**DOCUMENTAZIONE PROGETTO LAM 2022**

Corso: Informatica per il Management

****

**Alvise Amati🡪0000947898**

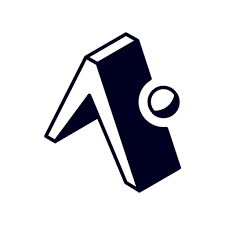
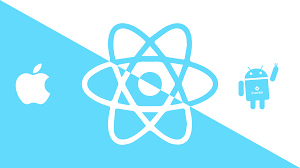
**Fabio**

**INDICE:**

* pagina 2 - 3: Introduzione e Struttura app
* pagina 4: Scelte progettuali e screen………
* pagina 5 – 9: Struttura del gioco……………..
* pagina 10: Fase di gioco e Conclusioni…….

**INTRODUZIONE A TOMBOLA**

Abbiamo scelto di proporre come app il gioco tradizionale della tombola, che come noi tutti sappiamo ci accompagna tradizionalmente nelle festività natalizie in modo fisico e noi l’abbiamo realizzato in un app multipiattaforma. Come punteggi supporta il terno, cinquina e ovviamente la Tombola.



**STRUTTURA DELL’APP**

Per la nostra applicazione abbiamo scelto React Native, per un ipotetico sviluppo reale lo avremmo scelto per i seguenti motivi:

* Quando si sviluppa un’applicazione multipiattaforma, si ha la possibilità di raggiungere un pubblico più ampio. Quando ci si rivolge a un pubblico più ampio, c’è una maggiore possibilità di successo. Lo stesso vale per la distribuzione dell’applicazione sul mercato: sarà più facile ed estesa.
* La manutenzione dell’app sviluppata necessita di un minore lavoro di manutenzione.  Differentemente dall’app nativa classica, che richiede più linguaggi e programmatori, React Native utilizza un unico codice per entrambe le App: iOS e Android. Questo significa che, nel momento in cui un cliente desideri fare qualsiasi cambiamento, aggiungere una nuova funzionalità o correggere qualsiasi bug, sarà un’operazione molto più agile e snella in quanto occorrerà apportare il cambiamento al codice per poi sincronizzarlo sulle piattaforme.
* Intuitivamente si capisce che questi vantaggi si tramutano anche, automaticamente, in risparmio economico: infatti, la riduzione del tempo equivale ad un risparmio monetario, così come l’utilizzo di minori risorse umane (gli sviluppatori). Le app native realizzate con react Native sono dunque notevoli, specie per le startup e le aziende con poco capitale iniziale, che possono creare una presenza online in modo rapido e ad un costo inferiore. React Native si adatta perfettamente a questi criteri consentendo di risparmiare tempo e denaro.

Inoltre, non avendolo mai visto avevamo un interesse personale.

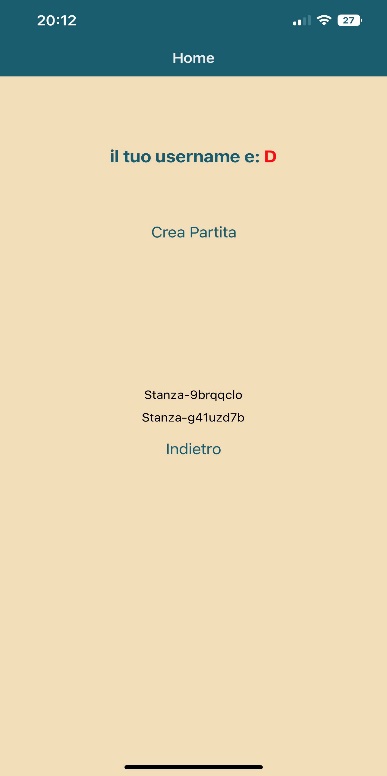
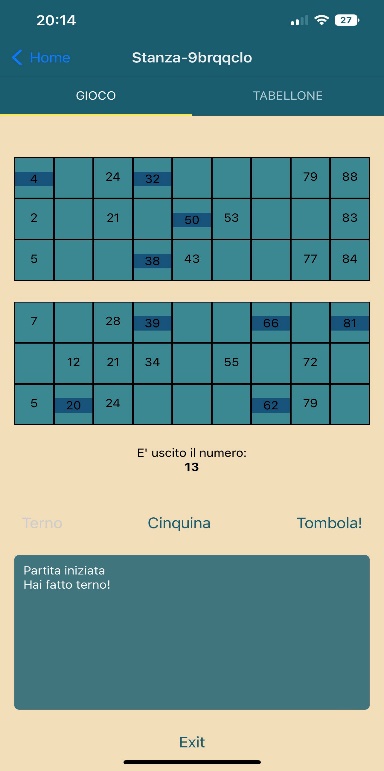
Per la build dell’app abbiamo deciso di creare l’app tramite il framework EXPO che ,grazie alla sua libreria nata in meta, ci semplifica e migliora testing/build multipiattaforma in locale. Expo Go accelera il processo di sviluppo e test poiché permette di eseguire codice JavaScript senza attendere il processo di build;

