Esercitazione 6 – React.js

Tecnologie Web T

Prof. Fedeli Massimo

IIS Fermi Sacconi Cpia

November 28, 2025

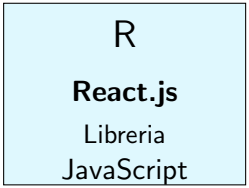
Cos'è React.js?

Definizione

React.js è una **libreria JavaScript** per la creazione di interfacce utente web moderne e interattive.

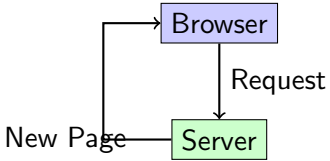
Caratteristiche principali:

- Sviluppato da Facebook
- Open Source
- Component-based
- Virtual DOM
- Dichiarativo



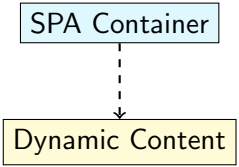
Single Page Application (SPA)

Approccio Tradizionale:



Caricamento di nuove pagine ad ogni interazione

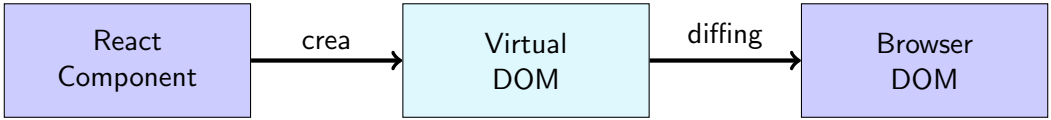
Single Page Application:



Nessun reload

La pagina evolve dinamicamente senza ricaricare

Virtual DOM



Solo le modifiche effettive vengono applicate al DOM reale

Vantaggi del Virtual DOM

- **Performance**: aggiornamenti ottimizzati
- **Efficienza**: solo i cambiamenti reali vengono applicati
- **Astrazione**: lo sviluppatore non manipola direttamente il DOM

Il Virtual DOM di React

Definizione

Il **Virtual DOM** è una rappresentazione in memoria (in JavaScript) della struttura DOM reale della pagina.

- È una copia leggera dell'albero DOM effettivo del browser
- Mantenuto da React come oggetto JavaScript
- Permette operazioni veloci senza toccare il DOM reale

Come Funziona il Virtual DOM

Processo di aggiornamento:

1. **Creazione del nuovo Virtual DOM**
   React crea un nuovo albero Virtual DOM con i nuovi dati

2. **Confronto (Diffing)**
   Algoritmo efficiente confronta il nuovo Virtual DOM con la versione precedente

3. **Riconciliazione**
   React calcola il modo più efficiente per aggiornare il DOM reale

4. **Aggiornamento del DOM reale**
   Solo le parti effettivamente cambiate vengono aggiornate nel browser

Vantaggi del Virtual DOM

Perché è efficiente?

Manipolare direttamente il DOM è costoso in termini di performance

Vantaggi principali:

- ✓ Le operazioni sul Virtual DOM sono molto più veloci (solo JavaScript in memoria)
- ✓ React raggruppa molteplici cambiamenti e li applica in un'unica operazione
- ✓ Vengono aggiornati solo gli elementi effettivamente modificati
- ✓ Ideale per applicazioni con interfacce dinamiche che cambiano frequentemente

Preparazione Ambiente di Sviluppo

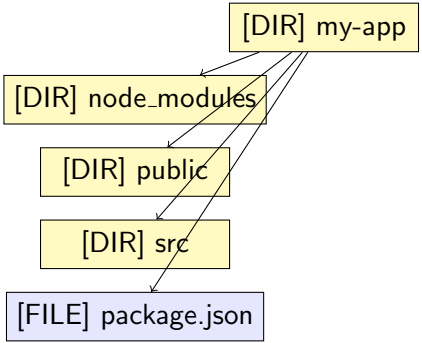
In Laboratorio

1. Accedere a `C:\Applicativi\TecnologieWeb\`
2. Copiare `react.zip` su pen drive USB
3. Scompattare il file (operazione lunga)
4. Risultato: directory `my-app` pronta

