Alma Mater Studiorum
Università di Bologna
Scuola di Ingegneria

Tecnologie Web T
A.A. 2021–2022

Esercitazione 6 – React.js

Home Page del corso: http://lia.disi.unibo.it/Courses/twt2122-info/
Versione elettronica: L.06.React.pdf
Versione elettronica: L.06.React-2p.pdf

Agenda

Il framework React.js

- Breve riepilogo
- Richiami di semplici esempi

Quali strumenti di sviluppo?

- Editor di testo

Esempio di applicazione multi-componente

Esercitazione

Appendice – Un ambiente di sviluppo avanzato

React.js: fatti essenziali

React.js è una libreria javascript per la **creazione di interfacce utente web**

React.js si ispira alla metodologia di sviluppo delle interfacce utenti del tipo "**Single Page Application (SPA)**"

- Si contrappone all'approccio classico in cui il browser carica nuove pagine in seguito all'interazione dell'utente

La SPA è un contenitore all'interno del quale la pagina Web *evolve dinamicamente*

Ai fini del rendering degli elementi, React manipola un **Virtual DOM** per poi trasmettere i risultati della manipolazione al DOM del browser (tramite *diffing*, vengono trasmessi solo i pezzi di Virtual DOM effettivamente manipolati)

Un primo esempio

(Scaricate il file 06a_TecWeb.zip e scompatte sul vostro file system)

- List-jsx.html

Semplice esempio di creazione di un **elemento React** e invocazione del relativo rendering

Si notino:

- l'impiego della **sintassi JSX** (Javascript XML) che permette di scrivere tag HTML all'interno di codice javascript e di piazzarli all'interno del DOM senza l'uso di metodi quali *createElement()* e/o *appendChild()*
- L'impiego del tag <*script type="text/babel"*> per attivare l'interpretazione JSX

Il motore javascript del browser non è in grado di interpretare JSX. Tale compito è affidato a Babel (che deve quindi essere incluso tra le librerie js nella sezione <head> della pagina html)

Approccio a componenti

Lo sviluppo di una pagina Web avviene attraverso la scrittura di cosiddetti **Componenti** che manipolano il DOM per la creazione di elementi di interfaccia utente

L'approccio a componenti adottato da React.js:

- abilita il **riuso**
- permette allo sviluppatore di costruire interfacce complesse attraverso la **composizione** di semplici "mattoncini"
- permette allo sviluppatore di concentrarsi su logica e layout dei componenti; la manipolazione del DOM è a carico di React

Componenti: funzioni e classi

Esistono due tipi di componenti:

- Componenti di tipo ***function***
- Componenti di tipo ***class***

Entrambi i tipi di componenti sono obbligati a "restituire" codice HTML attraverso la keyword *return*

Entrambi i tipi di componenti supportano le cosiddette proprietà immutabili (***props***), usate tipicamente per la configurazione iniziale

I componenti di tipo *class* hanno caratteristiche aggiuntive (il cosiddetto ***state***) rispetto ai componenti di tipo *function* (che vengono anche detti componenti state-less). Lo *state*, a differenza delle props, nasce per rappresentare particolari proprietà delle classi che nel tempo cambieranno (ad es., in seguito ad eventi)

Esempi di componenti

Esempi di semplici definizioni di componenti:

- Function.html -> esempio di definizione di function
- Class.html -> esempio di definizione di class

Gli stessi esempi con, in più, l'impiego delle props:

- Function-props.html -> esempio di function con *props*
- Class-props.html -> esempio di class con *props*

Esempio di impiego dello state in una classe

- Class-state.html

In questo ultimo esempio, si notino:

