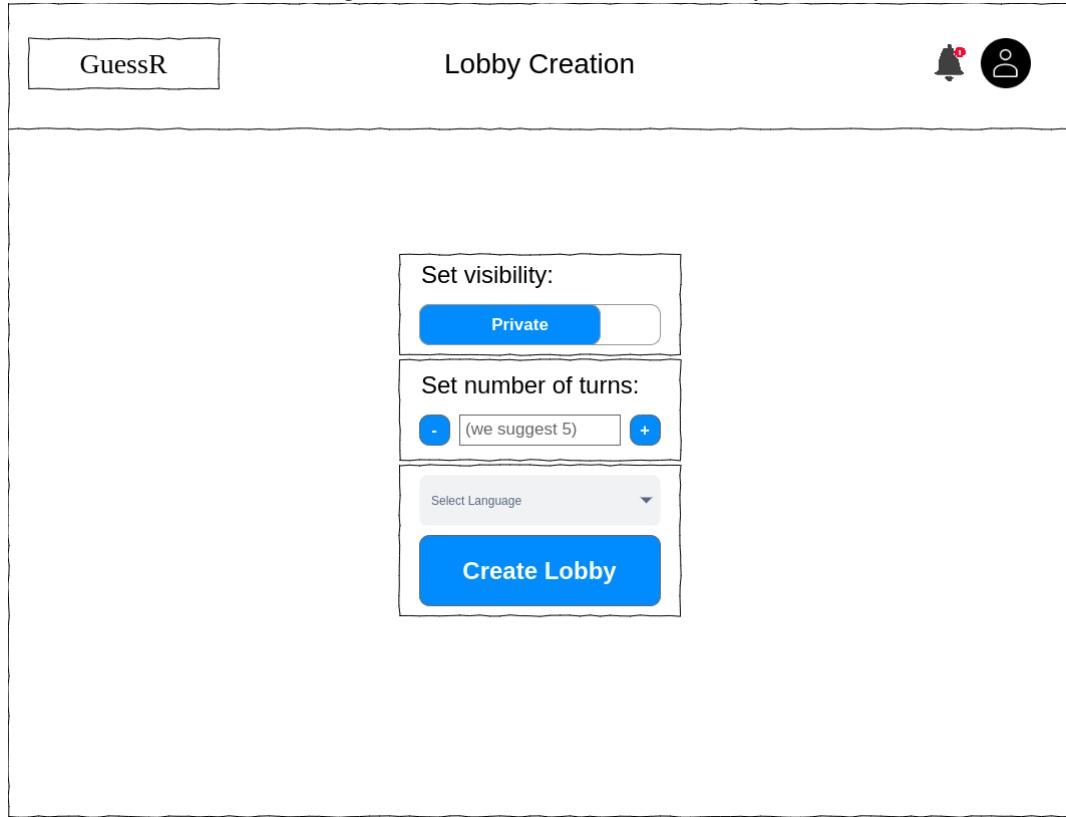
- **Create Lobby:** Reindirizza l'utente alla pagina di creazione di una Lobby;
- **Join Random Lobby:** Fa comparire una schermata in cui l'utente deve inserire la propria lingua per poter entrare una Lobby casuale in cui gli utenti parleranno la sua stessa lingua;
- **Join Specific Game:** Fa comparire una schermata in cui l'utente deve inserire il codice identificativo di una lobby già creata;
- **Show previous Report:** Reindirizza l'utente alla pagina in cui può scaricare e consultare i report delle partite che ha giocato in passato.

Figura 3.4: Wireframe menu principale



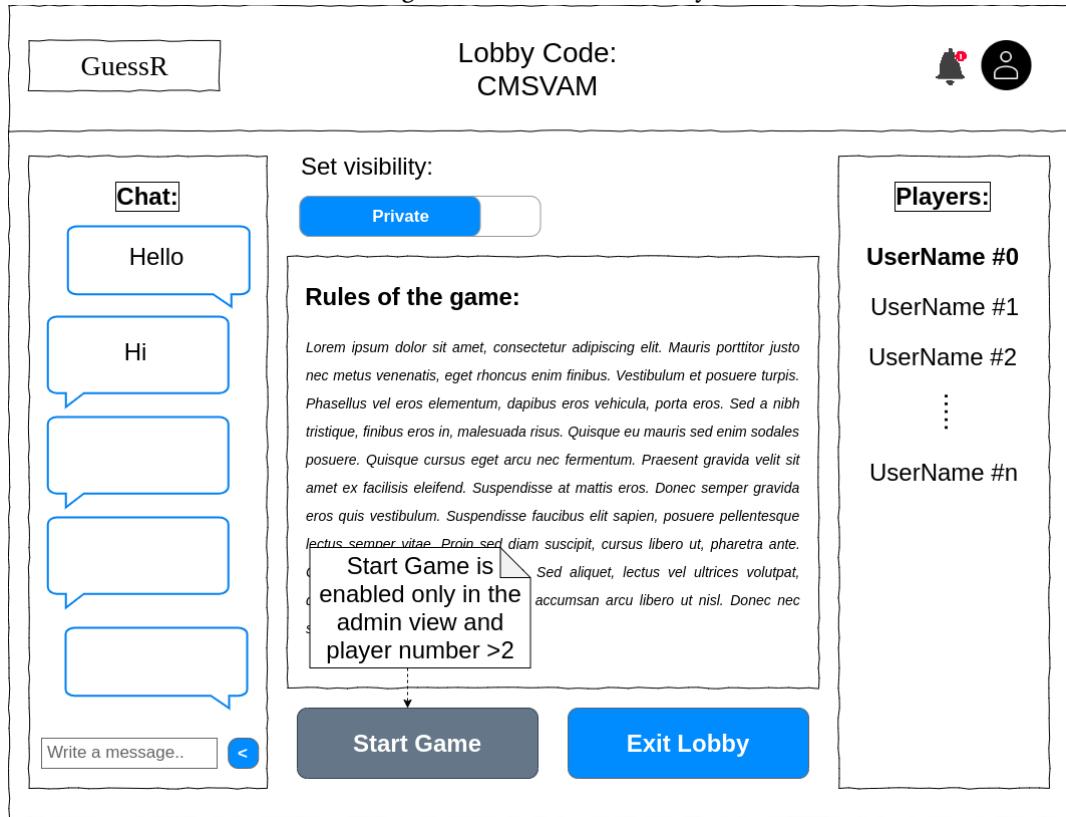
Premendo sul pulsante *Create Lobby* compare la pagina di creazione di una Lobby. Questa pagina contiene un Form in cui è possibile impostare i parametri della Lobby. L'interruttore “Set Visibility” permette di impostare la visibilità della Lobby. Se si imposta la Lobby come privata solo gli utenti che possiedono il codice di accesso possono accedere. Se si imposta la lobby come pubblica qualsiasi utente potrà entrare nella Lobby. È poi possibile inserire il numero di turni che si desidera giocare (un turno è definito come l'input sequenziale di una frase e un disegno da parte di tutti gli utenti). Infine è necessario selezionare la lingua in cui si vuole giocare tra le possibili offerte. Il pulsante “*Create Lobby*” reindirizza l’utente alla pagina in cui attende che altri utenti accedano alla lobby per poter giocare.

Figura 3.5: Wireframe creazione lobby



Nella schermata della Lobby creata compare il codice univoco assegnato alla Lobby sulla NavBar. Nella parte centrale dello schermo vengono illustrate le regole del gioco. Nella parte sinistra dello schermo compaiono i messaggi inviati dagli utenti ed il form utilizzato per mandarli. Nella parte destra dello schermo compare la lista dei giocatori all'interno della Lobby. Infine, sotto le regole del gioco è presente un pulsante "Exit Lobby" che se premuto fa uscire l'utente dalla Lobby. Un pulsante aggiuntivo "Start Game" compare solo all'utente che ha creato la Lobby (l'utente di tipo Admin). Se premuto, questo pulsante reindirizza tutti gli utenti all'interno della Lobby alla schermata di Gioco.

