Framework di sviluppo basati su componenti – React.js

Angelo Di Iorio

Università di Bologna

React.js

- Home: https://reactjs.org/
- Nasce nel 2011 all'interno di Facebook. Nel 2013 diventa un progetto open source
- Focus sulla costruzione dell'interfaccia, ottimo compromesso tra complessità e rapidità di sviluppo. Ma ha anche le sue debolezze…
- Nel 2015 nasce *React Native*, un ambiente di progettazione integrato che permette di creare interfacce anche per sistemi *mobile*
- Nel 2022 viene rilasciato React 18, una major release che integra il motore di React Native e introduce nuovi metodi (e altri diventano deprecati)
- Nel 2023 viene lanciato *react.dev* (https://react.dev/) che contiene sia la documentazione aggiornata all'ultima versione che le precedenti (https://legacy.reactjs.org/)
- Queste slide fanno riferimento alla versione React 17

React CLI

- Anche React offre strumenti a supporto dello sviluppo e del deploy da linea di comando. Molto utili per la **compilazione**, non obbligatoria ma fortemente consigliata in React
- Installati come pacchetti NodeJS, come per Vue e Angular:

```
npm install -g react
npm install -g create-react-app
```

```
npm init react-app helloworld   //crea un'applicazione nella
                                // directory helloworld
cd helloworld

npm run start                   // lancia il server di test. Le
                                // modifiche ai sorgenti
                                // sono immediatamenti visibili
                                // nell'interfaccia

npm run build                   // compila i file per il server
                                // in produzione
```

Hello World

- Una volta inclusa la libreria, l'oggetto globale `ReactDOM` gestisce il DOM
- In particolare ha un metodo `render()` che permette di specificare:
  - il frammento HTML o JSX da includere nel DOM
  - l'elemento nel DOM dove includere il codice HTML ottenuto

```
<div id="welcome">
</div>
```

```
ReactDOM.render(
    <h1>Benvenuti in React!</h1>,
    document.getElementById('welcome')
);
```

Benvenuti in React!

React e JSX

- Nell'esempio precedente abbiamo usato una speciale sintassi chiamata JSX che mescola codice Javascript, frammenti HTML ed elementi XML *custom*
- React si può usare anche senza JSX ma è la sintassi è più verbosa
- D'altra parte la sintassi JSX mescola elementi dichiarativi e procedurali e questa resta una debolezza in ottica di leggibilità, comprensione e riuso
- Un **template JSX** non è altro che una **funzione** che **restituisce** un **frammento HTML o JSX**

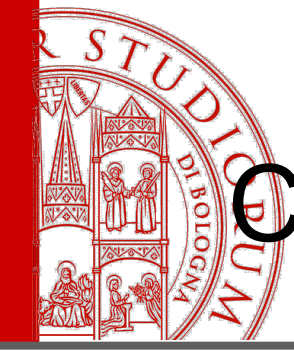
```
var Benvenuti = function() {return <h1>Benvenuti</h1>};

ReactDOM.render(
    <Benvenuti />,
    document.getElementById('welcome')
);
```

JSX e Babel

- Per poter usare JSX è necessario un interprete, in grado di trasformare il codice JSX in Javascript ed elaborare il DOM
- Babel (https://babeljs.io/) si occupa di questa traduzione, oltre ad altre operazioni per garantire retrocompatibilità. Esamina sistematicamente e ricorsivamente il codice JSX:
  - Ogni espressione JavaScript viene valutata ed eseguita.
  - Per ogni elemento per cui esiste una funzione con lo stesso nome, viene eseguita la funzione e l'output sostituisce l'elemento.
  - Per ogni elemento per cui esiste una classe con lo stesso nome, viene eseguita la funzione `render()` e l'output sostituisce l'elemento.
- React usa quindi Babel, o meglio un plugin di Babel, nella fase di compilazione e deploy