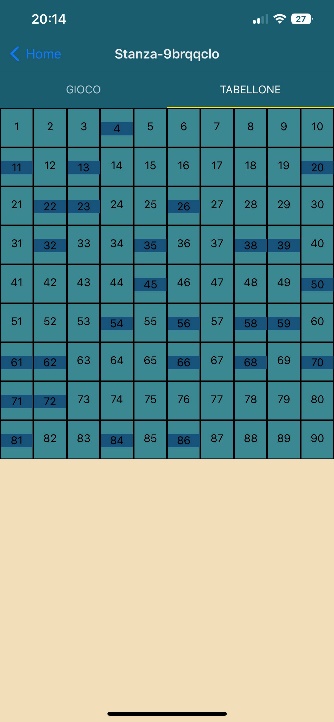
Per la struttura del Back-End abbiamo usato i socket. Tramite la libreria Socket.io abbiamo organizzato la comunicazione fra Client e Server usando appunto i socket.

Con il protocollo WebSocket, è sufficiente che il client apra la connessione a un server web. La connessione tra client e server viene stabilita con l'handshake del protocollo WebSocket. In questo caso un client invia al server tutte le identificazioni necessarie per lo scambio d'informazioni.

**SCELTE PROGETTUALI**

Abbiamo composto l’app con due componenti principali che sono Home e Partita e per navigare da uno all’altro abbiamo usato react-navigation; poi ogni componente ha e richiama al suo interno tutti i componenti renderizzando solo quelli richiesti e per gestire i render (ma non solo) abbiamo usato gli state che servono per i tipi dati che cambieranno durante il percorso (a differenza dei props) vengono inizializzati nel costruttore e invocati quando vogliamo cambiarli in sto caso per cambiare il render della view.



**STRUTTURA DEL GIOCO**

n.b. Concettualmente, i componenti sono come funzioni JavaScript: accettano in input dati arbitrari (sotto il nome di “props”) e ritornano elementi React che descrivono cosa dovrebbe apparire sullo schermo. E sono fatti per essere riutilizzati,

Il gioco è stato sviluppato nel seguente modo:

- componenteMenu , dove gestiamo la scelta del nickname e lo andiamo a salvare sia lato client sia lo comunichiamo al server tramite il corrispettivo websocket (WebSocket e un protocollo di rete e può essere inteso come un canale di comunicazione aperto, in cui una connessione attiva è aperta dopo un iniziale handshake tra client e server. Il server può anche inviare nuove informazioni al client senza previa richiesta. Possiamo inviare i dati in 2 direzioni). Successivamente cambiando l’opportuno state si renderizza la pagina dove è possibile creare una nuova partita o partecipare ad una presente già creata.