Figura 3.6: Wireframe lobby

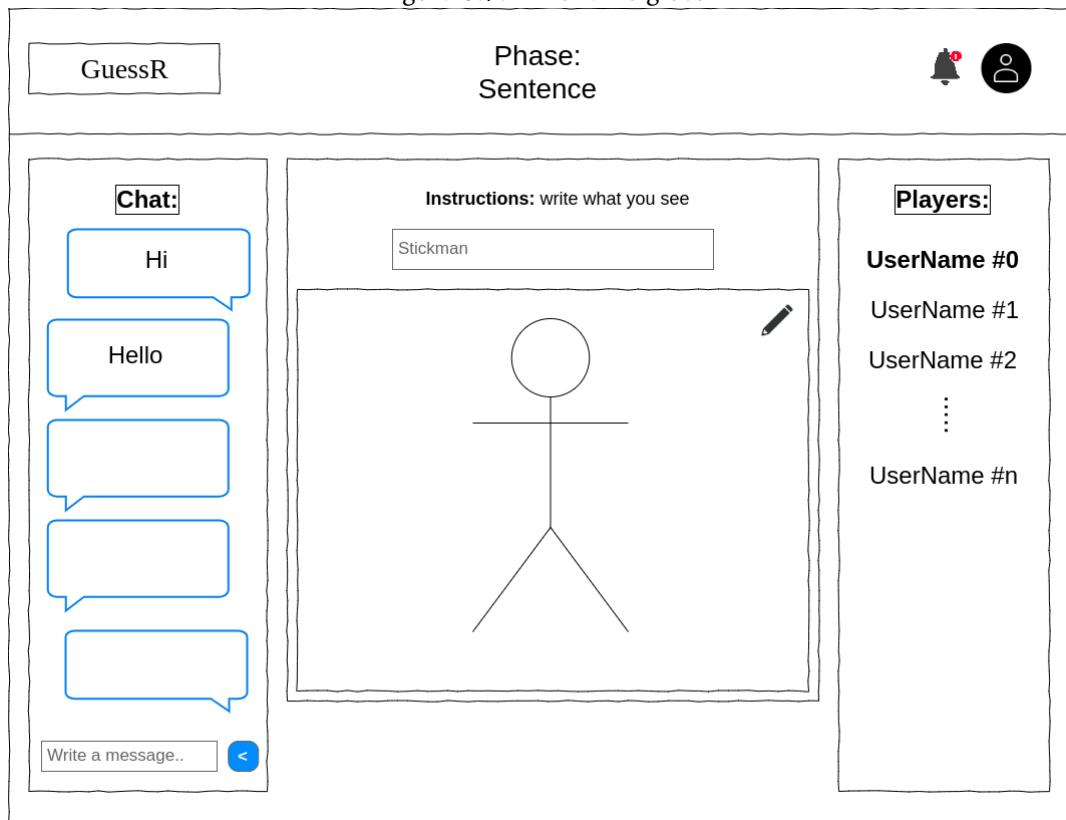


Nella schermata di gioco la parte destra e sinistra dello schermo rimane immutata rispetto alle pagine precedente. È quindi ancora possibile consultare ed inviare messaggi sulla chat e vedere i giocatori che stanno giocando.

Sulla NavBar viene illustrata la fase di gioco.

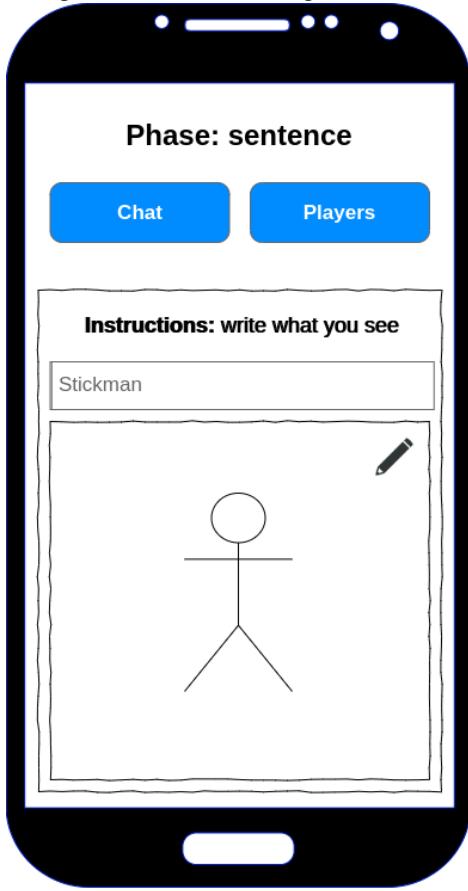
- Se la fase è “Sentence” l’utente dovrà digitare una frase a suo piacimento (al primo turno o nel caso in cui non abbia ricevuto alcun disegno) o una frase che rappresenta in disegno che vede a schermo nel box di testo.
- Se la fase è “Draw” l’utente dovrà disegnare come meglio possibile ciò che legge nella frase ricevuta. Il disegno va eseguito nell’apposito box bianco messo a disposizione.

Figura 3.7: Wireframe gioco



Quando il numero di turni finisce o un utente si disconnette tutti gli utenti vengono reindirizzati alla pagina in cui vengono visualizzati tutti i fogli (o report). Anche in questa pagina è presente un pulsante "Exit Lobby" per uscire dalla Lobby. L'utente che ha creato la lobby ha a disposizione un ulteriore pulsante "Back to Lobby" che reindirizza tutti gli utenti alla pagina in cui si attende che altri giocatori entrino nella Lobby.

Figura 3.8: Wireframe gioco mobile



La maggior parte delle pagine contengono poco contenuto, fatta eccezione per le pagine in cui compaiono la chat e la lista degli utenti.

Proprio questi due elementi, nella versione mobile dell'applicativo vengono condensati in due buttoni. Cliccando sui buttoni "Chat" e "Players" si apriranno dei pop-up che mostreranno le informazioni in un formato più consono alla dimensione dello schermo.

3.4.2 Accessibilità

Nello svolgimento del presente elaborato uno degli aspetti su cui ci siamo focalizzati maggiormente è quello dell'accessibilità, per poter permettere a più utenti possibile di interagire con l'applicativo giocando, e favorire l'inclusione anche di persone che fanno uso di screen reader. A tale scopo abbiamo cercato di uniformarci più possibile ai criteri di buona programmazione web e utilizzando delle best practices.

Sia in fase di design dell'applicativo, design dell'interfaccia utente e della user experience, sia in fase di sviluppo abbiamo cercato di rendere l'applicativo quanto più accessibile, anche facendo ricorso a tool come axe devTools (set di tools leader nel settore, utilizzato anche da Microsoft e Google) che grazie alle sue linee guida ci ha permesso di focalizzare gli aspetti più importanti, come il giusto contrasto di colore, la zoomabilità delle pagine (soprattutto in caso di accesso da dispositivo mobile), le corrette proporzioni del testo e soprattutto una corretta integrazione di tag per agevolare l'utilizzo di screen readers o screen magnifiers.

Guardando un risultato finale riteniamo di poter affermare dell'essere riusciti nell'intento di costruire un sito accessibile, in grado di includere persone ipovedenti o affette da daltonismo e adatto anche a chi ha necessità di ricorrere ad altri mezzi per la navigazione.

3.4.3 Schermate finali

Seguono gli screen relativi alle schermate di gioco definitive, effettuati durante una effettiva partita tra le personas descritte precedentemente.

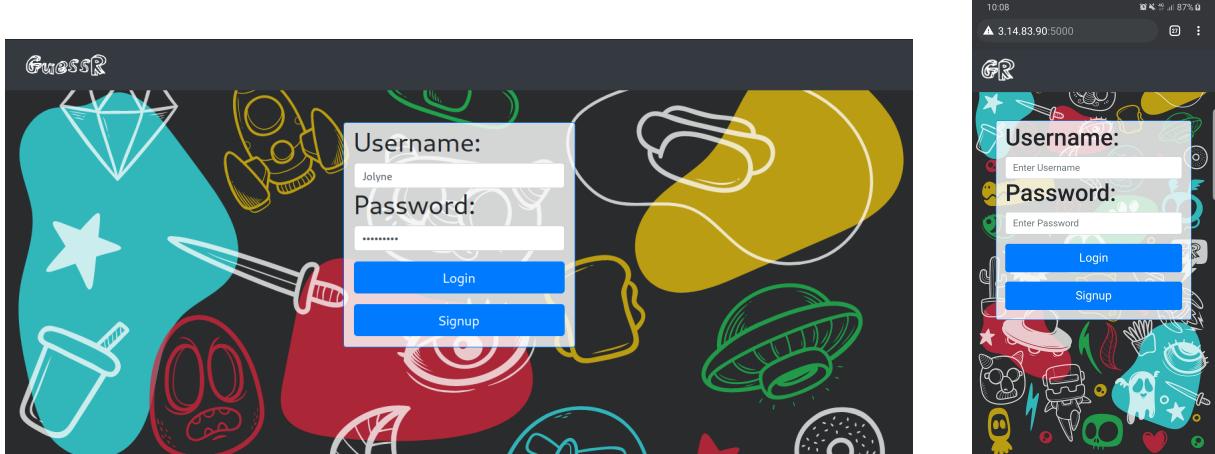


Figura 3.9: Schermate di Login (Desktop e Mobile)

La schermata di login presenta una form per l'autenticazione dell'utente, in cui dovrà essere inserita uno username univoco ed una password che deve rispettare i seguenti criteri: lunghezza maggiore o uguale a 8 caratteri, almeno una lettera maiuscola ed una minuscola e almeno un numero. Nei casi in cui questi criteri non vengono rispettati l'utente verrà notificato un apposito dialog.