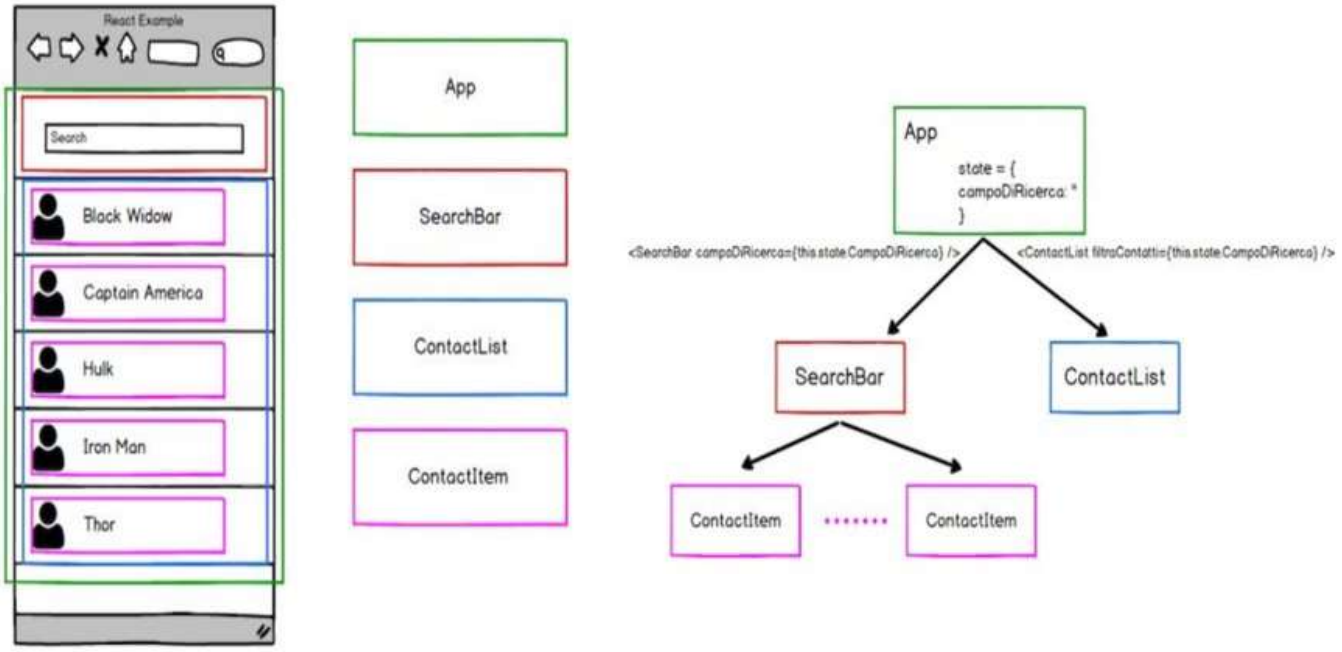
- l'uso obbligatorio del costruttore con l'invocazione *super()* -> serve per abilitare l'accesso a state
- ovunque nel codice, l'accesso a state avviene tramite la keyword ***this***

Uso raccomandato di state e props

Non tutti i componenti dovranno avere uno state

- al contrario è consigliato costruire componenti senza stato (state-less)

La **tipica applicazione React** è come una **gerarchia di componenti**: di solito, ci sono alcuni componenti ai vertici che saranno responsabili di mantenere lo stato della applicazione e di passare le informazioni giù ai componenti figli tramite props



La gestione degli eventi

Similmente a come avviene in javascript, una volta individuato l'elemento che scatenerà l'evento, occorre definire il tipo di evento che si vuole gestire e l'handler che lo gestirà

```
.......
handleClick(e) {
        console.log(`Pulsante premuto - Evento ${e.type}`);
    }
    render() {
      return (
        <button onClick={this.handleClick} >Pulsante</button>
      )
.........
```

E se occorresse accedere allo *state*?

Nel caso di componenti di tipo classe, se l'handler dell'evento deve fare accesso allo *state* del componente (cosa molto probabile) occorre apportare accorgimenti al codice di gestione dell'evento

Ci sono due alternative:

- All'interno del costruttore, forzare il *bind* del *this* del metodo handler al *this* del componente e invocare l'handler come fosse una stringa -> Interruttore_vers1.html
- Non effettuare il bind e invocare l'handler come una *arrow function* -> Interruttore_vers2.html

Un esempio più complesso: la calcolatrice

Proviamo a realizzare in React un'applicazione Web (esclusivamente front-end) che implementi l'interfaccia e le funzionalità di una calcolatrice

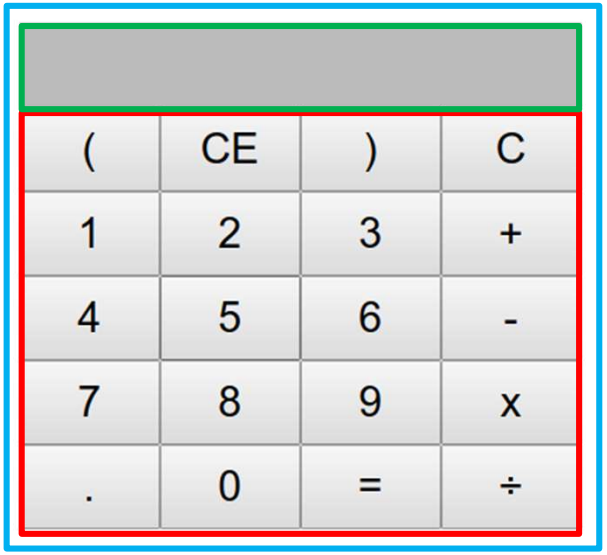
Occorrono i seguenti elementi grafici:

- Tastiera (con bottoni per numeri e operazioni)
- Campo di testo (per visualizzazione valori inseriti da tastiera e il risultato delle operazioni)

Definiamo due componenti (figli):

- Il tastierino numerico
- Il display

E infine serve un componente "contenitore" (padre) che inglobi i componenti e gestisca le interazioni utente



Calcolatrice (versione 1)

La directory Calculator_v1 contiene il file Calculator.html in cui sono definiti i componenti react

- Il componente ***Keyboard*** sarà popolato da bottoni (tasti della calcolatrice che rappresentano sia numeri che operazioni aritmetiche) a cui va associato l'handler dell'evento onClick
- Il componente ***Display*** visualizzerà sia la composizione dell'espressione aritmetica man mano che l'utente interviene sui tasti, sia il risultato finale dell'espressione (dopo che l'utente avrà cliccato sul tasto '=')
- Il componente ***App*** (contenitore) istanzia i componenti Keyboard e Display e gestisce gli eventi

Si noti che:

- *Keyboard* e *Display* non hanno state, ma solo props che vengono inizializzate da App all'atto della loro instanziazione
- *App* ha state

Riusabilità dei componenti

Abbiamo costruito una applicazione "a componenti" ….. ma è monolitica!!! Funziona, ma è poco riusabile

Se qualcuno volesse usare solo il mio componente *Keyboard*? O se volessi riutilizzarlo io in una nuova applicazione?

Bisogna "disaggregare" l'applicazione in modo tale che i singoli componenti possano avere una proprià identità (un proprio file) e poter essere così "inglobati" in eventuali altre applicazioni

Calcolatrice (versione 2)

La directory Calculator_v2 contiene:

- Il file *Calculator.html* in cui è stato creato il place-holder dell'applicazione (tag div) e in cui vengono richiamati gli script dei componenti react
- Una directory *react_components* contenente i files dei componenti React (***App.js***, ***Display.js***, ***KeyBoard.js***)
- Una directory *style* contenente un file di stile (*style.css*)

Attenzione!

- L'applicazione non funzionerà qualora si provasse ad accedere a Calculator.html direttamente da file system per via di violazioni delle policy CORS in cui si incorre invocando l'interprete Babel (verificate che sia effettivamente così)
- Occorre pertanto fare il deploy del progetto su un server Tomcat. Materialmente, occorre copiare la directory Calculator_v2 dentro la dir *webapps* di Tomcat

Esercizio #1

Provate a modificare l'applicazione in maniera tale che esponga **due display**:

- Uno in cui venga visualizzata l'espressione aritmetica man mano che l'utente la compone
- Uno in cui venga visualizzato il risultato quando l'utente preme il tasto '='

Inoltre, alla pressione del tasto 'C', occorre che vengano resettati i contenuti di entrambi i display

Ovviamente, riutilizziamo i componenti già creati!

Esercizio #2

Aggiungere funzionalità di calcolo "scientifiche" all'attuale calcolatrice con supporto di logica di programmazione lato server

Occorre:

- Creare un nuovo tastierino numerico con le seguenti operazioni: $\log_e(x)$, sqrt(x), e^x,1/x
- Applicare i nuovi operatori all'espressione attualmente composta dall'utente
- Implementare una servlet che svolga le funzionalità di calcolo relative ai nuovi operatori (lato client, implementare in AJAX l'interazione con la servlet)
- Nel layout dell'applicazione, posizionare il nuovo tastierino al di sotto dell'attuale tastiera

N.B. Ajax e React

Viene fornito un zile zip **06b_TecWeb.zip** contenente un Progetto Eclipse già pronto con un esempio di chiamata Ajax da parte di un componente React.

A questo scopo, viene esteso l'esempio di *LancioDado.html*, spostando la generazione del numbero random lato server.

Esiste infatti una servlet di nome *GenerateRandomServlet.java* che e' incaricata di generare il numero random e di restituirlo lato client al componente React il quale, una volta ricevuta la response, e' in grado di aggiornare il proprio stato e, di conseguenza, di fare un nuovo render del componente stesso.

Si fa notare come si renda necessario spostare la callback all'interno del componente React che quindi, quando invoca la funzione esterna incaricata di effettuare la response, deve anche passargli come parametro il setState per provocare un nuovo render.

APPENDICE

Un ambiente di sviluppo avanzato

Come si sviluppano le applicazioni React?

