Esercitazione 5

Si realizzi una applicazione basata su React che permetta, all'addetto alle vendite di un cinema, di gestire l'emissione dei biglietti e l'assegnazione dei posti in sala agli spettatori.

L'applicazione presenta una schermata divisa in due parti: a sinistra è presente una mappa interattiva della sala (realizzata con SVG), a destra un pannello di controllo all'interno del quale viene gestito il processo di vendita.

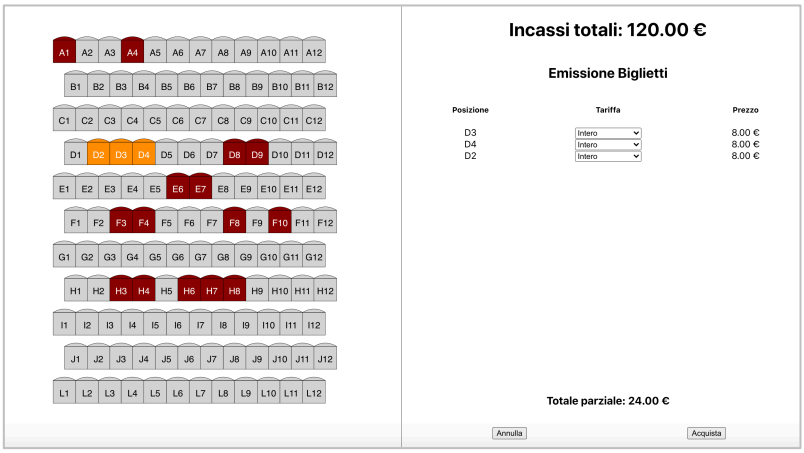
La mappa mostra la disposizione delle poltrone in sala (10 righe di 12 posti a sedere, sfalsati di mezza poltrona e numerati da "A1" a "J12"): le poltrone già vendute sono colorate in rosso, quelle libere in grigio chiaro.

Quando uno spettatore chiede di acquistare uno o più biglietti, l'operatore seleziona sulla mappa le postazioni richieste (che diventano temporaneamente arancione) e, parallelamente, vede la lista delle posizioni scelte e il totale parziale relativo alla vendita in corso nel lato destro dello schermo. Per ogni posizione selezionata, è presente un controllo a tendina che permette di selezionare una tariffa (intero: 8.00€, ridotto anziani: 6.40€, ridotto giovani: 5.60€, omaggio: 0.00€): selezionando la voce opportuna, il totale parziale si aggiornerà conseguentemente.

Se, mentre la vendita è in corso, l'operatore seleziona nuovamente una delle posizioni riservate (in arancione), questa torna grigia e viene eliminata dalla lista.

Nella schermata di destra sono presenti due pulsanti: "Annulla" e "Conferma". Premendo "Annulla", le postazioni temporaneamente selezionate diventano nuovamente grigie e la schermata di destra ritorna allo stato iniziale. Premendo invece "Conferma", le postazioni selezionate diventano rosse sulla mappa e non sono più selezionabili e la lista nel pannello di controllo si svuota.

Per comodità dell'operatore e consentire una verifica sull'incasso, nel pannello di controllo è anche indicato l'ammontare di tutte le vendite effettuate nella giornata.



Abilità da acquisire

- Gestire una rappresentazione grafica interattiva basata su SVG e React
- Gestire lo stato di un'applicazione tramite il pattern di azione e riduzione
- Propagare lo stato a tutti i componenti attraverso l'uso di un contesto

Svolgimento

1. Aprire Webstorm e selezionare File -> New -> Project -> React Project -> Create, assegnando un nome ed una cartella opportuna al progetto creato.
   Ripulire i file App.js e App.css, eliminando tutto ciò che non serve.
   Impostare la struttura del componente App affinché utilizzi l'intera finestra del browser e divida lo spazio uniformemente in due parti separate da una barra verticale.
   Verificare che la visualizzazione sia corretta e non compaiano errori sulla console.
2. Leggere attentamente il testo dell'esercizio ed individuare quali azioni l'utente finale può compiere: dovrebbero emergere 5 elementi[1] (sforzarsi di trovarli prima di andare a leggere il testo della nota).
   Per facilitare l'implementazione del pattern azione e riduzione, si crei il file "actions.js" in cui verranno definite ed esportate le costanti che indicano le azioni individuate: nel resto del progetto si farà riferimento a tali costanti piuttosto che al loro valore, così da limitare possibili errori di battitura.
   Per convenzione, le costanti di tipo azione sono scritte in lettere maiuscole.