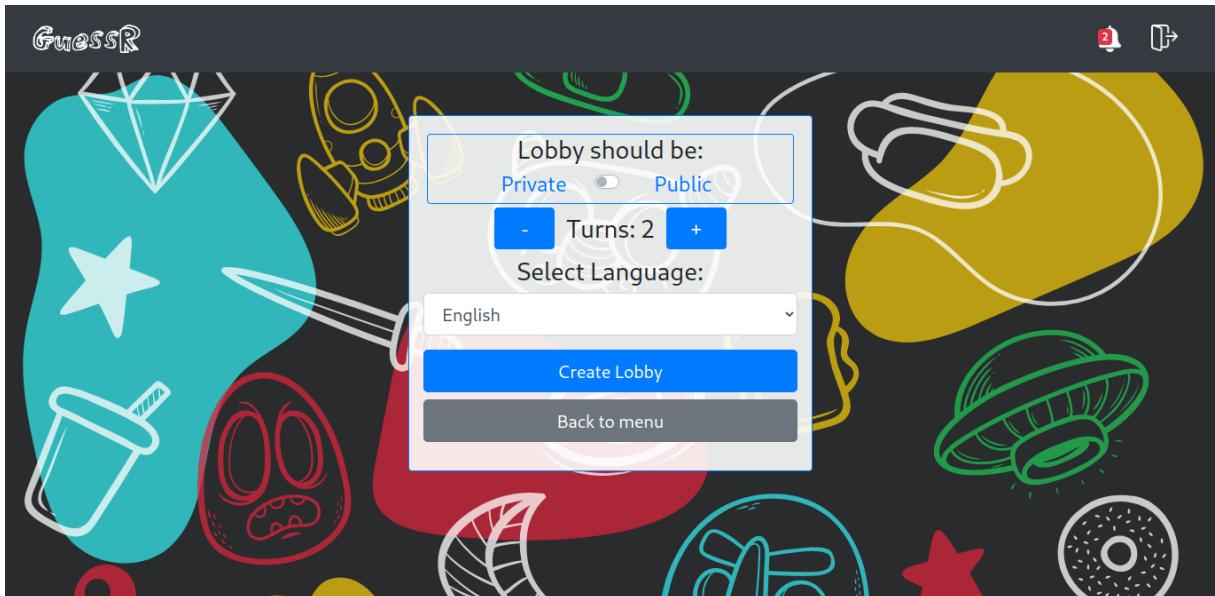


Figura 3.10: Schermata per la creazione di una nuova lobby

Nella schermata per creare una nuova lobby, un utente dovrà specificare la visibilità della lobby (se pubblica e quindi accessibile solo specificando una lingua o privata, quindi accessibile solo se a conoscenza del codice univoco). Successivamente l'utente selezionerà la lingua con cui i partecipanti vorranno comunicare ed il numero di turni di gioco. Una volta creata la lobby, l'utente che ha affrontato il processo di creazione sarà admin della lobby appena creata.

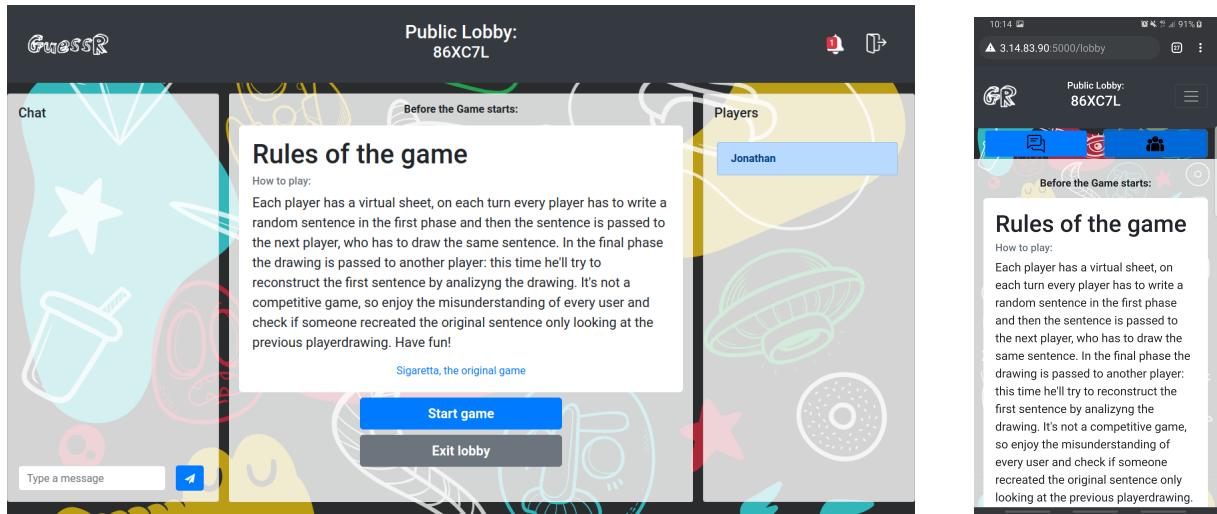


Figura 3.11: Schermate della Lobby (Desktop e Mobile)

La schermata che si visualizza una volta creata una lobby è denominata "insideLobby" e mostra agli utenti una chat, le regole del gioco ed il link alla pagina wikipedia del gioco da cui è tratto GuessR (sigaretta), l'elenco dei membri della lobby ed un pulsante per uscire dalla lobby. Solo l'utente che ha creato la lobby e che quindi ne è amministratore avrà un pulsante per dare il via al vero e proprio gioco.

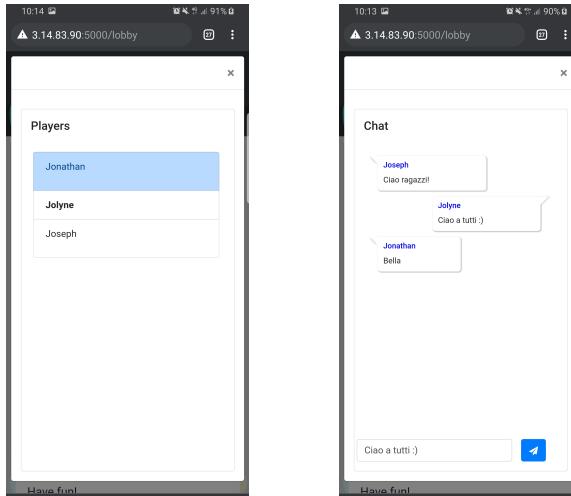


Figura 3.12: Schermate Mobile di Chat e Giocatori

Nella visualizzazione mobile i pannelli relativi alla chat ed ai giocatori sono collassati in due pulsanti presenti sotto da navbar, nella parte superiore dello schermo.

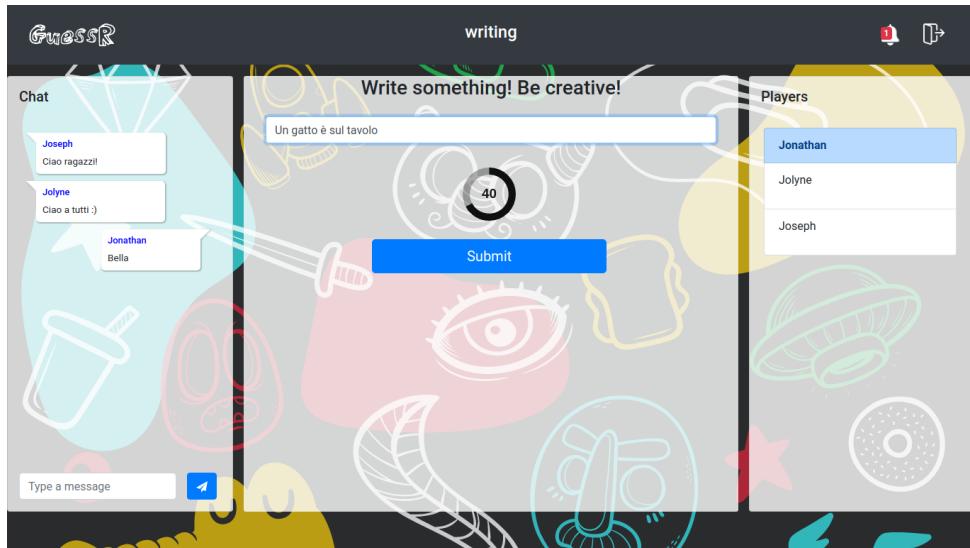


Figura 3.13: Schermata di sentence

Una volta avviato il gioco la prima fase in cui ogni giocatore si trova è quella di "sentence", che vede l'inserimento di una frase scelta dal giocatore. Ogni giocatore sia nella fase di sentence che in quella di draw avrà un timer sincronizzato ad indicargli di quanto tempo dispone prima che ciò che ha scritto/disegnato venga sottoposto automaticamente.

