

Alma Mater Studiorum - Università di Bologna
Scuola di Ingegneria



Tecnologie Web T
Corso di Laurea in Ingegneria Informatica
A.A. 2023-2024

Il framework React.js

Su Virtuale:

Versione 1 pagina per foglio = 3.01.React.pdf

Versione 2 pagine per foglio = 3.01.React-2p.pdf

Tecnologie Web T

1

1

Intro

- **React.js è una libreria javascript per la creazione di interfacce utente Web**
 - Rientra tra gli strumenti utili per il cosiddetto sviluppo "Front-end" di Web application
- **Sviluppata nei laboratori di Facebook nel 2011, viene impiegata per la prima volta per News Feed di Facebook e successivamente sull'applicazione Instagram nel 2012**
- **È stata rilasciata pubblicamente a maggio del 2013 con licenza MIT¹. Attualmente è mantenuta da Facebook e da comunità di aziende e singoli sviluppatori**
- **Oltre a vantare una community in costante crescita, React.js è apprezzata e utilizzata da famose aziende fra cui AirBnB, Netflix, Paypal e Uber**
- **Vocazione specifica: diventare la soluzione semplice, intuitiva e definitiva per gli sviluppatori front-end e app mobile basate su HTML5**

¹<https://opensource.org/licenses/MIT>

Tecnologie Web T

2

2

Collocazione fra gli strumenti orientati al Web

- Libreria costruita sul linguaggio javascript, pertanto qualsiasi codice scritto in React.js esegue all'interno del browser
- Ne consegue che React.js **NON è uno strumento per lo sviluppo lato back-end** delle Web application
- Come altre tecnologie front-end, React.js permette di invocare anche API lato server. React NON interagisce direttamente con database o qualsiasi altra sorgente dati che si trovi su back-end
- È in grado di interagire con tecnologie di back-end quali Python/Flask, Ruby on Rails, Java/Spring, PHP, etc.

L'approccio React in sintesi

- React.js si ispira alla metodologia di sviluppo delle interfacce utenti del tipo "Single Page Application (SPA)"
- Una SPA è un'applicazione Web che interagisce col browser per *modificare* pagine Web in modo dinamico in funzione dei dati che arrivano dal back-end
 - Si contrappone all'approccio classico in cui il browser carica nuove pagine in seguito all'interazione dell'utente
- Si dice infatti che la SPA è un contenitore all'interno del quale la pagina Web evolve dinamicamente
- Lo sviluppo di una pagina Web avviene attraverso la scrittura di cosiddetti "componenti" i quali interagiscono con le API della libreria React.js che, a loro volta, manipolano il DOM per creazione di elementi di interfaccia utente

Dietro le quinte

- React.js ha introdotto concetto di *Virtual DOM*
- Al verificarsi di un evento, invece di manipolare il DOM del browser (come fanno javascript e altre librerie), React.js manipola un virtual DOM che è una copia esatta del DOM del browser e si trova in memoria centrale
- La manipolazione del Virtual DOM è più "leggera" di quella del DOM del browser
- Lavorando con il Virtual DOM, React.js sarà in grado di inviare al DOM del browser solo le modifiche strettamente necessarie, rendendo così più leggero, efficiente e veloce il processo di rendering della pagina

Tecnologie Web T

5

5

Un esempio concreto di rendering selettivo

- Supponiamo che nel browser sia rappresentata una lista con 10 check-box, e l'utente effettui il check del primo elemento
- La maggior parte dei framework javascript, in seguito a questo evento, richiederebbero al DOM il rendering dell'intera lista! (ovvero, 10 volte il lavoro che invece servirebbe)
- In realtà, un solo elemento è cambiato mentre i restanti nove sono rimasti inalterati
- In React.js, invece, l'evento di check attiva la manipolazione del Virtual DOM (operazione molto più agile perché non deve fare rendering su schermo)
- In seguito a questa manipolazione, React confronta il virtual DOM appena manipolato con una copia dello stesso virtual DOM fatta precedentemente alla manipolazione (operazione chiamata *diffing*)
- Così facendo, individua solo gli oggetti che sono realmente cambiati: nel nostro caso, solo il primo check-box della lista. React.js comunica questa informazione al DOM del browser, il quale provvederà al rendering del solo check-box modificato

Tecnologie Web T

6

6

React.js: quali vantaggi?