   ```
   export const SELECT = "select"
   export const UNSELECT = "unselect"
   …
   ```

3. Si rifletta su come rappresentare lo stato dell'applicazione. Quali informazioni è necessario mantenere? Come conviene organizzarle? (smettere di leggere e provare a rispondere)
   È opportuno ragionare in termini astratti. L'applicazione deve mantenere l'elenco dei posti già venduti, con la relativa tariffa, quello dei posti che si stanno comprando (il carrello della spesa) e il totale delle vendite effettuate (ossia l'incasso).
   All'avvio, entrambi gli elenchi saranno vuoti e l'incasso sarà pari a 0.
   Via via che l'utente selezionerà dei posti sulla mappa interattiva, l'elenco contenuto nel carrello dovrà crescere, mentre, se l'utente deselezionerà un posto, tale elenco si dovrà contrarre. Nel momento in cui l'utente dovesse annullare la vendita in corso, il carrello si svuoterà mentre se l'utente confermerà l'acquisto, l'elenco contenuto nel carrello della spesa dovrà essere aggiunto a quello dei posti venduti, aggiornando parimenti l'incasso.
   In caso di selezione di una tariffa differente per un elemento contenuto nel carrello della spesa, questo dovrà aggiornarsi con le nuove informazioni.
   Da questa analisi emerge che conviene memorizzare nel carrello (e nella lista dei posti venduti) non solo il codice del posto selezionato ("C5"), ma anche il relativo prezzo, così da permettere il calcolo dell'importo dovuto. Queste informazioni dovranno essere presenti nell'oggetto azione, a corredo del verbo principale (SELECT, UNSELECT, …).
   Al fine di facilitare la costruzione degli oggetti di tipo azione da passare al riduttore affinché aggiorni lo stato a fronte di un'interazione dell'utente, nel file "actions.js" si creino ed esportino quindi tante funzioni quante sono le costanti precedentemente definite: tali funzioni (il cui nome è convenzionalmente minuscolo) dovranno restituire un oggetto dotato del campo "type", corredato delle informazioni accessorie. Nel caso, ad esempio, della selezione di un posto, avremo la funzione seguente:

   ```
   export function select(seatID, price) {
     return { type: SELECT, seatID, price }
   }
   ```

4. Si crei il file "reducer.js" all'interno del quale si definiscano ed esportino la costante initialState e la funzione reducer. Lo stato iniziale deve contenere le informazioni individuate al punto precedente. La funzione reducer(…) riceve due parametri (state e action) e deve restituire la versione aggiornata dello stato in base all'azione ed alla logica definita sopra. Questa funzione deve essere pura, ovvero la sua esecuzione non deve generare effetti collaterali di nessun tipo. In particolare occorre evitare

di modificare il dato in ingresso, inserendo o togliendo valori al suo interno. Eventuali aggiunte, cambiamenti o rimozioni devono avvenire sull'oggetto restituito che deve essere una copia di quello in ingresso. A questo scopo, è spesso conveniente servirsi del costrutto di object destructuring offerto dalla versione ES2017 del linguaggio Javascript, con frammenti del tipo

```
const outState = { ...state, key: value }
```

che creano un nuovo oggetto, con tutte le coppie chiavi valore presenti in quello in ingresso (state) a cui viene aggiunta/sostituita la coppia key:value.
Nel corpo della funzione reducer è presente un costrutto di tipo switch(…) basato sul campo type dell'azione. Tale costrutto dovrà avere tante clausole di tipo 'case' quante sono le azioni definite nel file "actions.js" ed una clausola 'default', nella quale si ritorna lo stato immutato. Per ciascun tipo di azione, si implementi la logica corrispondente.