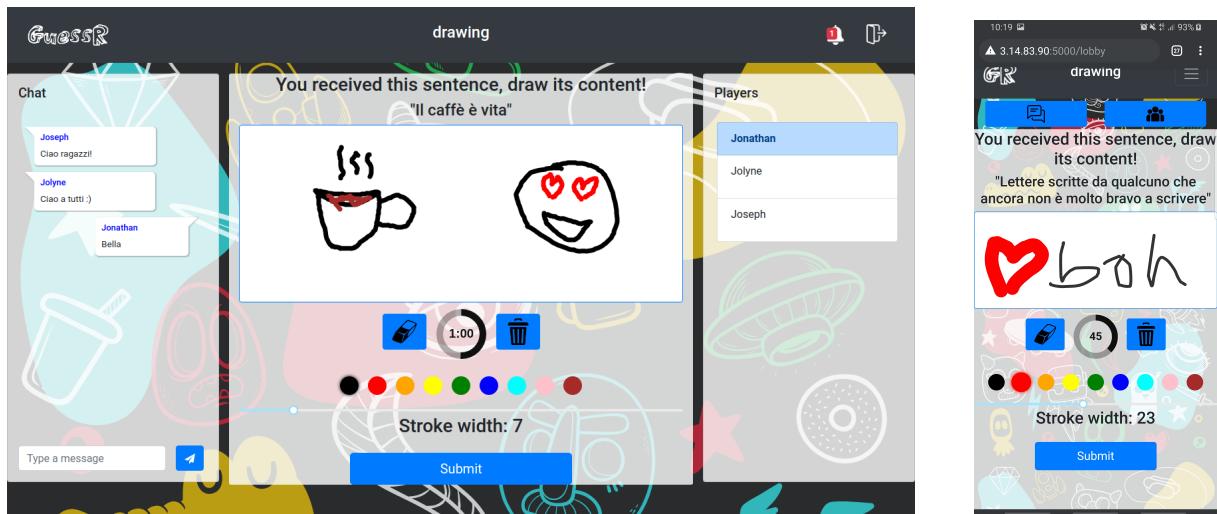


Figura 3.14: Schermate di Disegno (Desktop e Mobile)

Nella fase di "draw", a gli utenti viene predisposto un canvas centrale su cui possono disegnare la frase che gli spetta, oltre al timer questa volta gli utenti hanno una suite di strumenti da utilizzare per il disegno, come una serie di diversi colori, una gomma ed un pulsante per resettare il canvas e uno slider che gli permette di impostare lo spessore del tratto da utilizzare.



Figura 3.15: Schermata dei Report finale

Una volta terminato il gioco a seconda dei turni impostati in fase di creazione della lobby, i giocatori si ritrovano nella schermata di "show reports" in cui trovano un resoconto della partita appena svolta e l'evoluzione di ogni frase e disegno. Questa è la fase in cui i giocatori si confrontano, o in chat vocale separata o utilizzando la chat predisposta da GuessR per commentare la partita appena terminata.

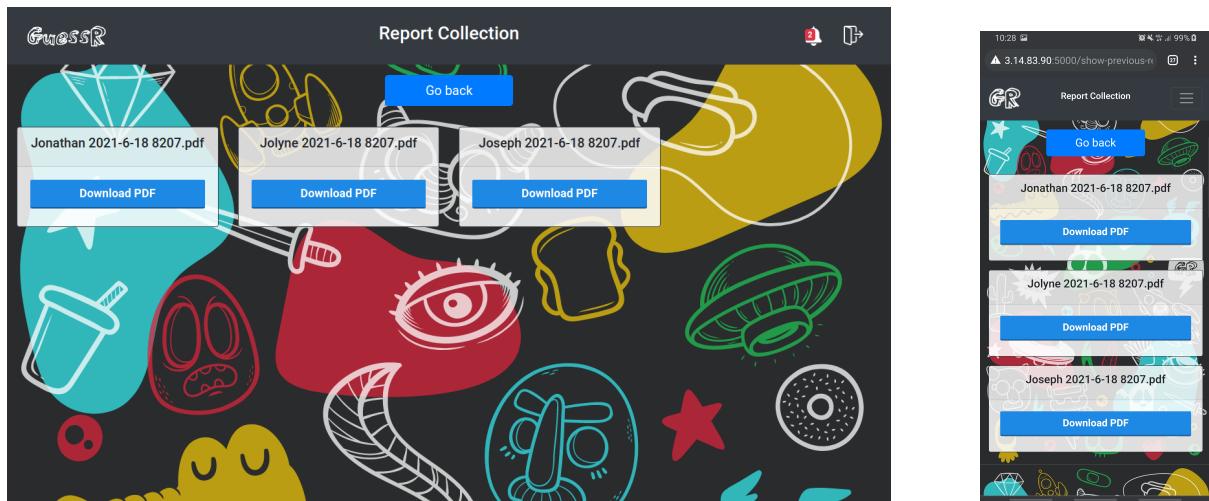


Figura 3.16: Schermata dei report precedenti (Desktop e Mobile)

Gli utenti che hanno svolto delle partite possono accedere in ogni momento ad una interfaccia apposita predisposta per consultare e scaricare i report di partite precedenti, identici a quelli visualizzati al termine della partita.

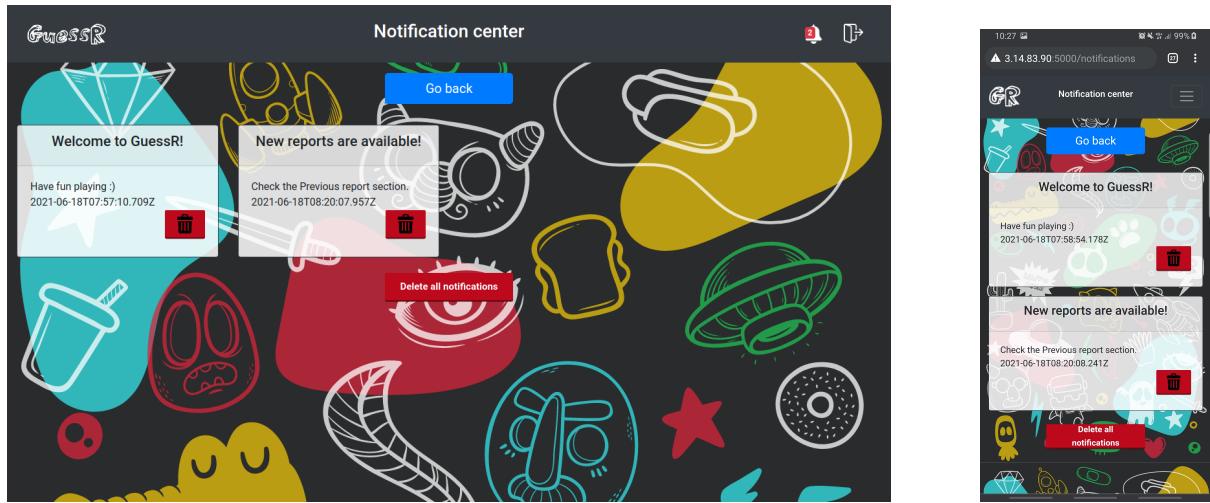


Figura 3.17: Schermate delle notifiche (Desktop e Mobile)

Gli utenti possono in ogni momento consultare la pagina relative alle notifiche, che se presenti si mostrano con un badge rosso contenente il numero di notifiche ricevute, nell'apposita icona presente sulla navbar.

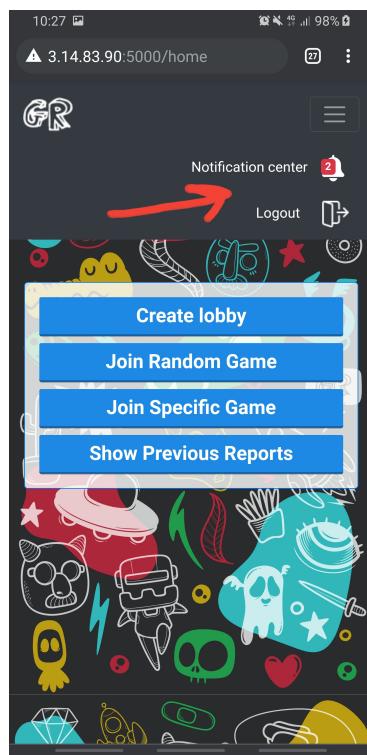


Figura 3.18: Schermata mobile contenente la navbar ed il menù a dropdown

Nella versione mobile l'icona di notifiche e di logout sono collassate in un menù a dropdown per favorire la responsiveness del sito e conferire una miglior UX.

3.4.4 Scelte estetiche e UI



Figura 3.19: Palette di colori utilizzata per i componenti del sito, il logo e lo sfondo

Lo sfondo, i colori e le trasparenze sono stati scelti in modo da rendere il sito quanto più gradevole alla vista ma mantenendo sempre come primo obiettivo la leggibilità, l'accessibilità e la scalabilità del sito. Ogni schermata è stata validata da appositi strumenti allo scopo di verificare che i colori adottati dalla palette e presenti nel sito non ostacolino in alcun modo la leggibilità, nemmeno nel caso di persone ipovedenti. Lo sfondo svolge un duplice ruolo, contenendo numerosi disegni e forme funge anche da fonte di ispirazione per utenti che si ritrovano a dover disegnare qualcosa, alcuni dei disegni presenti sullo sfondo sono stati effettuati in fase di testing del gioco con veri utenti.



Figura 3.20: Logo di GuessR

Il logo nella sua versione estesa riporta il nome del gioco "GuessR" in un font che ricorda un disegno effettuato a mano libera e varia nel formato desktop quando viene effettuato un hover. Nella versione mobile riporta le lettere "GR".

Capitolo 4

Tecnologie

Le tecnologie impiegate per il progetto in questione sono state tra le prime cose dibattute e definite, previa fase di modellazione dell'applicativo. Ogni tecnologia, framework e ambiente di sviluppo è stato scelto valutando la fattibilità, la coerenza e l'impiego ottimale di risorse oltre che all'ottimizzazione delle prestazioni. In questa prima fase ci siamo soffermati affinché potessimo garantire che ogni tecnologia utilizzata potesse permetterci di costruire un solido sistema distribuito, che pertanto avesse tutte le caratteristiche fondamentali di esso. In seguito andremo ad indicare e descrivere le principali tecnologie utilizzate suddividendole tra tecnologie condivise tra client e server, server side, client side e database.