- Per lo sviluppatore
 - L'approccio *a componenti*, oltre che abilitare il riuso, permette allo sviluppatore di costruire interfacce complesse attraverso la composizione di semplici "mattoncini" (appunto, i componenti)
 - Lo sviluppatore definisce la logica dei componenti e la loro collocazione all'interno dell'interfaccia utente. La gestione del Virtual DOM, delle sue trasformazioni, della comunicazione con il DOM del browser è completamente a carico di React.js
- Per l'utilizzatore dell'interfaccia
 - L'impiego del Virtual DOM alleggerisce il processo di rendering dell'interfaccia sul browser (*rendering selettivo*), con un conseguente aumento delle prestazioni percettibili dall'utilizzatore

Un primo assaggio di codice (File "01.Introduzione/Hello1.html")

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Primi passi con React</title>
    <script src="https://unpkg.com/react@15/dist/react.js"></script>
    <script src="https://unpkg.com/react-dom@15/dist/react-dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.js"></script>
  </head>
  <body>
    <!-- ... -->
  </body>
</html>
```

Creiamo una normale struttura di pagina html e importiamo le librerie React

Nella sezione <body>

```
<body>
  <div id="root"></div>

  <script type="text/babel">
    const elem = <p>Hello <strong>React</strong>!</p>;

    ReactDOM.render(elem, document.getElementById('root'));
  </script>
</body>
```

Tramite il tag div, creiamo il "contenitore" per l'interfaccia visuale

Creazione di un React Element

Cosa visualizzare

Dove visualizzare

Tecnologie Web T

9

9

React Element e sintassi JSX

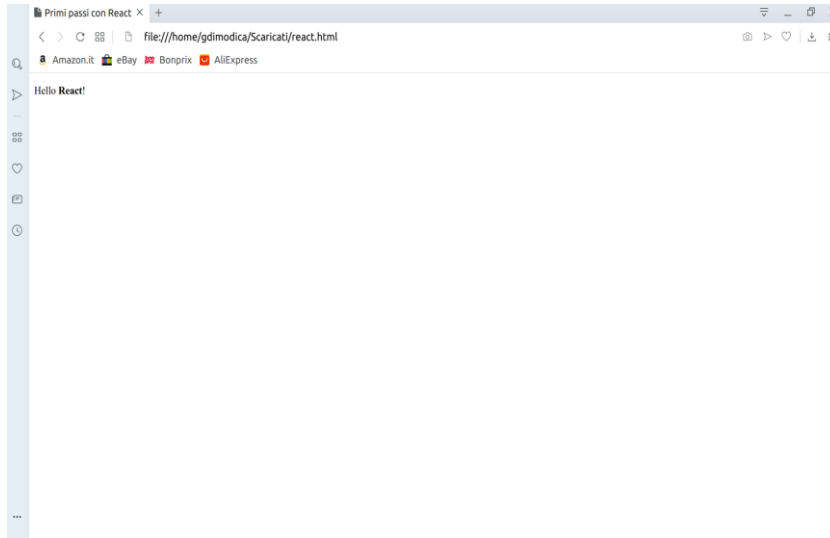
- Nell'esempio precedente, è stato definito un *React Element* e successivamente è stato chiesto al DOM del browser di visualizzare l'elemento in una specifica posizione (attraverso l'istruzione **ReactDOM.render**)
- Un React element è un oggetto semplice e immutabile che descrive cosa si vuole visualizzare sullo schermo
- Solitamente è un "nodo" html (completo di eventuali nodi figli e di attributi), ma può anche avere al suo interno istanze di *componenti* (si veda oltre per la definizione di componente)
- Attenzione a questa istruzione:
 - **const elem = <p>Hello React!</p>**
- In questo esempio, è stata utilizzata la sintassi JSX (Javascript XML) per rappresentare in modo semplice proprio l'oggetto da visualizzare

Tecnologie Web T

10

10

Il risultato finale sul browser



11

Un altro esempio (File "01.Introduzione/Hello2.html")

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Primi passi con React</title>
    <script
src="https://unpkg.com/react@15/dist/react.js"></script>
    <script src="https://unpkg.com/react-dom@15/dist/react-
dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.24/browser.js"></script>
  </head>
  <body>
    <!-- ... -->
  </body>
</html>
```

Creiamo la solita
struttura di pagina
html e importiamo
le librerie React

12

Nella sezione <body>

```
<body>
  <div id="root"></div>
  <script type="text/babel">
    class HelloWorld extends React.Component{
      render() {
        return <p>Hello <strong>React</strong>!</p>;
      }
    };
    ReactDOM.render(<HelloWorld />,
      document.getElementById('root'));
  </script>
</body>
```

Tramite il tag div, creiamo il "contenitore" per l'interfaccia visuale

Creazione di un componente

Restituzione di un React Element

Cosa visualizzare

Dove visualizzare

React Component

- Nell'esempio precedente è stato creato un *React Component*
- I React Component sono oggetti complessi e dinamici, che ricevono input dall'esterno (interazioni utente, time-out, comandi dal back-end) e forgiano l'elemento grafico da restituire
- Concettualmente, i React Component sono come funzioni JavaScript: possono accettare in input dati arbitrari (sotto il nome di "props") e restituiscono elementi React che descrivono che cosa dovrebbe apparire sullo schermo
- Nell'esempio in questione, il componente non accetta input e restituisce il seguente React Element:
 - `<p>Hello React!</p>`
- Il risultato finale su browser è identico a quello dell'esempio precedente