Senza JSX

```
<div id="welcome"></div>
<script type="text/javascript">

      class Benvenuti extends React.Component {
        render() {
          return React.createElement('h1', null, 'Hello
                                React (with no JSX)');
        }
    }

      ReactDOM.render(
        React.createElement(Benvenuti, null, null),
        document.getElementById('welcome')
      );
</script>
```



Con JSX (e Babel nel browser)

```
<script
src="https://unpkg.com/@babel/standalone/babel.min.js"></scr
ipt>
…
<div id="welcome"></div>

<script type="text/babel">

      var Benvenuti = function() {
        return <h1>Hello Babel!</h1>;
      }

      ReactDOM.render(
        <Benvenuti/>,
        document.getElementById('welcome')
      );
</script>
```

Template e parametri

- Un template JSX, essendo una funzione, può prendere in input parametri che il motore React userà per il rendering
- **props** è un oggetto le cui proprietà potranno essere specificate come attributi dell'elemento JSX usato per includere il template
- Nota: a differenza di Vue non è necessario dichiarare le *props* che saranno usate ma è sufficiente passare il parametro alla funzione/template

```
var Benvenuto = function(props) {
  return <p>Ciao <b>{props.nome}</b>, benvenuto in
              React.js!</p>
};

ReactDOM.render(
       <Benvenuto nome="Angelo"/>,
    document.getElementById('welcome')
);
```

Attenzione:
una sola parentesi!

Condizioni e cicli (1)

- A differenza di Vue e Angular, React non usa direttive per esprimere condizioni e cicli nei template ma i costrutti di Javascript (e JSX)

```
function(props) {
    …
    if (props.language == "EN") {
      msg = <Welcome />;
    } else {
      msg = <Benvenuto />;
    }

    return (
      <div>{msg}</div>
    );
  }
```

Condizioni e cicli (2)

- Nel caso di cicli sui vettori si crea un vettore di elementi JSX o HTML a partire dai dati di input
- Si può usare un ciclo `for` o il metodo `map(…)` che prende in input una funzione di trasformazione da applicare ad ogni elemento del vettore