Creazione room:

Client side inviamo la richiesta di creare la room

  eseguiBottonePartita = () => {

    socket.emit("createRoom","Stanza")

  }

E server side rispondiamo generando la room (poi client stampa id)

socket.on("createRoom", (name) => { //riceve messaggio dal client

        chatRooms.unshift({ id: generateID(),iniziata: false, giocatori: [], name, numeriEstratti:[], numeriDaEstrarre: creaArrayDaEstrarre()});

        socket.emit("roomsList", chatRooms);

    });

Accesso room:

Client side mandiamo un messaggio al server con un emit e riceviamo risposta

enterRoom = async(id) => {

    socket.emit("enterRoom",{id: id, username: this.state.username})

    socket.on("roomEntered",async (idRoom) => {

      console.log("Accesso alla stanza completato")

      console.log(idRoom)

      await AsyncStorage.setItem("id",idRoom)

      this.props.navigation.navigate("Partita")

    })

Server side prepariamo e gestiamo la room e con socketIO.to comunichiamo un messaggio a tutti i client di quella room

    socket.on("enterRoom", (obj) => {

        /\*let result = chatRooms.filter((room) => room.id == obj.id);\*/

        console.log("Entro nella lobby")

        console.log(obj)

        for(var i = 0; i < chatRooms.length; i++){

            if(chatRooms[i].id == obj.id){

                console.log("Iniziata: ")

                console.log(chatRooms[i].iniziata)

                if(chatRooms[i].iniziata == false){

                    chatRooms[i].giocatori.push({

                        socket: socket.id,

                        username: obj.username

                    })

                    socket.join(chatRooms[i].id)

                    socketIO.to(chatRooms[i].id).emit("nuovoPlayerEntrato",obj.username)

                    socket.emit("roomEntered", chatRooms[i].id);

                }else{

                    socket.emit("erroreEnterRoom")

                }

            }

        }

    });

A questo punto la room e pronta e si passa al componente partita

- componentePartita , qua viene gestito tutto il gioco vero e proprio dove vengono visualizzate le due cartelle generate randomicamente per ogni client tramite la funzione generaCarta():

generaCarta = () => {

        const tools = new RandTools();

        const extract\_pool = [];

        var card = [[], [], []];

        for (var i = 0; i < 9; i++) {

            extract\_pool[i] = new RandTools();

            extract\_pool[i].distRandInit((i \* 10) + 11, (i \* 10) + 1);

        }

        for (var i = 0; i < 9; i++) {

            card[0].push(extract\_pool[i].distRandNext());

            card[1].push(extract\_pool[i].distRandNext());

            card[2].push(extract\_pool[i].distRandNext());

        }

        tools.distRandInit(9);

        for (var i = 0; i < 4; i++) card[0][tools.distRandNext()] = "";

        tools.distRandInit(9);

        for (var i = 0; i < 4; i++) card[1][tools.distRandNext()] = "";

        tools.distRandInit(9);

        var buchi = 0;

        while (buchi < 4) {

            const hit = tools.distRandNext();

            if (card[0][hit] != -1 || card[1][hit] != -1) {

                card[2][hit] = "";

                buchi++;

            }

        }

        for (var i = 0; i < card.length; i++) {

            for (var j = 0; j < card[0].length; j++) {

                card[i][j] = card[i][j].toString()

            }

        }

        return card;

    }

Questa funzione logicamente va a creare una tabella riempita tutta da numeri casuali(con il range giusto per colonna) poi successivamente va a bucare 4 celle per riga settandole vuote

Poi tramite il componente componenteTabellaSingola renderizzo la tabella passandogli le tabelle create (il .map serve per ciclarle come un foreach e prendere il singolo elemento) e in questo componente effettuamo anche questa funzione:

  \_alertIndex(indexRow,indexCell) {

        let newArr = [...this.state.tabella];

        if(newArr[indexRow][indexCell].includes("X")){

            newArr[indexRow][indexCell] = newArr[indexRow][indexCell].replace("X","")

        }else{

            newArr[indexRow][indexCell] = newArr[indexRow][indexCell] + "X";

        }

        this.setState({tabella: newArr})

    }

Che va a prendere la cella su cui abbiamo cliccato per “spuntarla” e ci va ad aggiungere una X per segnarla (con view nascosta) per poi fare i calcoli per i punteggi effettuati.