Linguaggio JSX

- Negli esempi è stata usata una sintassi particolare che permette di mescolare javascript e html:
 - `const elem = <p>Hello React!</p>`
 - `return <p>Hello React!</p>;`
- Questa sintassi appartiene al linguaggio JSX (JavaScript XML)
- JSX permette allo sviluppatore di scrivere facilmente tag HTML all'interno di codice JavaScript e di piazzarli all'interno del DOM senza l'uso di metodi quali *createElement()* e *appendChild()*
- Attenzione:
 - L'uso di JSX non è obbligatorio, ma di sicuro semplifica la vita dello sviluppatore
 - React mette comunque a disposizione delle funzioni per creare elementi HTML

Linguaggio JSX: un esempio

- Con l'impiego di JSX:
`const myelement = <h1>I Love JSX!</h1>;`
`ReactDOM.render(myelement, document.getElementById('root'));`
- Senza l'impiego di JSX:
`const myelement = React.createElement('h1', {}, 'I do not use JSX!');`
`ReactDOM.render(myelement, document.getElementById('root'));`
- Il risultato finale è identico

N.B.: in questo esempio, stiamo assumendo che nel file .html sia presente un contenitore (div) avente come id *root*

Creazione di una lista senza l'impiego di JSX

```
// Creazione degli elementi da inserire in una lista non ordinata
var item1 = React.DOM.li({ className: "item-1", key: "key-1"}, "Item 1");
var item2 = React.DOM.li({ className: "item-2", key: "key-2"}, "Item 2");
var item3 = React.DOM.li({ className: "item-3", key: "key-3"}, "Item 3");

// Creazione di un array degli elementi
var itemArray = [item1, item2, item3];

// Creazione della lista degli elementi
var listElement = React.DOM.ul({ className: "list-of-items" }, itemArray);

// Avvio del rendering nella pagina
ReactDOM.render(listElement, document.getElementById("container"));

N.B.: in questo esempio e in quello successivo, stiamo assumendo
che nel file .html sia presente un contenitore (div) avente come
id container
```

Creazione di una lista con l'impiego di JSX (File "01.Introduzione/List-jsx.html")

```
// Creazione della lista degli elementi -->
const listElement = <ul className="list-of-items">
  <li className="item-1" key="key-1">Item 1</li>
  <li className="item-2" key="key-2">Item 2</li>
  <li className="item-3" key="key-3">Item 3</li>
</ul>;

// Esecuzione del rendering nella pagina
ReactDOM.render(listElement,
  document.getElementById("container"));
```

Linguaggio JSX: uso di variabili, costanti e funzioni

- Nell'esempio che segue, viene dichiarata una costante chiamata *nome* che viene poi utilizzata all'interno di JSX racchiusa all'interno di { } :

```
const nome = 'Giuseppe Verdi';  
const element = <h1>Hello, {nome}</h1>;  
ReactDOM.render(element, document.getElementById('root'));
```

- Il risultato sarà la visualizzazione della scritta *Hello, Giuseppe Verdi* all'interno del contenitore denominato *root*
- All'interno di { } è possibile inserire qualsiasi tipo di espressione o funzione che restituisca un valore (es., *2+2*, *formatName(nome)*)

Chi interpreta il linguaggio JSX?

- È ovvio che il browser non è in grado di interpretare nativamente costrutti scritti in JSX, in quanto non scritti in linguaggio JavaScript
- È quindi necessario aggiungere all'interno della pagina un riferimento a un cosiddetto pre-compilatore in grado di trasformare JSX in linguaggio javascript
- Babel è un compilatore che supporta la traduzione in JavaScript di codice espresso in altri linguaggi, tra cui appunto JSX

```
<head>
```

```
....
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.js"></script>
```

```
</head>
```

I COMPONENTI

Definizione di componente

- I React Components sono i “mattoncini” fondamentali che consentono di passare da una pagina statica a un'applicazione Web dinamica la cui interfaccia è in grado di rispondere agli eventi che si verificano nella pagina, ossia reagire (react) e aggiornare se stessa di conseguenza
- Ogni "mattoncino" ha un ruolo ben definito dal punto di vista di ciò che rappresenta graficamente e si fa carico di gestire le interazioni dell'utente su quella particolare sezione di interfaccia

Esempio

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

- La sezione più esterna, quella col bordo giallo, è il componente React che rappresenta l'applicazione e contiene tutti gli altri componenti.
- A "livello" più basso, il riquadro blu contiene il pannello per la ricerca incrementale dei prodotti
- Allo stesso livello, con colore verde, c'è la lista dei prodotti che a sua volta è formata da altri componenti interni
- Per ogni riquadro di colore diverso sarà stato dichiarato un componente React