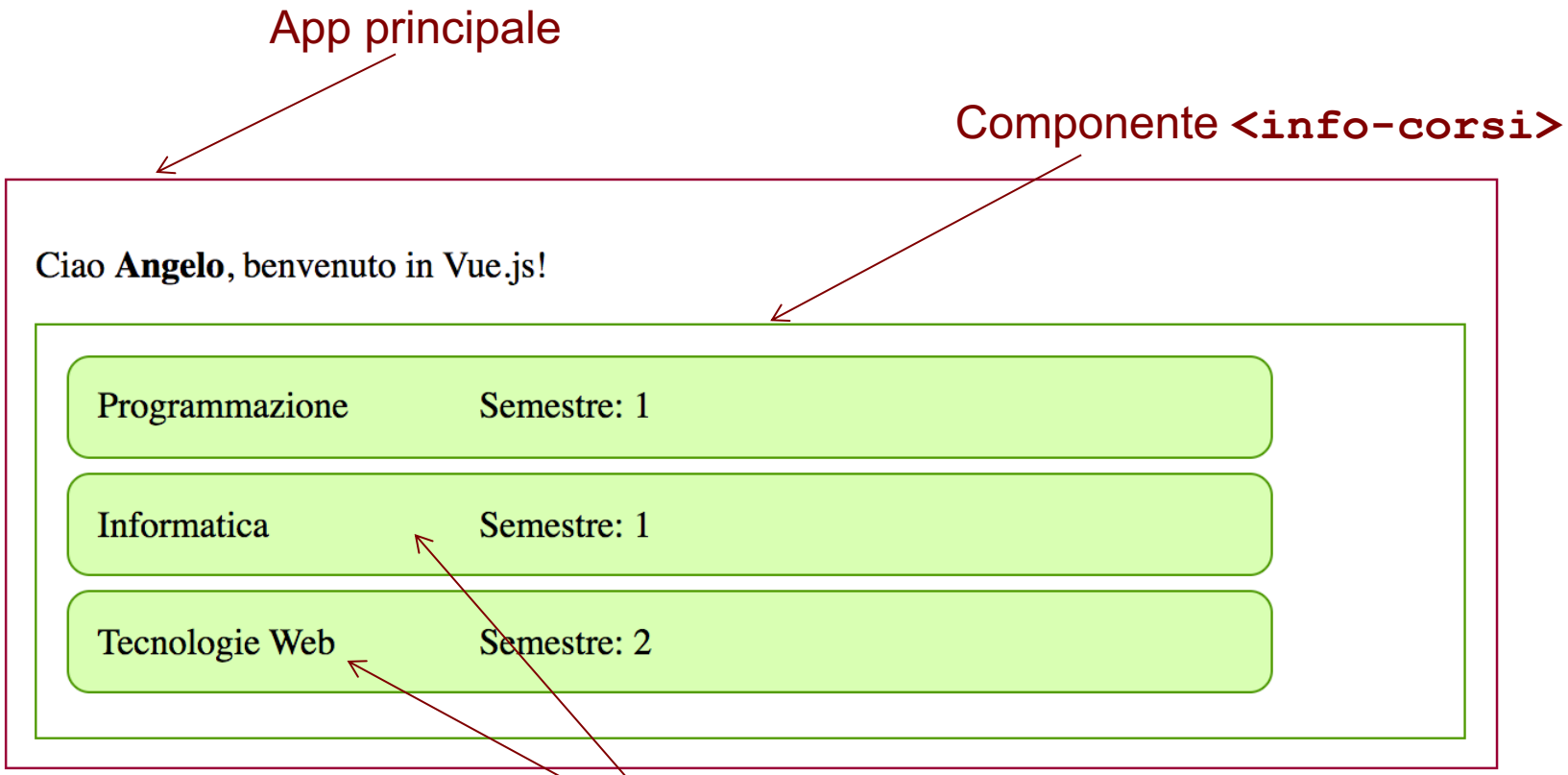
```
…
const infoCorsi = props.corsi.map(
                (corso) =>
            <div className="corso" key={corso.nome}>
              <div>{corso.nome}</div>
              <div>Semestre: {corso.semestre}</div>
            </div>
)

return <div>{infoCorsi}</div>
```

Componenti

- I template React visti finora, costruiti come funzioni che restituiscono un frammento JSX, sono in realtà Componenti
- Possono essere create anche usando la sintassi di ES2015
- **Si estende la classe `React.Component` (da importare) e si implementa in metodo `render()`**
- `this.props` contiene le proprietà del componente, ossia i dati passati in input dal componente che l'ha creato

```
class Benvenuto extends React.Component {


 render(){
    return <p>Ciao <b>{this.props.nome}</b>, benvenuto
in React.js!</p>
  }
}
```