5. Per verificare la correttezza dell'algoritmo di riduzione, si crei il file "reducer.test.js". Al suo interno verranno scritti i test, nella forma

```
test( "descrizione", () => { /* codice da eseguire */ })
```

dove la stringa "descrizione" deve essere sostituita con la spiegazione di cosa si intende verificare nel test corrente e all'interno del blocco dove è indicato il commento, viene posto il codice che si intende eseguire seguito da una serie di invocazioni del tipo

```
expect(qualcosa).toEqual(qualcosaltro) //confronto mediante ==
expect(qualcosa).toBe(qualcosaltro)    //confronto mediante ===
```

Quando verranno eseguiti i test, il sistema invocherà ciascuna funzione riportata nel file e verificherà le condizioni indicate. In caso di mancata verifica, mostrerà l'errore e il contesto in cui si è verificato (la descrizione). A titolo di esempio, per verificare il corretto funzionamento della deselezione di un elemento, si può scrivere un test simile al seguente:

```
test('verifica deselezione', () => {
    const oldState = {
        occupied: [],
        cart: [{seatID:"A1", price: 8.0}, {seatID:"A2", price: 6.4}],
        total: 0.0
    }

    const newState1 = reducer(oldState, unselect("A1"))
    expect(newState1.cart).toEqual([{seatID:"A2", price: 6.4}])
    expect(newState1.occupied).toEqual(oldState.occupied)
    expect(newState1.total).toEqual(oldState.total)

    const newState2 = reducer(oldState, unselect("A2"))
    expect(newState2.cart).toEqual([{seatID:"A1", price: 8.0}])
    expect(newState2.occupied).toEqual(oldState.occupied)
    expect(newState2.total).toEqual(oldState.total)
})
```

Per verificare la correttezza di un singolo test, è possibile fare click sul triangolo verde posto a lato della definizione di ciascuna invocazione della funzione test(…). Per eseguire tutti i test, nella finestra "Terminal" posta nella parte bassa di WebStorm, si esegua il comando npm run test

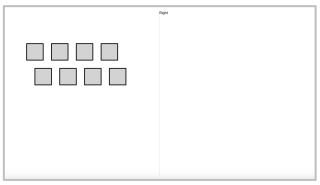
Attenzione, se nella cartella del progetto è rimasto il file App.test.js, rimuoverlo, altrimenti verranno riportati errori che non corrispondono a specifiche del progetto.

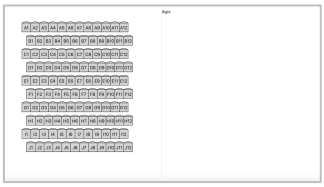
Assicurarsi che tutti i test scritti funzionino correttamente prima di procedere.

6. Si crei il file “InteractiveMap.js” all’interno del quale verrà definito il componente omonimo. Tale componente ha il compito di mostrare la mappa interattiva della sala. Questa verrà costruita dinamicamente a partire dalla lista dei posti e realizzata come elemento SVG all’interno di React. All’interno di tale file si definisca la costante width impostandola al valore 170. Questa costante avrà lo scopo di definire la larghezza della sala. Analogamente si definisca la costante height ponendo questa a 200. Si definisca inoltre la costante seats nel seguente modo:

```
const seats = [
    { seatID: "A1", x: 30, y:50, size: 20 },
    { seatID: "A2", x: 60, y:50, size: 20 },
    { seatID: "A3", x: 90, y:50, size: 20 },
    { seatID: "A4", x: 120, y:50, size: 20 },
    { seatID: "B1", x: 40, y:80, size: 20 },
    { seatID: "B2", x: 70, y:80, size: 20 },
    { seatID: "B3", x: 100, y:80, size: 20 },
    { seatID: "B4", x: 130, y:80, size: 20 },
]
```

   Tale costante ha lo scopo di elencare tutti i posti disponibili in sala con le relative coordinate.
   Si definisca ora la funzione InteractiveMap() come componente React e si restituisca un elemento SVG con larghezza e altezza impostate al 100% dello spazio disponibile e viewBox pari a 0 0 width height.
   All’interno di tale elemento si inseriscano dei rettangoli, ottenuti trasformando la lista seats in elementi di tipo <rect /> con gli opportuni attributi. Non si scordi di inserire l’attributo key con un opportuno valore. Si esporti questa funzione come valore di default del file.
   All’interno della funzione App(), si modifichi il pannello di sinistra affinché contenga un componente di tipo InteractiveMap.
   Si esegua l’applicazione e si verifichi che viene correttamente mostrata una mappa con 8 quadrati sfalsati tra loro, come mostrato nella seguente immagine a sinistra.