Altro esempio: la pagina di Facebook

- Nella pagina di Facebook sono presenti decine e decine di componenti (si stima che Facebook ne abbia creati complessivamente più di 15.000!)
- La sezione delle news è una di questi, che al suo interno contiene "componenti figli" per rappresentare la singola notizia
- Scendendo ancora in profondità troviamo il contenitore dei commenti, poi il singolo commento, il tasto "Mi piace" per il commento e così via, fino ad arrivare al componente che rappresenta la più piccola unità rappresentabile e gestibile individualmente nella pagina

Tipi di componenti

- In React.js i componenti sono pezzi di codice indipendenti e riusabili
- Richiamano il concetto di funzioni in javascript
- Esistono due tipi di componenti:
 - Componenti di tipo "class"
 - Componenti di tipo "function"
- Entrambi i tipi di componenti restituiscono codice HTML
 - attraverso l'istruzione *return*
- I componenti di tipo "class" hanno caratteristiche aggiuntive rispetto ai componenti di tipo "function"

Tecnologie Web T

26

26

Esempio di componente di tipo function e relativo rendering (File "02.Componenti/Function.html")

```
function Car() {  
  return <h2>I am a Car!</h2>;  
}
```

Il nome del componente va scritto qui come fosse un tag html

Il contenitore dentro cui mostrare il componente

```
ReactDOM.render(<Car />, document.getElementById("root"));
```

- Ricordarsi che il nome del componente (anche per un componente di tipo class) deve cominciare con una lettera maiuscola, altrimenti React lo tratterebbe come un normale tag HTML
- Vincolo: la funzione deve restituire l'elemento di cui fare rendering attraverso la parola chiave *return*
- ReactDOM.render è l'istruzione che attiva la manipolazione del DOM e il successivo rendering su browser. In questo caso, viene impartito al DOM il comando di renderizzare il componente "Car" all'interno del contenitore div avente come id "root"

Tecnologie Web T

27

27

Esempio di componente di tipo class e relativo rendering (File "02.Componenti/Component.html")

```
class Car extends React.Component {  
  render() {  
    return <h2>Hi, I am a Car!</h2>;  
  }  
}  
ReactDOM.render(<Car />, document.getElementById('root'));
```

- Per creare un componente di tipo *class*, occorre creare una classe che estenda da *React.Component* e implementi obbligatoriamente il metodo `render()`. Così come per i componenti di tipo *function*, occorre che questo metodo restituisca l'elemento da renderizzare attraverso la parola chiave *return*

Il concetto di *props*

- Sia per le funzioni che per le classi è possibile specificare delle proprietà (in inglese *properties*, ovvero *props*) ed assegnare a queste determinati valori
- In React, le *props* assumono valori immutabili per i quali non è prevista alcuna alterazione, utili ad esempio per configurare il componente
- L'oggetto che contiene queste proprietà prende il nome di *props* (è una parola chiave riservata)
- In fase di rendering, è possibile accedere alle *props* di un componente richiamandole come fossero attributi di un tag HTML

Uso delle *props* nelle funzioni (File "02.Componenti/Function-props.html")

L'oggetto *props* viene passato
come parametro alla funzione

```
function Car(props) {  
  
  return <h2>I am a {props.colore} Car!</h2>;  
}
```

Qui viene impiegata la proprietà di
nome *colore* (notazione puntata)

Al DOM viene passata una istanza
della funzione col parametro *colore*
impostato a *red*

```
ReactDOM.render(<Car colore="red"/>,  
  document.getElementById('root'));
```

Uso delle *props* nelle classi (File "02.Componenti/Class-props.html")

```
class Car extends React.Component {  
  render() {
```

Per le classi, l'oggetto *props* è *built-in*.
Per invocare l'uso, occorre servirsi
della parola chiave *"this"*

```
    return <h2>Hi, I am a Car. My name is {this.props.nome}</h2>;  
  }  
}
```

Al DOM viene passata un'istanza della classe
col parametro *nome* impostato a
Saetta McQueen

```
ReactDOM.render(<Car nome="Saetta McQueen"/>,  
  document.getElementById('root'));
```

Creazione della classe: metodo factory

```
var Car = React.createClass({  
  render: function() {  
    return <h2>Hi, I am a Car!</h2>;  
  }  
});
```

- `React.createClass` è l'istruzione per la creazione “al volo” del componente di tipo class
- Notare le piccole differenze di sintassi rispetto all'alternativa precedente

Il concetto di *state*

- Esiste un altro modo per rappresentare le proprietà di un componente di tipo classe
- Tutti i componenti di tipo classe possiedono un oggetto incorporato (built-in) che prende il nome di *state*
- A differenza delle *props*, le proprietà definite all'interno l'oggetto *state* NON sono immutabili. Anzi, *state* è pensato proprio per contenere proprietà che nel tempo cambieranno (in seguito al verificarsi di determinati eventi)
- Quando una delle proprietà all'interno di *state* cambia valore, viene invocata la ri-renderizzazione del relativo componente
- N.B.: le componenti di tipo function sono stateless, ovvero non hanno un oggetto state