4.1 Tecnologie condivise tra client e server

4.1.1 Node Package Manager

Node.js [9] è una runtime di JavaScript multipiattaforma per l'esecuzione di codice JavaScript, costruita sul motore JavaScript V8 di Google Chrome.

Node.js dispone di una grande quantità di moduli scritti completamente in Javascript. Essendo il progetto open source è inoltre possibile per gli sviluppatori aggiungere i propri moduli in modo da renderli disponibili pubblicamente [9].

Il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js si chiama Node Package Manager (npm). npm può essere richiamato tramite linea di comando usando la seguente sintassi:

```
npm <command> [args]
```

Il comando base per ottenere un pacchetto è:

```
npm install packet_name
```

Tutte le dipendenze e i conflitti vengono gestiti automaticamente [10].

Grazie a Node è anche possibile per esempio creare un progetto React utilizzando il comando seguente:

```
npx create-react-app project
```

npx è uno strumento integrato in npm in grado di eseguire pacchetti, anche se non sono ancora installati nel sistema.

Sarà poi possibile avviare il server di sviluppo utilizzando i seguenti comandi:

```
cd project  
npm start
```

L'interfaccia sarà poi visualizzabile all'indirizzo <http://localhost:3000> [11].

Figura 4.1: Logo di npm [1]



4.1.2 Socket.io

Socket.io è una libreria JavaScript che fornisce un servizio molto simile alle originali Web-Socket. Socket.io offre delle API JavaScript cross-browser che permettono la creazione di un canale di comunicazione full-duplex tra domini a bassa latenza tra il browser e il server web.

Socket.io tenta di utilizzare prima di tutto le WebSocket native. In caso fallisca (in caso di incompatibilità coi sistemi o a causa di altri problemi) ricorrerà all'utilizzo di polling HTTP.

Socket.io è stato progettato per funzionare con tutti i browser moderni (97% di compatibilità nel 2020) e in ambienti che non supportano il protocollo WebSocket, ad esempio dietro proxy aziendali restrittivi.

Nonostante il client di GuessR utilizzi a sua volta un ambiente javascript, Socket.io offre un supporto per la realizzazione di client in altri linguaggi quali Java, C++, Python, ecc...

Grazie a tale supporto, sarà possibile implementare molteplici client che possano in modo agile interfacciarsi a tale mezzo di comunicazione.

Oltre alle funzionalità offerte da una tradizionale WebSocket, Socket.io offre inoltre le seguenti features:

- reliability (switch a polling HTTP nel caso la connessione WebSocket non possa essere stabilita)
- riconnessione automatica
- buffering dei pacchetti
- acknowledgments
- broadcast a tutti i client o ad un sottoinsieme di essi (Room)
- multiplexing

Le ultime due feature in particolare ben si sposano con GuessR:

- Il concetto di Room trova una sua trasposizione uno a uno con il concetto di lobby di gioco, permettendo di scambiare i messaggi agilmente all'interno di tale lobby
- Tramite multiplexing è possibile definire molteplici namespace così da poter associare handler diversi e meglio separare le tipologie di messaggi scambiati tra client e server (ad esempio creando un namespace dedicato alla chat, un namespace dedicato all'inoltro di dati, ecc...)

Figura 4.2: Logo di Socket.io



4.2 Server-side

4.2.1 Node.js

Come runtime JavaScript guidato da eventi asincroni, Node.js è progettato per creare applicazioni di rete scalabili; molte connessioni, difatti, possono essere gestite contemporaneamente. Ad ogni connessione verrà invocata una callback, rendendo Node attivo solo al momento necessario.

Nonostante in questo progetto Node sia stato utilizzato per sviluppare un server, è importante sottolineare che Node.js non è un web server, ma una piattaforma in grado di eseguire codice javascript *lato server* e, tramite apposite librerie, sviluppare per l'appunto un web server.

Node.js implementa un'architettura event-driven, facendo dunque affidamento su un event loop. Non esiste alcuna chiamata per avviare il ciclo: Node.js entra semplicemente nel ciclo degli eventi dopo aver eseguito lo script di input e, analogamente, esce dal ciclo di eventi quando non ci sono più callback da eseguire. Questo comportamento è simile a JavaScript in browser: il ciclo degli eventi è nascosto all'utente.

Per natura dell'event loop, Node è single-threaded, ma è possibile, su necessità, effettuare delle fork per sfruttare al meglio i core offerti dalla macchina creando nuovi thread.

Figura 4.3: Logo di Node.js [2]



4.2.2 Mongoose

Mongoose è una libreria per Node.js che permette di creare degli Schema per rappresentare i dati da archiviare nel sistema.

Ogni Schema è associato ad una collezione nel Database di MongoDB.

Mongoose viene utilizzato per la creazione del proprio model, è infatti possibile creare delle istanze dallo Schema attraverso delle Factory ed utilizzarli come dei semplici oggetti Javascript.

Oltre ad offrire metodi aggiuntivi già pronti per salvare i dati dentro il Database è possibile creare funzioni che sono oggetti appartenenti ad un relativo schema possono richiamare, rendendo la modellazione molto object-oriented.

La creazione di uno schema di esempio per archiviare gattini su Mongoose avviene nel seguente modo:

```
const kittySchema = new mongoose.Schema({  
  name: String  
});
```

Dopo aver creato uno Schema è necessario creare una costante che rappresenta il model e collegarlo allo Schema appena creato:

```
const Kitten = mongoose.model('Kitten', kittySchema);
```

Ora è possibile creare dei nuovi gattini utilizzando una sintassi object oriented:

```
let jojo = new Kitten({ name: 'JoJo' });  
  
console.log(jojo.name); // 'JoJo'
```

È anche possibile creare funzioni da associare al model appena creato:

```
kittySchema.methods.meow = function () {  
  const greeting = this.name  
  ? "Meow name is " + this.name  
  : "I don't have a name";  
  console.log(greeting);  
  
  jojo.meow(); // 'Meow name is JoJo'
```

Per salvare il gattino nella collezione basterà richiamare il metodo `save()`. Si consiglia di utilizzare i metodi `then()` e `catch()` per avere controllo sul risultato dell'operazione:

```

jojo.save() //save jojo in the collection
  .then() //do something when operation is completed
  .catch() //do something if an error occurred

```

Figura 4.4: Logo di Mongoose [3]



4.3 Client-side

Per il lato client è stato scelto uno stack tecnologico che ci permettesse il più possibile il riutilizzo di codice e l'automazione.

È stato dunque scelto l'utilizzo di javascript con l'appoggio di node.js grazie ai quali è stato possibile realizzare agilmente sia la versione web del client che quella mobile nativa.

4.3.1 React.js

React è un framework open-source che permette di implementare applicazioni web seguendo i principi della programmazione ad oggetti. In modo particolare risulta essenziale comprendere tre concetti chiave:

- **JSX**

JSX [12] è un'estensione della sintassi di JavaScript.

Permette di unire gli aspetti di html come linguaggio di template agli aspetti di JavaScript come linguaggio di scripting in una forma che ne aumenta semplicità e leggibilità.

Gli elementi JSX vengono utilizzati nelle definizioni delle funzioni di rendering (che verranno trattate a breve) semplificando la costruzione della user interface.

Attraverso JSX possiamo richiamare un componente React tramite con un meccanismo analogo ai tag in html.

- **Componenti**

Attraverso un componente [13] andiamo a definire quella che risulta essere a tutti gli effetti una classe.

Il nostro componente, definito da uno o più costruttori, contiene uno stato; questo stato verrà mantenuto, ed eventualmente aggiornato, durante tutto il ciclo di vita (lifecycle) del componente.

È possibile ricevere dati ed istruzioni da altri componenti attraverso le props.

- **Stato**

Lo stato [14] un insieme di proprietà di un componente.

Queste proprietà possono variare a seguito dell'interazione con altri componenti o come azione del componente stesso nel caso esso esegua delle azioni a cadenza temporale.

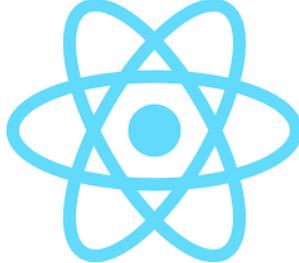
In React inoltre lo stato risulta fondamentale ai fini di ottenere un rendering a schermo performante; ogni componente React infatti deve obbligatoriamente definire una funzione `render()`.

Attraverso di essa verrà ritornato il contenuto da renderizzare a schermo.

React cambierà il contenuto a schermo (consumando quindi risorse) solamente quando vi saranno delle modifiche nel contenuto ritornato dalla funzione `render()`; utilizzando dunque le proprietà che definiscono lo stato del componente all'interno della funzione `render()` del componente stesso riusciremo a ridurre al minimo indispensabile il numero di volte in cui l'applicazione verrà renderizzata, riflettendo i cambiamenti di stato del componente.

React introduce anche i componenti funzione, che di fatto svolgono la stessa funzione dei componenti precedentemente spiegati (componenti classe), ma con una sintassi più concisa.

Figura 4.5: Logo di React.js [4]



4.3.2 Redux

Redux [15] è un *contenitore di stato* per applicazioni JavaScript.

Godet di quattro caratteristiche fondamentali per progetti portata medio-grande:

- **Deterministico**

Un aspetto fondamentale nelle applicazioni web di ogni genere è il determinismo.