7. Nel file InteractiveMap.js, si definisca il componente Seat, avente il compito di visualizzare un singolo posto. Tale componente dovrà sostituire i rettangoli grigi con una rappresentazione della poltroncina che permetta di capire come è orientata verso lo schermo e riporti il codice associato. A tale scopo, il componente Seat restituirà un elemento di tipo <g/> (gruppo) che racchiude la seduta, lo schienale ed il testo con l’identificativo del posto. Si estenda inoltre la lista per considerare i vincoli della sala, adattando opportunamente i valori di width e height (figura superiore a destra).
   Il componente Seat dovrà ricevere come props il proprio identificativo (seatID), le proprie coordinate (x, y), la propria dimensione (size), l’indicazione se è occupato o meno (occupied), se è selezionato o meno (selected) e quale funzione invocare quando viene premuto (onClick).
   Se viene indicato come occupato, verrà riempito di rosso scuro, se è selezionato di arancione, altrimenti sarà grigio chiaro. Il colore del testo al suo interno dovrà adattarsi di conseguenza, per garantire il necessario contrasto. Si esegua l’applicazione e si verifichi che tutto funziona come ci si aspetta e non sono presenti errori in console.
8. Si imposti il pattern azione/riduzione. Nel file App.js si importino le funzioni createContext e useReducer dal file ‘react’ e i simboli reducer e initialState dal file “./reducer.js”.

Si definisca ed esporti la costante StateContext così definita

```
export const StateContext = createContext()
```

Tale costante definisce un contesto che sarà popolato con lo stato e con la funzione dispatch(…) necessaria ad invocare il riduttore.
All'interno del componente App, si racchiuda tutta la struttura JSX restituita con l'elemento

```
<StateContext.Provider
            value = {useReducer(reducer, initialState, A => A) }>
            …
</StateContext.Provider>
```

Tale elemento provvede a valorizzare il contesto con il risultato della funzione useReducer(…) che è formato dall'array contenente lo stato corrente e la funzione dispatch(…).
Ora, tutti i componenti React racchiusi nel contesto potranno fare accesso ad esso tramite lo Hook useContext(StateContext) e avere accesso al suo contenuto.
Nel componente InteractiveMap, si reperisca il contenuto del contesto e si utilizzino le informazioni contenute nello stato per assegnare le props occupied e selected dei posti presenti in sala. Se un dato identificativo è contenuto nell'elenco state.occupied, la prop relativa sarà posta a true. Se, viceversa, l'identificativo è contenuto nell'elenco state.cart, allora verrà impostata a true la prop selected. Si imposti la funzione onClick, da passare ai singoli componenti di tipo Seat, in modo che non faccia nulla se il posto è occupato, invochi dispatch(deselect(seatID)) se il posto è selezionato ed invochi dispatch(select(seatID, price)) altrimenti. Per avere accesso alla lista dei prezzi è opportuno estendere lo stato iniziale nel file reducer.js affinché riporti anche la lista delle tariffe vigenti (campo 'fares' dello stato, inizializzato con una lista contenente le diverse tipologie di biglietto con il relativo prezzo): poiché tale valore non è soggetto a mutazioni, se si è scritto correttamente il riduttore, non occorre fare altre modifiche.
Si esegua l'applicazione e si verifichi che ora è possibile selezionare e deselezionare i singoli posti presenti sulla mappa e che non sono presenti errori sulla console.

9. Si crei il componente ControlPanel, nell'apposito file. Tale componente dovrà fare accesso al contesto tramite lo hook useContext(…) e visualizzare il totale incassato ed il contenuto del carrello presente nello stato. Per ogni voce presente nel carrello, visualizzerà una riga contenente l'identificativo del posto, il menu drop-down con la tariffa applicata (basato su un costrutto di tipo <select value=… onChange=… >…</select>) ed il prezzo relativo. Selezionando un'altra tariffa dal menu drop-down, dovrà essere invocata la funzione dispatch(change_price(…)) con gli opportuni valori.
   Al termine della lista verrà mostrato il totale parziale (ottenuto come somma delle voci presenti nel carrello) e i due bottoni che permettono di confermare o annullare l'acquisto.
   Si esegua l'applicazione e si verifichi che ora è possibile acquistare l'insieme dei posti selezionati o annullare l'intera selezione e che non sono presenti errori sulla console.

[1] selezionare un posto, deselezionarlo, annullare tutte le selezioni, eseguire la vendita dei posti selezionati, cambiare la tariffa di un posto selezionato