Il costruttore e l'inizializzazione di *state*

- Analogamente a tutti i linguaggi ad oggetti, anche per il tipo di componente *class* è possibile definire un costruttore
 - Viene invocato prima del rendering e funge da iniziatore delle proprietà del componente
- Il costruttore in generale serve a:
 - Inizializzare lo stato del componente
 - Inizializzare la gestione degli eventi*

*Il concetto di "evento" verrà affrontato più avanti

Esempio di utilizzo di *state* nel costruttore

```
class Car extends React.Component {
```

```
  constructor() {
```

```
    super();
```

```
    this.state = {color: "red"};
```

```
  }
```

```
  render() {
```

```
    return <h2>I am a {this.state.color} Car!</h2>;
```

```
  }
```

```
}
```

```
ReactDOM.render(<Car />,
```

```
  document.getElementById('root'));
```

L'oggetto *state* è *built-in*: per invocarne l'uso, occorre servirsi della parola chiave "this"

L'utilizzo delle proprietà di *state* avviene dentro la funzione *render* della classe

Considerazioni sul costruttore

- Nel costruttore è stato usato il metodo *super()* per invocare il costruttore dell'oggetto padre e inizializzare correttamente il componente. Se non si invoca il metodo *super()*, non è possibile usare la keyword *this* all'interno del costruttore
- È possibile imbattersi anche in questa sintassi (passaggio di props al costruttore):

```
.....  
constructor(props) {  
    super(props);  
    this.state = {.....};  
}
```

- In seguito si faranno esempi di impiego di tale sintassi

Tecnologie Web T

36

36

Altro esempio di utilizzo dell'oggetto state (File "02.Componenti/Class-state.html")

```
class Car extends React.Component {  
    constructor() {  
        super();  
        this.state = {brand: "Ford", model: "Mustang", color: "red", year:  
1964};  
    }  
    render() {  
        return (  
            <div>  
                <h1>My {this.state.brand}</h1>  
                <p> It is a {this.state.color} {this.state.model} from {this.state.year}  
</p>  
            </div>  
        );  
    }  
}
```

```
ReactDOM.render(<Car />, document.getElementById('root'));
```

Tecnologie Web T

37

37

Componenti annidati

(File "02.Componenti/Class-nested.html")

- All'interno di un componente è possibile fare riferimento ad altri componenti:

```
class Car extends React.Component { // componente "contenuto"
  render() {
    return <h2>I am a Car!</h2>;
  }
}

class Garage extends React.Component { // componente "contenitore"
  render() {
    return (
      <div>
        <h1>Who lives in my Garage?</h1>
        <Car />
      </div>
    );
  }
}
```

È sufficiente fare il rendering del componente contenitore. React, in cascata, farà il rendering degli eventuali componenti contenuti

```
ReactDOM.render(<Garage />, document.getElementById("root"));
```

Tecnologie Web T

38

38

Aggiornamento dell'oggetto state

- L'oggetto state di un componente classe può essere modificato attraverso la funzione `setState()`, che viene definita nella classe `React.Component` e quindi viene ereditata dai componenti classe
- L'invocazione di tale funzione scatena una reazione da parte della libreria React, la quale provvederà a modificare lo stato e a re-invocare la funzione `render()` della classe in modo del tutto trasparente
- Attenzione: l'oggetto state è incapsulato all'interno di un componente, il quale è l'unico ad avere diritto e responsabilità di modificarlo
 - Nessun altro componente potrà mai invocare la funzione `setState()`

Tecnologie Web T

39

39

Esempio: lancio di un dado

- Costruiamo una semplice applicazione grafica costituita da un bottone e da un display (elemento di visualizzazione). Ogni qual volta l'utente farà pressione sul bottone, sarà estratto un numero casuale tra 1 e 6 da visualizzare
- In questo esempio, utilizzeremo l'oggetto state per definire una proprietà (numero da visualizzare) e la funzione setState() che dovrà modificare il valore della proprietà in questione, innescando così il rendering del display

Esempio: lancio di un dado

```
class Dado extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {numeroEstratto: 0};  
  }  
  
  randomNumber() {  
    return Math.round(Math.random() * 5) + 1;  
  }  
  
  lanciaDado() {  
    this.setState({numeroEstratto: this.randomNumber()});  
  }  
  .....  
}
```