Il nostro esempio in React

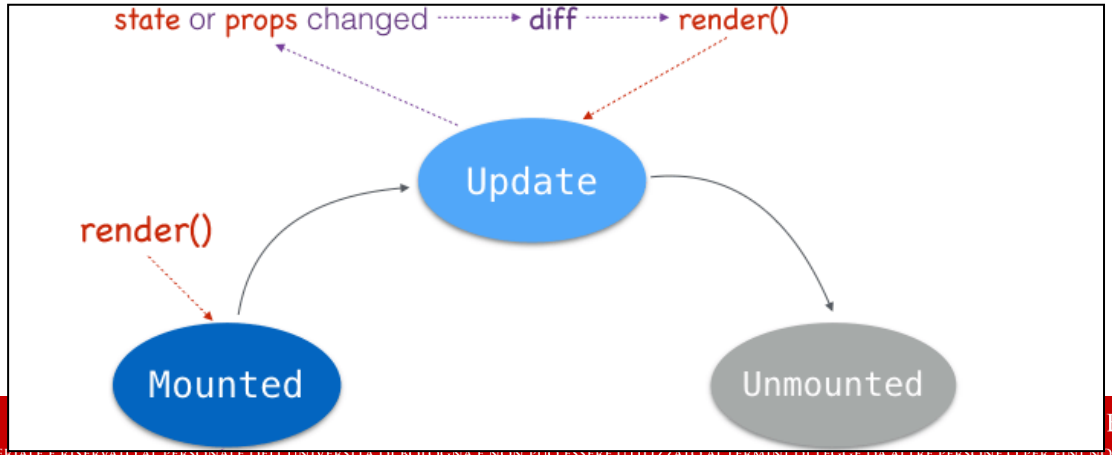


```
class App extends React.Component {

  render(){
   return (
    <div>
       <p>Ciao <b>{this.props.nome}</b>,
             benvenuto in React.js!</p>
       <InfoCorsi corsi={this.props.corsi} />
    </div>
    )
  }
}
```

```
class InfoCorsi extends React.Component {

 render(){

    const infoCorsi = this.props.corsi.map(
         (corso) =>
          <div className="corso" key={corso.nome}>
            <div>{corso.nome}</div>
            <div>Semestre: {corso.semestre}</div>
          </div>
       )
    return <div className="infoCorsi">{infoCorsi}</div>
}}
```

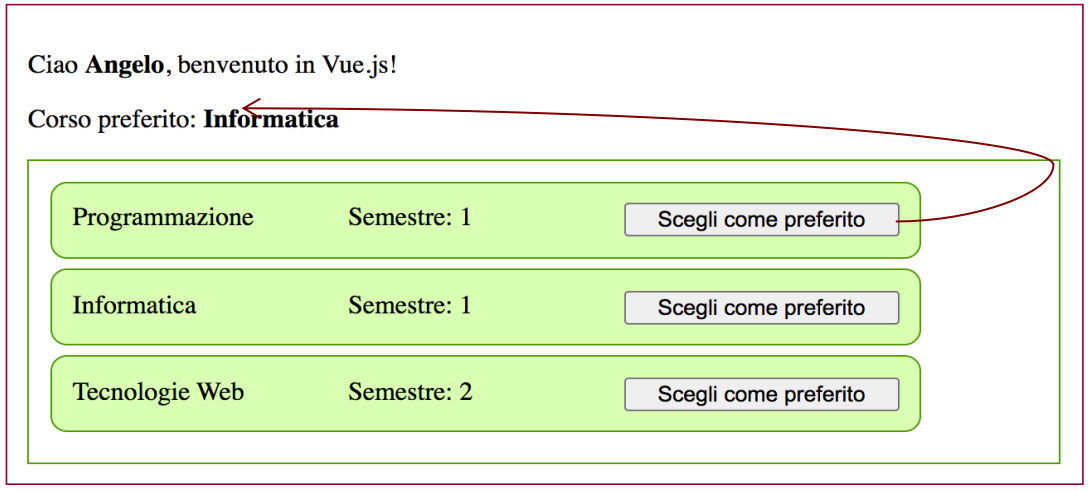
Proprietà e stato di un componente

- Oltre alle proprietà, un componente può essere dotato di uno **stato**, ossia un oggetto accessibile e modificabile dall'interno del componente
  - si usa il metodo `setState()` per aggiornare lo stato di un componente
- La modifica della stato, così come avviene per le proprietà, causa l'aggiornamento dell'interfaccia (invocazione di `render()`)
- Ogni componente attraversa le fasi di *mount*/*update*/*unmount* alle quali è possibile associare *hooks*, come in Vue anche se con nomi diversi