L'oneroso compito di far collaborare tutti i componenti al fine di ottenere un comportamento predecibile viene largamente semplificato dal framework Redux.

- **Centralizzato**

Avere stato e logica centralizzati permette di ottenere rapidamente delle feature di fondamentale importanza, come funzioni di "annulla" e "ripeti" che ci permettono di muoverci agilmente tra lo storico degli stati.

Inoltre un approccio centralizzato garantisce un consistenza maggiore dei dati, rendendo possibili modificarli solamente attraverso specifiche funzioni (azioni) create durante la definizione della struttura dati.

- **Debug oriented**

Attraverso semplici plugin browser come Redux DevTools risulta immediato il debug dell'applicazione.

Difatti attraverso questi tool otteniamo una visione completa dello stato dei dati e della sua evoluzione nel tempo, riuscendo ad identificare con precisione quando e soprattutto perché lo stato abbia subito delle modifiche.

- **Flessibile**

Redux funziona con ogni layer di User Interface e, essendo ormai uno strumento consolidato, dispone di un solido supporto e un vasto parco di plugin e pacchetti aggiuntivi per ogni esigenza.

Per comprendere come queste caratteristiche si concretizzino risulta fondamentale comprendere tre concetti chiave nella struttura di Redux:

- **Store**

Lo store [16] è un oggetto che contiene l'intera struttura ad albero dello stato.

Fornisce metodi per la lettura dello stato corrente.

- **Reducer**

I reducer [17] definiscono la struttura dello store.

Vi possono essere più reducer all'interno di una stessa applicazione, al fine di meglio suddividere lo stato.

- **Azioni**

Le azioni [18] permettono di modificare il contenuto dello store.

Essendo queste l'unico modo per alterare lo stato attuale, qualsiasi componente che voglia agire sullo stato deve passare per le azioni definite.

Ma perché utilizzare un contenitore di stato se ogni componente React possiede già uno stato? React, pur essendo uno strumento molto potente, ci pone davanti a delle limitazioni:

- Innanzitutto non è raro che più componenti debbano fare riferimento allo stesso dato; la presenza di uno stato comune evita di dover implementare meccanismi ad hoc di sincronia tra gli stati dei due componenti.
- Lo stato di ogni componente, attraverso tool ready to use, sarà dunque sincronizzato con la struttura principale.
- In secondo luogo React, data la sua natura orientata al reactive programming, incoraggia la propagazione dell'informazione solo in una direzione: da un componente padre verso un componente figlio (utilizzato dunque dal padre). La presenza delle azioni Redux risolvono questo problema in quanto ogni cambiamento apportato allo stato Redux si rifletterà su tutti i componenti React che utilizzano quel particolare dato.

Figura 4.6: Logo di Redux [5]



4.3.3 Bootstrap

Bootstrap è stato creato nel 2010, tramite Twitter, da @mdo and @fat.

È rapidamente diventato il framework più popolare per il front-end a livello mondiale. Con le prime release della versione 2 vennero introdotti i primi fogli di stile (opzionali) per aggiungere funzionalità riguardanti il design responsive.

Con l'uscita della versione 3 l'intera libreria è stata ripensata e riscritta per offrire supporto al design responsive con un approccio mobile-first. La versione 4 di Bootstrap infine (quella attuale) ha subito due principali cambiamenti nella sua architettura:

- una migrazione a Sass, un'estensione del linguaggio CSS che permette l'uso di variabili e di funzioni. L'estensione dei file Sass è .scss;
- l'uso di CSS flexbox, ovvero contenitori per elementi.

Bootstrap adotta un approccio mobile-first. L'assunzione di base è che la maggior parte dei visitatori utilizzeranno uno smartphone a causa del sempre più crescente utilizzo di questo tipo di dispositivi.

L'utilizzo continuo di media query è alla base di ogni layout creato utilizzando Bootstrap [?].

Moltissime classi offerte utilizzano i cosiddetti breakpoints. Si tratta di meccanismi che permettono di nascondere, mostrare o modificare il comportamento degli elementi a cui vengono applicate in base alla risoluzione dello schermo del visitatore. Bootstrap mette a disposizione i seguenti breakpoints:

```
// Small devices (landscape phones, 576px and up) KEYWORD:sm
@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up) KEYWORD:md
@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up) KEYWORD:lg
@media (min-width: 992px) { ... }

// Extra large devices (large desktops, 1200px and up) KEYWORD:xl
@media (min-width: 1200px) { ... }
```

Segue un rapido esempio per l'uso dei breakpoints. Per la gestione dei margini e del padding Bootstrap mette a disposizione le classi di tipo {property}{sides}-{breakpoint}-{size}.

- **{property}**: **m** se si vuole aggiungere un margine, **p** se si vuole aggiungere del padding;
- **{sides}**: **t** per top, **b** per bottom, **r** per right, **l** per left, **x** per right e left, **y** per top e bottom;
- **{breakpoints}**: se omesso, la classe funziona su qualsiasi tipo di schermo, anche i più piccoli, altrimenti funziona solo sugli schermi più grandi anche solo di un pixel del breakpoint specificato;
- **{size}**: Bootstrap non utilizza valori statici per specificare di quanto deve essere il padding o il margin. Il valore che può assumere size va da 0 a 5, e sono valori che indicano quando deve essere grande il moltiplicatore che gestisce lo spazio. 0 significa nessun padding/margin.

Se si desidera applicare un piccolo margine a destra ad un determinato elemento solo se il visitatore lo visualizza su uno schermo con larghezza maggiore di 992px, le classi che l'elemento dovrà assumere saranno **mr-0 mr-lg-2**.

mr-0 significa nessun margine a destra su qualsiasi tipo di schermo. **mr-lg-2** invece vuol dire margine a destra con proporzione 2 solo su schermi più grandi di 992px. Combinando insieme le due classi otteniamo un margine solo su schermi con larghezza maggiore di 992px [?].



Figura 4.7: Logo di Bootstrap [6].

4.4 Database-side

4.4.1 mongoDB

MongoDB è un DBMS NoSQL, cioè non utilizza un meccanismo di persistenza relazionale come un tradizionale SQL.

Il modello NoSQL non è unico e può dunque utilizzare varie strutture dati per sostituire le tabelle con campi uniformi usate in SQL.

In particolare mongoDB utilizza un modello orientato al documento, dove le informazioni sono memorizzate in una struttura gerarchica ad albero ed un qualsiasi numero di campi con qualsiasi lunghezza può essere aggiunto. I campi a loro volta possono anche contenere pezzi multipli di dati.

I DBMS orientati al documento offrono alcuni vantaggi, specialmente in ambito web, rispetto ai tradizionali RDBMS:

- Maggiore flessibilità dei dati, utile per avere meno rigidità in fase di sviluppo o, in generale, per scenari in cui i dati memorizzati non sono sempre uniformi
- Facilità nella trasposizione in strutture dati javascript in quanto i JSON (i documenti utilizzati internamente in mongoDB e, ormai, standard de facto) trovano una corrispondenza uno a uno con esse.

D'altro canto è pur vero che una struttura meno rigida aumenta il rischio di duplicazione di dati ed inconsistenze; è dunque richiesta al progettista una maggiore cautela nella manipolazione di dati.



Figura 4.8: Logo di mongoDB [7].

Capitolo 5

Codice

5.1 Server

5.1.1 Routes

La definizione delle Route è situata nei file della cartella *routes*.

Il router del server crea la route attraverso un metodo che ha lo stesso nome del metodo HTTP usato per accedere alla route e vi associa l'handler che deve gestire le request.

È possibile associare anche dei middleware alle route, ovvero una funzione che viene eseguita prima dell'effettivo handler, solitamente per effettuare dei controlli.

Nello snippet che segue il middleware auth non permette lo scaricamento di un report, a meno che l'utente non sia autenticato.

Tutti i componenti che si occupano di gestire la logica delle route si trovano nella cartella *controller*.

```
router.post("/auth/signup", (req, res) => AuthController.signup(req, res));

router.post("/auth/login", (req, res) => AuthController.login(req, res));
//notice the auth middleware
router.get("/dw/report", auth, (req, res) => ResourceController.downloadReport(req, res));
```

5.1.2 Model su Mongoose

La definizione degli Schema del DataBase sono state strutturate utilizzando Mongoose. Lo Schema degli User per esempio è definito nel seguente modo:

```
let UserSchema = new Schema({
  username: {
    type: String,
    require: true
  },
  password: {
    type: String,
    require: true
  },
  notifications: [
    type: Schema.Types.ObjectId,
    ref: "Notification"
  ],
  user_in_games: [
    type: Schema.Types.ObjectId,
    ref: "UserInGame"
  ]
});
```

Anche tutti gli altri Schema sono stati creati utilizzando la stessa sintassi. È possibile salvare un nuovo utente creandolo attraverso il costruttore e poi richiamando il metodo `save()`.

```
let newUser = new User({username: username, password: pw});
newUser.save().then(
    //handler if operation succeeds
).catch(
    //handler if operation fails
);
```

Per effettuare operazioni direttamente sulla collezione è possibile utilizzare direttamente i metodi offerti dallo schema:

```
User.findOne({ username: "Jotaro" }).then(result => /*...*/)
```

5.1.3 Socket Handlers

Una volta che una connessione tramite socket viene instaurata col client è possibile ricevere ed inviare messaggi attraverso un canale full duplex. Ogni messaggio viene mandato e ricevuto su uno specifico channel, in modo che un handler possa leggerlo, gestirlo, e con ogni probabilità aggiornare lo stato interno del sistema. Gli handler dei canali delle socket si trovano nella cartella `socket/handlers`, mentre il controller che li utilizza è il file `socket/controller.js`.

```
io.on('connection', (socket) => {

    socket.on(Channels.CREATE_LOBBY, (json) => createLobbyHandler(socket, json));

    socket.on(Channels.SENTENCE, (json) => sentenceHandler(socket, json));

    socket.on(Channels.DRAW, (json) => drawHandler(socket, json));

    //...

    socket.on(Channels.DISCONNECT, () => disconnectHandler(socket));
});
```

5.2 Client

5.2.1 Requests

La gestione delle request sul client è incapsulata nei file js il cui nome finisce con `"Logic"`, i quali mettono a disposizione funzioni che effettuano richieste e aggiornano lo stato interno dell'applicazione.