Definizione e inizializzazione
della proprietà numeroEstratto

Modifica del valore
della proprietà numeroEstratto

Esempio: lancio di un dado (cont.)

```
.....
render() {
  let valore;
  if (this.state.numeroEstratto === 0) {
    valore = <small>Lancia il dado cliccando <br /> sul pulsante
    sottostante</small>;
  } else {
    valore = <span>{this.state.numeroEstratto}</span>;
  }
  return (
    <div className="card" >
      <p className="card__number">{valore}</p>
      <button className="card__button" onClick={() => this.lanciaDado()}>
        >Lancia il Dado</button>
    </div>
  )
}
```

valore è un React Element che farà da display. Inizialmente riporta del testo

Esempio completo (File "02.Componenti/LancioDado.html")

- Nell'esempio precedente è stato riportato solo il codice del componente
- Per far funzionare l'esempio, occorre aggiungere tutto l'occorrente per creare una applicazione completa
 - Tag html, head, body
 - Inclusione librerie React e JSX
 - Rendering del componente all'interno di un div contenitore

Considerazioni

- Osserviamo la riga del tag button
`<button className="card__button" onClick={() => this.lanciaDado()} >Lancia il Dado</button>`
- Sull'evento *onClick*, la funzione che deve gestire l'evento (*lanciaDado*) viene invocata in modalità *arrow*
- La giustificazione di ciò si trova più avanti nella sezione "Gestione degli eventi"

Uso raccomandato di state e props

- Non tutti i componenti dovranno avere uno state, al contrario è consigliato costruire componenti senza stato (stateless)
- La tipica applicazione React è realizzata 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

Esempio pratico

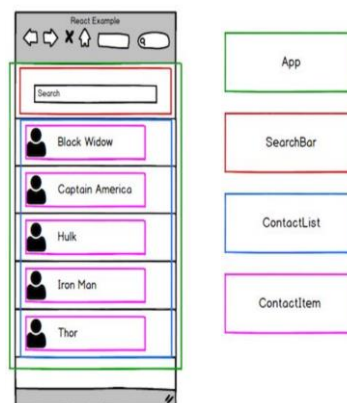
- Supponiamo di voler costruire un'applicazione di tipo Rubrica con una serie di contatti
- Secondo le specifiche ricevute, occorre cercare tra i contatti tramite un campo di ricerca
- Ogni volta che digiteremo un carattere nel campo di ricerca, verrà filtrata la lista dei contatti in base al testo digitato

Esempio pratico (cont.)

All'interno dell'applicazione, l'unica informazione dinamica è il testo che digitiamo all'interno del campo di ricerca

Creiamo due componenti **SearchBar** e **ContactList** che useranno il testo digitato nel campo di ricerca

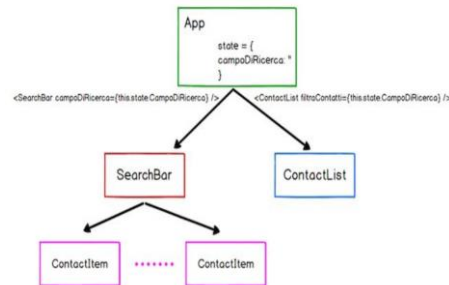
Entrambi i componenti saranno contenuti all'interno del componente più esterno **App**, che quindi è il candidato ideale a gestire il testo (ovvero, il nostro oggetto state)



Esempio pratico (cont.)

App dunque ha la responsabilità di aggiornare l'oggetto state attraverso l'invocazione della funzione `setState()`

Successivamente, passerà il nuovo valore di state ai componenti figli attraverso l'oggetto props



Quando questi riceveranno il nuovo valore, verrà invocata la loro funzione `render()` che provvederà al loro aggiornamento

GESTIONE DEGLI EVENTI

Sintassi

In html/javascript

```
<button  
  onclick="attivaLasers()">  
  Attiva Lasers  
</button>
```

In JSX

Nome
evento in
camel case

Event handler
invocato come
stringa tra
graffe

```
<button onClick={attivaLasers}>  
  Attiva Lasers  
</button>
```

- L'event handler viene solitamente realizzato attraverso un metodo della classe (in questo caso, *attivaLasers*)

Esempio

```
class App extends React.Component {  
  constructor(props) {  
    super(props);  
  }  
  handleClick() {  
    console.log("Pulsante premuto - Evento click");  
  }  
  render() {  
    return (  
      <button onClick={this.handleClick}>Pulsante</button>  
    )  
  }  
}
```

Handler dell'evento

Stampato nella console
dello strumento di
ispezione del browser

Esempio: ispezione evento

(File "03.Gestione eventi/IspezioneEvento.html")

```
class App extends React.Component {
  constructor(props) {
    super(props);
  }
  handleClick(e) {
    console.log('Pulsante premuto - Evento ${e.type}');
  }
  render() {
    return (
      <button onClick={this.handleClick} >Pulsante</button>
    )
  }
}
```

Gli eventi in React

- il parametro *e* è un evento sintetico (synthetic event)
- React definisce questi eventi sintetici in base alle specifiche W3C, quindi la compatibilità tra browser è garantita
- Gli eventi React non funzionano esattamente allo stesso modo degli eventi nativi. È possibile consultare la guida di riferimento¹ SyntheticEvent per saperne di più
- **ATTENZIONE:** Se l'handler dell'evento deve fare accesso allo *state* del componente, occorre apportare accorgimenti al codice di gestione dell'evento