Da Casa

```
# 1. Installare Node.js da https://
   nodejs.org
# 2. Creare applicazione React
npx create-react-app my-app
# 3. Avviare l'applicazione
cd my-app
npm start
```

Struttura del Progetto React



Descrizione cartelle:

- `node_modules`: librerie e dipendenze
- `public`: template HTML
- `src`: codice sorgente JavaScript
- `package.json`: configurazione progetto

Comando di avvio

`npm start` → Avvia server su porta 3000

Struttura di un Progetto React

Cartelle e File Principali

`my-react-app/`

- `node_modules/` - Librerie e dipendenze installate
- `public/` - File statici accessibili pubblicamente
  - `index.html` - Template HTML principale
  - `favicon.ico` - Icona del sito
- `src/` - Codice sorgente dell'applicazione
  - `index.js` - Punto di ingresso dell'app
  - `App.js` - Componente principale
  - `App.css` - Stili del componente principale
  - `components/` - Componenti riutilizzabili
- `package.json` - Configurazione e dipendenze del progetto
- `.gitignore` - File da ignorare in Git

JSX - JavaScript XML

Cos'è JSX?

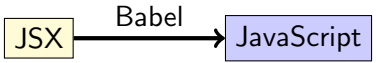
Sintassi che permette di scrivere tag HTML all'interno di codice JavaScript

Senza JSX:

```
const element =
React.
      createElement
      (
'h1',
{className: '
      greeting'},
'Hello, World!'
);
```

Con JSX:

```
const element =
      (
<h1 className="
      greeting">
Hello, World!
</h1>
);
```



JSX - JavaScript XML

Cos'è JSX?

Sintassi che permette di scrivere tag HTML all'interno di codice JavaScript

Senza JSX:

```
const element =
React.createElement(
'h1',
{className: 'greeting'},
'Hello, World!'
);
```

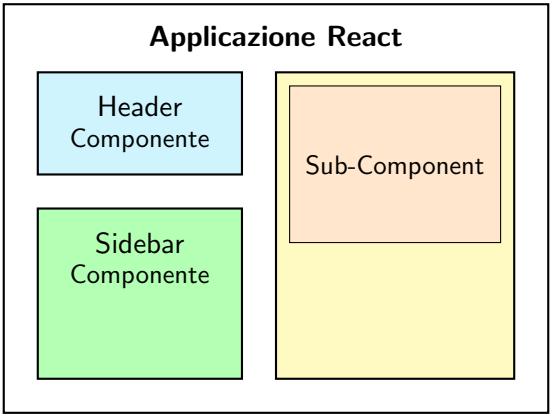
const element = React.createElement('h1',

Con JSX:

```
const element = (
<h1
className="greeting">
Hello, World!
</h1>
);
```

const element = (¡h1 class-Name="greeting"¿ Hello, World! ¡/h1¿);

Approccio a Componenti



Vantaggi

- **Riusabilità**: componenti utilizzabili ovunque
- **Modularità**: costruzione per composizione

Tipi di Componenti

Function Component

Function

Props
Return JSX

- Più semplici
- Stateless
- Solo props

Class Component

Class

Props + State
Lifecycle
Return JSX

- Più complessi
- Stateful
- Props + State

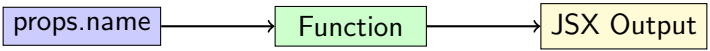
Nota

Entrambi i tipi devono restituire JSX tramite `return`

Function Component - Esempio

```
function Welcome(props) {
        return <h1>Benvenuto, {props.name
           }!</h1>;
}

// Utilizzo
<Welcome name="Massimo" />
```



Caratteristiche

- Sintassi più concisa
- Ricevono props come parametro
- Ideali per componenti di presentazione

Class Component - Esempio

```
class Welcome extends React.Component {
        render() {
                return <h1>Benvenuto, {this.props.
                    name}!</h1>;
        }
}

// Utilizzo
<Welcome name="Massimo" />
```