Per effettuare le richieste è stata utilizzata la libreria JQuery. Una richiesta ha il seguente formato:

```
$.ajax({
    url: SERVER_ADDRESS + "/route",
    type: 'POST', //can be for example GET, or any other HTTP method
    data: {...} //data to send, not mandatory
    headers: {"Authorization": token} //used in protected routes
}).done(function (result) {
    // do something with the result... probably updating redux status
});
```

L'header `"Authorization"` è necessario solo per le route che richiedono che l'utente sia autenticato. In questo caso il valore è il Bearer token affidato all'utente, incapsulato nello stato di redux.

5.2.2 Socket Handlers

Una volta che una connessione tramite socket viene instaurata col server è possibile ricevere ed inviare messaggi attraverso un canale full duplex. Ogni messaggio viene mandato e ricevuto su uno specifico channel, in modo che un handler possa leggerlo, gestirlo, e con ogni probabilità aggiornare lo stato interno del sistema. Gli handler dei canali delle socket si trovano nella cartella *socket/handlers*

```
io.on('connection', (socket) => {

    socket.on(Channels.CREATE_LOBBY, (json) => createLobbyHandler(socket, json));

    socket.on(Channels.SENTENCE, (json) => sentenceHandler(socket, json));

    socket.on(Channels.DRAW, (json) => drawHandler(socket, json));

    [...]

    socket.on(Channels.DISCONNECT, () => disconnectHandler(socket));
});
```

Capitolo 6

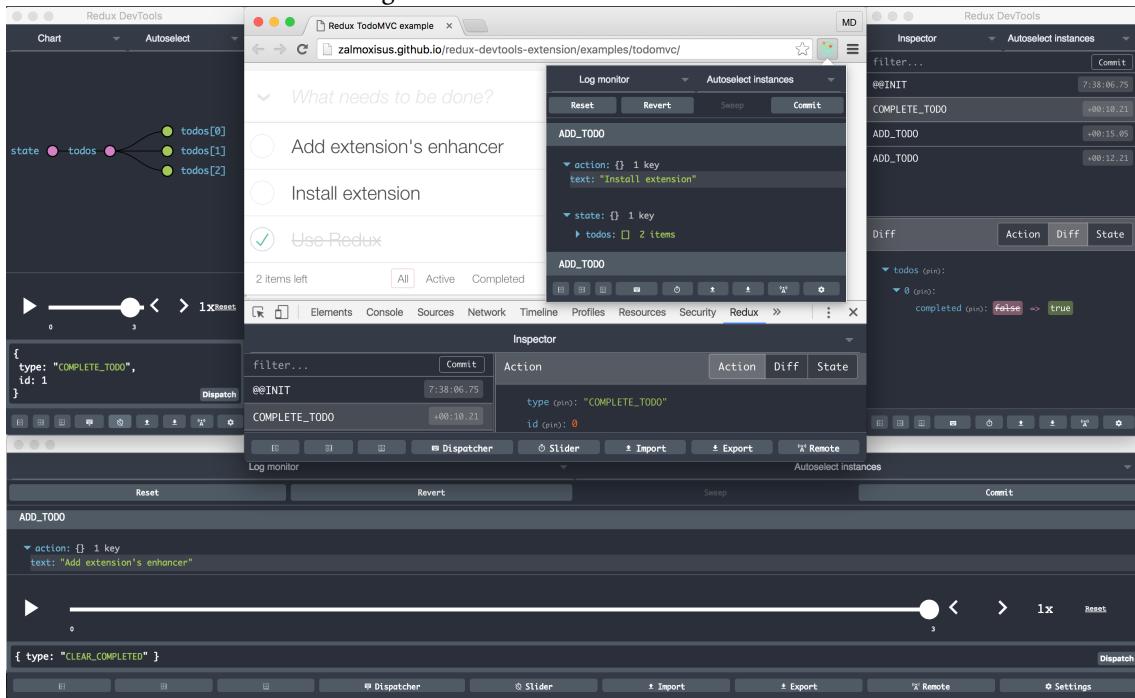
Test

6.1 Redux DevTools

I browser più famosi mettono a disposizione un'estensione per il controllo degli stati di Redux in tempo reale. Redux DevTools è l'estensione utilizzata per il debug e i test degli stati.

In realtà è possibile utilizzare Redux DevTools con qualsiasi altra architettura che utilizza gli stati.

Figura 6.1: Finestra di Redux Devtool



Redux DevTools è stato utilizzato soprattutto per testare il corretto funzionamento di comunicazione tramite WebSocket.

Figura 6.2: Logo di Redux DevTools



6.2 Axe DevTools

Il set di tool di cui ci siamo serviti per monitorare ed eventualmente correggere questioni riguardanti l'accessibilità nel presente applicativo. Axe DevTools è un insieme di strumenti leader del settore, che permette tramite un'estensione installabile direttamente su browser di ispezionare una pagina web o sottoparti di essa allo scopo di individuare problemi di accessibilità. Il tool poi procede con il suddividere i problemi in varie categorie a seconda della gravità dei problemi eventualmente riscontrati:

- **critical**: il set di errori più gravi che potrebbero compromettere la navigabilità del sito;
- **serious**: set di errori che non compromettono la navigabilità ma possono ostacolare o peggiorare l'esperienza dell'utente all'interno di un sito;
- **moderate**: errori che principalmente riguardano il mancato utilizzo di pratiche ritenute opportune, tuttavia non fondamentali per garantire l'accessibilità;
- **minor**: dove sono evidenziate sviste ed errori che impattano in maniera inferiore sull'esperienza utente.

Oltre a questa comoda suddivisione i tool permettono anche di raggruppare più errore dello stesso tipo che avrebbero soluzione comune, solitamente predispongono una descrizione del problema identificato e delle linee guida per risolverlo.

Figura 6.3: Logo di Axe DevTools



6.3 Postman

L'app Postman è un comodo strumento per testare una API REST in API Gateway. Postman è stato utilizzato per effettuare testing sulle API del server.

Grazie a Postman è possibile creare richieste verso una certa API e verificare il contenuto e il formato delle risposte.

Ogni API del sistema è stata testata, creando richieste opportune per verificare che tutto andasse a buon fine, e richieste sbagliate o non valide per verificare che il sistema rispondesse in maniera comunque consona e giusta.

Figura 6.4: Schermata della Workspace Postman di GuessR

The screenshot shows the Postman application interface. At the top, there's a navigation bar with File, Edit, View, Help, Home, Workspaces, Reports, Explore, and a search bar. Below the navigation is a toolbar with various icons. The main area is titled 'Team Workspace' and contains a sidebar with collections like 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'History'. A 'guessr' collection is selected, showing several API endpoints: POST /auth/signup, POST /auth/login, GET /profile, GET /languages, GET /dw/report:id, GET /notification, POST /notification, and DEL /notification:id. The 'POST /auth/signup' endpoint is highlighted. The request details panel shows a POST method to 'localhost:3000/auth/signup', headers (Content-Type: application/json), and a body with the JSON payload: { "username": "Jotaro", "password": "OraOraOra70" }. The response panel is currently empty.

Per chiamare un'API utilizzando Postman è necessario:

- Immettere l'URL dell'endpoint di una richiesta nella barra degli indirizzi (nel nostro caso il server) e selezionare il metodo HTTP appropriato nell'elenco a discesa a sinistra della barra degli indirizzi.
- Se necessario, scegliere la scheda Authorization (Autorizzazione). Per le route protette dal middleware di autenticazione è necessario specificare un JWT valido per poter effettuare la richiesta. Qualora questo non venisse specificato la risposta sarà 401 (Unauthorized).
- Selezionare la scheda Headers (Intestazioni). Facoltativamente, eliminare le intestazioni esistenti. Questa operazione può cancellare eventuali impostazioni non aggiornate che potrebbero essere causa di errori. Aggiungere le intestazioni personalizzate (se richieste).
- Scegliere Send (Invia) per inviare la richiesta e ricevere una risposta.

Figura 6.5: Logo di Postman



Capitolo 7

Deployment

In questo capitolo verrà illustrato come poter effettuare il deploy dell'applicativo.