[1] <https://it.reactjs.org/docs/events.html>

Accesso allo *state* da parte dei metodi di classe

- Per una corretta invocazione dell'handler dell'evento, occorre che l'oggetto invocante sia il componente. A tal fine, faremo ricorso alla keyword *this*
- Ci sono due alternative:
 - All'interno del costruttore, forzare *bind* di *this* del metodo a *this* del componente (esempio interruttore, vers. 1)
 - Invocare l'handler come una *arrow function* (esempio interruttore, vers.2)

Esempio: gestione eventi interruttore - vers. 1 (File "03.Gestione eventi/Interruttore.html")

```
class Interruttore extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {acceso: true};
```

Con questa istruzione, al 'this' dell'handler viene assegnato il 'this' del componente

```
    this.handleClick =  
      this.handleClick.bind(this);  
  }  
  handleClick() {  
    this.setState({acceso: !this.state.acceso})  
  }  
  // in alternativa  
  // this.setState(state => ({  
  //   acceso: !state.acceso  
  // }));  
}
```

```
.....  
render() {  
  return (  
    <button onClick={this.handleClick}>  
      {this.state.acceso ? 'Acceso' : 'Spento'}  
    </button>  
  );  
}
```

Event handler invocato come stringa

Esempio: gestione eventi interruttore - vers. 2

```
class Interruttore extends
  React.Component {
  constructor(props) {
    super(props);
    this.state = {acceso: true};

    handleClick() {
      this.setState({acceso:
        !this.state.acceso}
      // in alternativa
      // this.setState(state => ({
      //   acceso: !state.acceso
      // }));
    }
    .....
  }

  .....
  render() {
    return (
      <button onClick = {() =>
        this.handleClick()}>
        {this.state.acceso ? 'Acceso' :
        'Spento'}
      </button>
    );
  }
}
```

Invocazione
dell'handler
tramite
arrow
function

I FORM

Elementi di un Form e stato interno

- In React, gli elementi HTML di cui è composto un *form* funzionano in un modo leggermente differente
- La motivazione sta nel fatto che gli elementi *form* mantengono naturalmente uno stato interno
- Gli elementi di un form quali `<input>`, `<textarea>` e `<select>` mantengono e aggiornano il proprio stato in base all'input dell'utente
- In React, come è già noto, lo stato mutabile viene mantenuto nella proprietà *state* del componente e viene poi aggiornato solo mediante *setState()*

Esempi di form con `<input>` e `<select>`

- Nei due esempi che seguono creeremo form che contengono rispettivamente elementi di tipo `<input>` e di tipo `<select>`
- In entrambi i casi, l'applicazione risponderà a due eventi:
 - evento *onChange*
 - evento *onSubmit*
- A ciascun evento sarà associato uno specifico handler

Esempio form con <input>
(File "04.Form/FormInput.html")

```
class EsempioForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ""};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
    console.log('onChange: lo stato ora vale ' +
event.target.value);}

  handleSubmit(event) {
    alert('E' stato inserito un nome: ' + this.state.value);

    event.preventDefault();
  }
}
.....
```

Previene l'esecuzione del
comportamento predefinito
(es., apertura nuova pagina)

.....

```
render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Nome:
        <input type="text"
          value={this.state.value}
          onChange={this.handleChange} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

Tecnologie Web T60

Esempio form con <select>
(File "04.Form/FormSelect.html")

```
class FormGusti extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'cocco'};
    this.handleChange =
this.handleChange.bind(this);
    this.handleSubmit =
this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value:
event.target.value});
    console.log('onChange: lo stato ora vale ' +
event.target.value);
  }

  handleSubmit(event) {
    alert('Il tuo gusto preferito è: ' +
this.state.value);
    event.preventDefault();
  }
}
.....
```

```
render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Seleziona il tuo gusto preferito:
        <select value={this.state.value}
          onChange={this.handleChange}>
          <option
            value="pompelmo">Pompelmo</option>
          <option value="limone">Limone</option>
          <option value="cocco">Cocco</option>
          <option value="mango">Mango</option>
        </select>
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

Tecnologie Web T61

Gestione eventi: invocazione risorsa su server

- In React si possono effettuare HTTP request in diversi modi
- Uno elegante (e dal semplice utilizzo) fa uso delle *Fetch API* fornite nativamente da javascript
- Le Fetch API forniscono un'interfaccia js per accedere e manipolare parti della pipeline HTTP, come ad es. richieste e risposte
- Mettono a disposizione, inoltre, un metodo che fornisce un modo semplice e logico per recuperare le risorse in modo asincrono
- Nella pagina seguente viene mostrata la composizione di una request HTTP di tipo POST all'interno di un event handler. Si noti che "FormData" è un'interfaccia javascript nativa supportata da tutti i browser