Un modo semplice è quello di lavorare con l'editor di testo

- Si crea un file html
- Si importano le librerie React nella sezione <head>
- Si scrive il codice React nella sezione <body>

In contesti di sviluppo professionali ci sono altre esigenze:

- Scalare su molti file e componenti
- Utilizzare librerie di terze parti
- Individuare subito errori comuni
- Visualizzare in tempo reale l'effetto delle modifiche al codice javascript e CSS durante lo sviluppo
- Ottimizzare l'output per la produzione

Servono strumenti adeguati per agevolare la produzione di codice

Create React App

Per soddisfare le esigenze appena discusse, si fa ricorso alle cosiddette *Toolchains:*

- Insiemi di strumenti integrati che facilitano i compiti dello sviluppatore

Esistono numerose Toolchains pronte all'utilizzo (es., Next.js, Gatsby). Volendo, è possibile costruirsene una (serve un gran lavoro di configurazione.....).
Di seguito forniamo una descrizione sintetica della toolchain Create React App

Nelle seguenti pagine web si possono trovare approfondimenti su Create React App:

https://it.reactjs.org/docs/create-a-new-react-app.html

https://www.tutorialspoint.com/reactjs/reactjs_environment_setup.htm

Preparazione dell'ambiente di sviluppo

Istruzioni per l'uso da casa:

- Scaricare ed installare l'ambiente Node.js (https://nodejs.org/it/download/)
- Da terminale, creare una directory di lavoro e lanciare il comando "npx create-react-app my-app" (operazione lunga)
- La directory appena creata (my-app) rappresenta l'area di lavoro di un progetto React di esempio
- Rimuovere (o modificare) i files di esempio per iniziare un nuovo progetto
- Per lanciare l'applicazione React eseguire il comando "npm start" all'interno della root del progetto

Nota: NON installate l'ambiente sui PC del laboratorio

Cosa offre l'ambiente

Di base, si tratta di un ambiente di sviluppo *Node.js(*)* per lavorare con applicazioni javascript (sia front-end che back-end) con, in più, le librerie React ed alcune librerie aggiuntive per React:

- ***Webpack*** -> finita l'implementazione della vostra applicazione, serve per impacchettare tutti i file js in un unico file (detto "bundle") ai fini del deploy
- ***Babel*** -> fornisce il supporto per JSX

La farraginosità del processo di creazione dell'ambiente è un prezzo da pagare per avere, alla fine, un ambiente ready-to-use (non occorre manipolare alcun file di configurazione per far parlare i vari strumenti)

(*) vedrete più avanti di che cosa si tratta

Guardiamo dentro la directory my-app

- **node_modules** (dir) -> contiene moduli Node e React
- **public** (dir) -> template html da usare ai fini del "build"
- **src** (dir)-> contiene i sorgenti js (index.js, App.js, ….)

Per lanciare l'applicazione, è sufficiente eseguire il comando:

"npm start"

Cosa è successo? È stato **avviato localmente un Web server sulla porta 3000** (lo ha fatto Node) su cui si trova l'applicazione Web. Si è poi **aperta una nuova finestra del browser** che effettua la connessione all'applicazione

Si noti che le modifiche effettuate al volo sul codice sorgente javascript sono immediatamente riscontrabili sulla pagina del browser (fate la prova)

L'applicazione my-app

Concentriamoci su due file:

- **Index.js** -> implementa il contenitore dell'applicazione
- **App.js** -> implementa il componente Ract

Si notino:

- l'uso della direttiva di importazione ad inizio file (**import**)
- l'uso della direttiva di esportazione a fine file (**export**)

L'esempio della calcolatrice

Usiamo la nostra toolchain per implementare secondo la logica a componenti riusabili la nostra calcolatrice

Estrapoliamo il codice React dei componenti Display, Keyboard, App e creiamo altrettanti file javascript (**Display.js**, **KeyBoard.js**, **App.js**)

Ricordiamoci di effettuare correttamente importazioni ed esportazioni delle risorse

- Direttiva import '…' all'inizio del file
- Direttiva 'export default …' alla fine del file

Aggiustiamo opportunamente il file **index.js**

Effettuiamo il run dell'applicazione da terminale: "npm start"

Esportiamo l'applicazione

L'applicazione funziona, ma **gira dentro il javascript engine di *Node***. È arrivato il momento di esportarla per poterne fare il deploy in un qualunque Web server

Da terminale -> "npm run build"

Viene creata una **directory build** che contiene i file dell'applicazione Web (abbastanza criptici) da poter caricare su un Web server -> sono i cosiddetti file "di produzione"

In questa operazione, vengono usate le risorse della **directory public** (icone, immagine, file index.html, etc.) come base su cui "innestare" l'applicazione React