Tornando al componentePartita in alto abbiamo un bottone “Inizia Partita” che ogni client può premere se sono connessi almeno 2 persone alla stanza per iniziare la partita. Una volta premuto il server da inizio alla partita e quindi alla generazione dei numeri

Poi una volta generate le tabelle client side sotto avviene la generazione random dei numeri estratti da 1 a 90 processati lato server e comunicati a tutti i client connessi a quella stanza

function estraiNumero(numeriDaEstrarre,numeriEstratti,id){

    if(numeriDaEstrarre.length == 0){

        socketIO.to(id).emit("partitaFinita")

    }else{

        const random = Math.floor(Math.random() \* numeriDaEstrarre.length);

        var numeroEstratto = numeriDaEstrarre[random]

        numeriEstratti.push(numeroEstratto)

        socketIO.to(id).emit("numeroEstratto",numeriDaEstrarre[random])

        numeriDaEstrarre.splice(random, 1);

        setTimeout(() => {estraiNumero(numeriDaEstrarre,numeriEstratti,id)}, 4500)

    }

}

Mentre sotto abbiamo i 3 bottoni Terno, Cinquina e Tombola che al click controllano prima l’effettivo punteggio effettuato controllando le celle segnate dai giocatori nella cartella se rispettano i criteri del punteggio si controlla che quei numeri siano presenti nel tabellone dei numeri estratti e se rispetta tutto il bottone si disattiva e viene stampato chi ha fatto il punteggio e comunicato prima al server che poi lo comunica di conseguenza a tutti i client, questo è l’esempio di uno dei tre che sono pressoché simili:

  provaTerno = async() => {

        console.log('bottone terno cliccato')

        for (var i=0; i < this.state.tabelle.length; i++) {

            console.log('tabella')

            var tabella = this.state.tabelle[i]

            for (var r=0; r < tabella.length; r++) { //ciclo riga

                console.log('riga')

                var contatoreCelle = 0;

                for (var c=0; c < tabella[r].length; c++) {//ciclo singole celle

                    console.log('cella')

                    if (tabella[r][c].toString().includes("X")) {

                        var indice = this.state.listaNumeri.indexOf(parseInt(tabella[r][c].toString().split('X')[0])) //controllo se numero e uscito

                        console.log(indice)

                       if(indice != -1){

                        contatoreCelle++;

                       }

                    }

                }

                if (contatoreCelle >= 3) {

                    console.log("terno")

                    socket.emit('terno')

                    return

                }

            }

        }

    }

Sempre nel componentePartita tramite una TabBar scorrevole di react native nella tab del tabellone abbiamo una tabella generata dal componenteTabellone dove vengono segnati i numeri da 1 a 90 nello stesso modo delle cartelle e consentono al palyer di vedere i numeri usciti.

Una volta terminata la partita cioè quando tombola è stata fatta verrà renderizzata una View con un riepilogo di chi ha fatto i punteggi e un bottone per terminare la room nel server.

**FASE DI GIOCO**

Step 1: scegli il tuo nickname

Step 2: Crea una partita o entra in una esistente ed aspetta che almeno siate in 2 ad essere connessi

Step 3: Inizia a segnare nelle cartelle i numeri che escono leggendoli da sotto o nel tabellone a fianco

Step 4: controlla se effettui i punti correttamente:

* Terno: tre numeri sulla stessa riga
* Cinquina: cinque numeri sulla stessa riga
* Tombola: completa totalmente una delle due cartelle

Step 5: controlla i log in basso per vedere chi fa i punteggi e spera di essere il primo a fare Tombola!

**DETTAGLI FINALI:**

Nella prima schermata è presente un modal (un componente che in modo semplice presenta il contenuto sopra una vista che lo racchiude.) dove è possibile vedere un breve recap delle regole del gioco.

N.B per avviare l’applicazione se non sono presenti i file “node\_modules” bisogna entrare sia nella directory dell’app sia del server a fare npm install poi sempre in ogni directory con npm start lo startiamo e tramite il qr code e dalla stessa connessione accediamo dai nostri device.

- cambiare indirizzi ip in utils/socket.js & componenteMenu.js

Grazie per l’attenzione, Alvise e Fabio.