Tecnologie Web T

62

62

Esempio di invio request HTTP di tipo POST

```
class MyForm extends
  React.Component {
  constructor() {
    super();
    this.handleSubmit =
      this.handleSubmit.bind(this);
  }
  handleSubmit(event) {
    event.preventDefault();
    const data = new
      FormData(event.target);
    fetch('/api/form-submit-url', {
      method: 'POST',
      body: data,
    });
  }
  .....

  .....
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label htmlFor="username">Enter
          username</label>
        <input id="username" name="username"
          type="text" />

        <label htmlFor="email">Enter your
          email</label>
        <input id="email" name="email"
          type="email" />

        <label htmlFor="birthdate">Enter your birth
          date</label>
        <input id="birthdate" name="birthdate"
          type="text" />

        <button>Send data!</button>
      </form>
    );
  }
}
```

Tecnologie Web T

63

63

LIBRERIE E FRAMEWORK ALTERNATIVI A REACT.JS

Librerie e framework javascript

- Oltre a React.js esistono numerose iniziative che propongono librerie e framework basate su javascript
- L'obiettivo di ciascuna iniziativa è quello di fornire allo sviluppatore uno strumento/ambiente di sviluppo lato front-end più "comodo" rispetto a javascript (soprattutto per la gestione del DOM) e che possa abbattere i tempi di sviluppo delle interfacce delle applicazioni Web
- Di seguito proponiamo alcune tra librerie/framework più popolari e più utilizzati, elencandone le caratteristiche principali

JQuery

- La libreria opensource jQuery è in assoluto la più utilizzata e conosciuta dalla comunità degli sviluppatori
- jQuery semplifica molto la gestione degli elementi DOM e presenta diverse funzioni per questo scopo: con i selettori del CSS3 si possono selezionare facilmente e manipolare gli elementi della pagina. Inoltre, offre una gestione semplificata delle richieste Ajax
- Il codice è compatibile con tutti i browser ed esistono molti plug-in
- È una componente essenziale di molti CMS come WordPress, Drupal o Joomla!
- La sua estensione jQuery UI è particolarmente adatta per realizzare effetti semplici ed elementi interattivi come drag&drop, ingrandimento e ridimensionamento degli elementi del sito, animazioni ed effetti vari

Tecnologie Web T

66

66

Angular (noto anche come Angular 2)

- Creato e mantenuto da Google, è il successore di AngularJS. Insieme a React.js, dispone di una grande community di sviluppatori
- È riconosciuto come l'antagonista principale di React.js. Analogamente a React.js, serve per realizzare Single Page Application
- Implementa il design pattern MVVM (Model View ViewModel). Si basa su jQuery Lite, una variante compatta della altrettanto famosa libreria js jQuery
- Rispetto al suo antecedente (AngularJS) la differenza principale è che per la programmazione non viene più utilizzato JavaScript, ma TypeScript, un linguaggio di programmazione sviluppato da Microsoft che si basa su javascript
- Punto di forza è la facilità di sviluppo delle applicazioni per diversi dispositivi (desktop, mobile, tablet)

Tecnologie Web T

67

67

Vue.js

- Analogamente ad Angular e React, Vue.js è un framework js per lo sviluppo di Single Page Application
- Adotta il design pattern Model–View–ViewModel
- L'intento degli sviluppatori di Vue.js è stato quello di creare uno strumento più facile per i principianti rispetto agli altri framework
- Ciò, però, va a discapito della completezza di funzionalità (in cui i competitor eccellono), per le quali però è comunque possibile integrare un numero ristretto di librerie aggiuntive opzionali

Meteor

- Meteor, chiamato a volte MeteorJS, è un framework javascript particolarmente adatto per lo sviluppo su diverse piattaforme
- Consente agli sviluppatori di creare con lo stesso codice sia applicazioni Web sia app per i dispositivi mobili
- Un altro vantaggio consiste nel fatto che le modifiche al codice possono essere inoltrate direttamente ai client grazie al protocollo proprietario Distributed Data Protocol (DDP)
- Questo framework js funziona su una base Node.js (ne parleremo presto 😊), pertanto può essere impiegato sia per sviluppo front-end che per sviluppo back-end
- Risulta molto utile disporre di conoscenze su Node.js per lavorare con Meteor

Backbones

- Backbones non è un vero e proprio framework ma, piuttosto, un ottimo strumento per modellare e strutturare il codice
- Grazie a questa caratteristica, backbones lascia più spazio al programmatore. Per contro, impiegato da solo non fornisce un framework completo, quindi lo si deve abbinare obbligatoriamente ad altre librerie quali underscore.js e jquery
- È nato per sviluppare applicazioni single-page ed adotta il design pattern Model-View-Presenter (MVP)