Oltre alla versione in locale, utilizzata in fase di sviluppo e testing, è stato possibile di rendere disponibile una versione online funzionante, così da testarne in maniera più accurata alcuni aspetti e ottenere feedback da utenti reali facendoli interagire con il sistema.

Per tale scopo è stato utilizzato il servizio gratuito di Amazon AWS per ottenere una macchina virtuale sul cui poter effettuare il deploy.

Verranno esposte due modalità per effettuare il deployment: una tradizionale ed una con il supporto di Docker.

7.1 Docker-free deployment

7.1.1 Prerequisiti, installazione di Software

Per poter effettuare il deployment risulta necessario installare i seguenti software permettere al sistema un corretto funzionamento:

- **git**: necessario per reperire il repository del progetto. È possibile reperire i sorgenti in qualsiasi altro modo, ma utilizzare git permette ovviamente più flessibilità.
- **npm**: necessario per installare le dipendenze necessarie e per il corretto funzionamento del componente client.
- **node**: necessario per il corretto funzionamento del componente server.
- **mongodb**: necessario per avere a disposizione il database.

7.1.2 Deployment Server

Il deployment del server è molto più semplice della controparte client. È sufficiente collocarsi nella cartella del server, installare le dipendenze utilizzando:

```
npm install
```

ed avviare il server utilizzando il comando

```
node app.js
```



ATTENZIONE: nel caso si scelga di effettuare il deployment docker-free sarà necessario adeguare la stringa di connessione al db, reperibile presso asw-project/server/conf/conf.js

7.1.3 Deployment Client

Per il deployment del client sarà necessario spostarsi all'interno della cartella "client" e lanciare il comando

```
npm install
```

che provvederà all'import di tutte le librerie npm utilizzate dal client.

È poi necessario eseguire il comando:

```
npm run build
```

che si occuperà di creare i file di build all'interno della cartella *build*[19].

Infine basterà eseguire il comando:

```
serve -s build
```

7.2 Dockerized deployment

7.2.1 Docker

Docker è un sistema open-source tramite il quale è possibile automatizzare il processo di deployment di applicazioni all'interno di contenitori software.

Docker implementa API di alto livello per gestire *container* che eseguono processi in ambienti isolati.

Utilizzando i container dunque le risorse possono essere isolate, i servizi limitati e i processi avviati in modo da avere una prospettiva completamente privata del sistema operativo, col loro proprio identificativo, file system e interfaccia di rete. Più container condividono lo stesso kernel, ma ciascuno di essi può essere costretto a utilizzare una certa quantità di risorse, come la CPU, la memoria e l'I/O.

L'utilizzo di Docker per creare e gestire i container può semplificare la creazione di sistemi distribuiti, permettendo a diverse applicazioni o processi di lavorare in modo autonomo sulla stessa macchina fisica o su diverse macchine virtuali. Ciò consente di effettuare il deployment di nuovi nodi solo quando necessario.

Al fine di creare un container Docker sarà necessario specificare un **Dockerfile** per ogni servizio erogato. Similmente ai gitignore per il versioning git, è possibile definire dei **dockerignore** per segnalare a docker quali file ignorare durante la copia dei file all'interno del container.

Figura 7.1: Logo di Docker



7.2.2 Docker compose

Il deployment di più sottosistemi può essere effettuato tramite lo strumento Compose di Docker; tramite essi verrà eseguita un'applicazione Docker multi-container.

Sarà necessario creare un file **docker-compose.yml** dove verrà fatto riferimento ai Dockerfile di ogni singolo servizio al fine di creare un macro-container che contenga al suo interno tutti i container dei servizi.

In quanto di default ogni container risulta isolato dagli altri, sarà necessario specificare l'esistenza di una sotto-rete interna all'esecuzione del macro-container del Docker compose. Gli indirizzi da specificare all'interno degli applicativi dovranno tenere conto dei namespace della rete così definita in docker-compose.yml.

Per fare in modo che i servizi siano raggiungibili dall'esterno del container, sarà fondamentale inoltre mappare le porte interne al container con le porte della macchina sulla quale il container è in esecuzione.

7.2.3 Deployment

Una volta installato Docker sul sistema, sarà necessario semplicemente posizionarsi nella cartella asw-project ed eseguire il comando

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

Per terminare l'esecuzione sarà semplicemente necessario usare la combinazione di tasti CTRL+C e successivamente lanciare il comando

```
docker-compose down
```

Per far sì che lo script sia lanciato in modalità detached basterà lanciare il comando

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

così che l'esecuzione del processo sia indipendente dalla bash presso la quale è stato lanciato il comando. In questo caso per terminare l'esecuzione sarà necessario solamente il comando docker-compose down.

Capitolo 8

Conclusioni

8.1 Note finali

L'elaborato finale sicuramente soddisfa tutti gli obiettivi imposti nella fase di definizione di progetto, in alcuni casi supera le aspettative, il tutto ponendo la giusta importanza alle varie componenti di gioco e rispettando la scala di priorità definita in una prima fase.

Al termine di questo progetto possiamo affermare che si è realizzata un'architettura efficiente e a nostro parere ben strutturata, con una particolare attenzione riguardo alla user experience, all'accessibilità e in generale ad un'attività ed un'esperienza di gioco inclusive e implementate a regola d'arte.

Ogni servizio svolge il suo compito correttamente garantendo il risultato atteso e si coordina con gli altri senza problemi.

Al di fuori dell'elaborato, uno dei più grandi goal che ci siamo imposti inizialmente era far sì che questo progetto ci permetesse di rimanere in contatto virtuale con amici, che non potremmo vedere a causa delle forti limitazioni alla mobilità dovute alla pandemia in atto. Anche in tal fronte ci possiamo ritenere soddisfatti e confermare di aver ritrovato lo stesso entusiasmo che abbiamo avuto noi nel crearlo durante il gioco.

8.1.1 Sviluppi futuri

Degli sviluppi abbastanza estensivi potrebbero vedere un tipo di interazione utente-utente, come scambio di amicizie o creazione di gruppi per velocizzare l'invito in lobby, oppure la personalizzazione di più parametri di gioco da parte dell'admin o di ogni utente, ad esempio la possibilità di scegliere il tempo da lasciare ad ogni utente per produrre una frase o un disegno.

Per quanto riguarda i report uno degli sviluppi più immediati potrebbe essere l'aggiunta per ogni utente di scegliere se salvare i report o meno ed eventualmente dare la possibilità di eliminare alcuni dei report salvati, qualora non siano particolarmente d'interesse per l'utente.

Ulteriori sviluppi sotto altri fronti che contribuirebbero a migliorare ulteriormente UI/UX e personalizzazione delle schermate utente potrebbero vedere l'integrazione di meccanismi di gamification (come avatar per gli utenti, badges e/o reward system), oppure l'immissione di nuove dinamiche come l'introduzione di classifiche basate sul voto degli utenti, che potrebbero giudicare le frasi e i disegni migliori.

Bibliografia

- [1] Npm Inc. npm | build amazing things. <https://www.npmjs.com/>.
- [2] Node.js. <https://nodejs.org/it/>.
- [3] Mongoose odm v5.12.8. <https://mongoosejs.com/>.
- [4] Facebook Inc. React – a javascript library for building user interfaces. <https://reactjs.org/>.
- [5] Dan Abramov. Redux - a predictable state container for javascript apps. | redux. <https://redux.js.org/>.
- [6] Bootstrap · the most popular html, css, and js library in the world. <https://getbootstrap.com/>.
- [7] Il database più diffuso per le app moderne | mongodb. <https://www.mongodb.com/it>.
- [8] Wikipedia. Sigaretta (gioco) — Wikipedia, the free encyclopedia. [http://it.wikipedia.org/w/index.php?title=Sigaretta%20\(gioco\)&oldid=104339926](http://it.wikipedia.org/w/index.php?title=Sigaretta%20(gioco)&oldid=104339926), 2020.
- [9] OpenJS Foundation. Node.js. <https://nodejs.org/en/>.
- [10] npm Inc. npm | npm docs. <https://docs.npmjs.com/cli/v6/commands/npm>.
- [11] Facebook Inc. Primi passi – react. <https://it.reactjs.org/docs/getting-started.html>.
- [12] Facebook Inc. Introduzione a jsx – react. <https://it.reactjs.org/docs/introducing-jsx.html>.
- [13] Facebook Inc. Componenti e props – react. <https://it.reactjs.org/docs/components-and-props.html>.
- [14] facebook Inc. State e lifecycle – react. <https://it.reactjs.org/docs/state-and-lifecycle.html>.
- [15] Dan Abramov and the Redux documentation authors. Redux - a predictable state container for javascript apps. | redux. <https://redux.js.org/>.
- [16] Dan Abramov and the Redux documentation authors. Redux fundamentals, part 4: Store | redux. <https://redux.js.org/tutorials/fundamentals/part-4-store>.
- [17] Dan Abramov and the Redux documentation authors. Redux fundamentals, part 3: State, actions, and reducers | redux. <https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers>.
- [18] Dan Abramov and the Redux documentation authors. Redux fundamentals, part 2: Concepts and data flow | redux. <https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow>.
- [19] Creating a production build | create react app. <https://create-react-app.dev/docs/production-build/>.