Tornando al nostro esempio

- Anche React ha un **flusso *one-way*** di modifica delle proprietà, che possono essere modificate solo dall'esterno (dall'alto verso il basso)
- Per comunicare dal basso verso l'alto si usano funzioni di callback passate dalla componente padre alla componente figlio
- Nel nostro esempio:
  - aggiungiamo una proprietà allo stato per memorizzare il corso scelto
  - aggiungiamo un handler per l'evento *onclick* sul bottone
  - passiamo l'handler (funzione di callback) tramite una proprietà del componente



```
class App extends React.Component {

  constructor(props){ … } //costruttore: inizializzo lo stato

  render(){
       return (…
        <p>Corso preferito: <b>{this.state.preferito}</b></p>
       <InfoCorsi corsi={this.props.corsi}
       scelto={this.mostraPreferito.bind(this)}/>
       …)
  }

  mostraPreferito(p){
    this.setState({
      preferito : p
    })
 }
}
```

Template InfoCorsi

```
<button onClick={this.props.scelto.bind(this,corso.nome)}>
Scegli come preferito</button>
```

Usiamo `bind()` per legare la funzione all'oggetto corrente (`this`) e passare il parametro.
Viene creata una copia della funzione che sarà invocata successivamente.

Generare componenti

- Come per altri framework, esistono diversi pacchetti Node per creare e gestire componenti all'interno di applicazioni React da linea di comando
- Generano opportunamente directory e file JS, CSS, ecc.
- Permettono di configurare diverse opzioni del progetto da un menù interattivo
- Ad esempio **`generate-react-cli`** produce un file **`generate-react-cli.json`** con le opzioni globali del progetto e dei componeneti e crea componenti

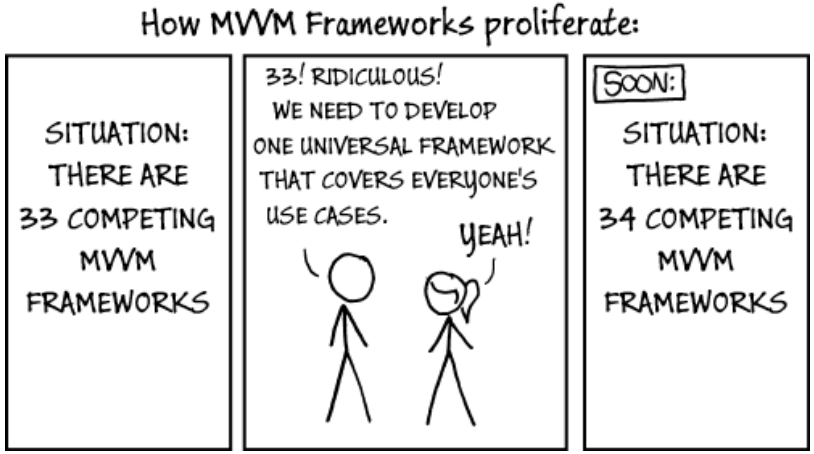
```
npm install generate-react-cli    //installa package

npx generate-react-cli            //prima esercuzione e
                                    configurazione ambiente

npx generate-react-cli component MyComponent    // creazione
                                                // componente
```

Conclusioni su React

- Oggi abbiamo parlato degli elementi principali di React, in relazione ad altri framework
- Anche React pone l'accento su componenti riutilizzabili e programmabili
- Altri moduli sono facilmente integrabili nel framework, ad esempio React Router per la navigazione (https://reactrouter.com/) o Redux (https://redux.js.org/) per mantenere/gestire lo stato
- Buon compromesso tra facilità d'uso, velocità di sviluppo, complessità ed efficienza
- Forte legame con JSX

Conclusioni sui framework

- I framework spingono decisamente verso un modello computazionale e non dichiarativo
- Maggiore velocità, produttività e flessibilità ma anche dipendenza dal framework (librerie e linguaggi proprietari) e meno visibilità e standardizzazione



Adattato da: https://xkcd.com/927/