Accesso alle props

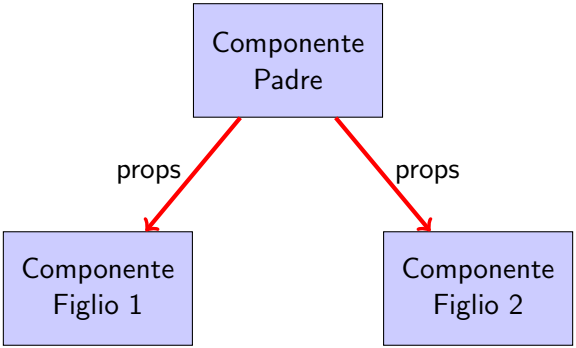
Nelle classi si usa `this.props` invece di `props`

React.Component

constructor()

render()

Props - Proprietà Immutabili



Caratteristiche delle Props

- **Immutabili**: non possono essere modificate dal componente figlio
- **Unidirezionali**: flusso dati dal padre al figlio
- **Configurazione**: usate per parametrizzare i componenti

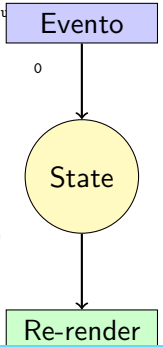
State - Stato del Componente

```
class Counter extends
React.Component {
        constructor(
             props) {
                super(
                     prop
                     );
                this.
                     stat
                      =
                     {
                        cou
                     };
        }

        render() {
                return (
                <div>
                {this.
                     state
                     .
                     count
                     }
                </div>
                );
```

Cos'è lo State?

Dati che cambiano nel tempo durante il ciclo di vita del componente



State vs Props

Props

- Immutabili
- ↓ Dal padre
- ⋆ Configurazione
- [FILE] Read-only

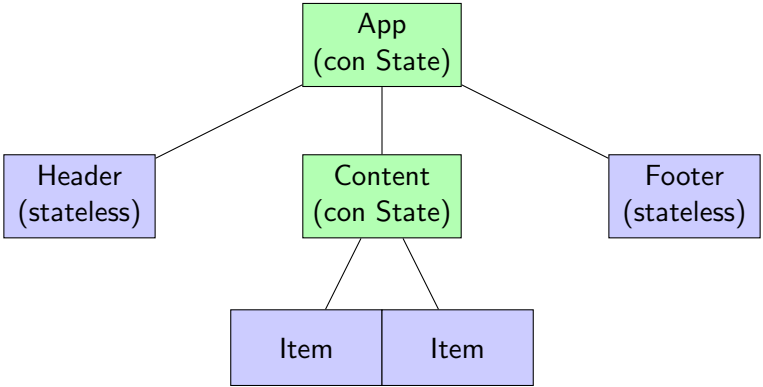
State

- ↔ Mutabile
- Interno
- ⟳ Dinamico
- Read-write

Best Practice

- Minimizzare i componenti con state
- Usare props per passare dati ai figli
- State solo dove necessario

Gerarchia di Componenti

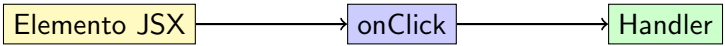


Componenti ai vertici mantengono
lo state e passano dati via props

Gestione degli Eventi

```
class Button extends React.Component {
        handleClick(e) {
                console.log('Pulsante premuto -
                    Evento: ' + e.type);
        }

        render() {
                return (
                <button onClick={this.handleClick}>
                Cliccami
                </button>
                );
        }
}
```



Eventi comuni

Binding del this negli Eventi

Problema: accedere allo state nell'handler

Versione 1: Bind nel costruttore

```
constructor(props) {
        super(props);
        this.state = {on
             : false};
        this.handleClick
             =
        this.handleClick
             .bind(this)
             ;
}

handleClick() {
        this.setState({
             on: !
                  this
                  .
                  state
                  .on
        });
}

render() {
        return (
        <button onClick=
        {this.
```

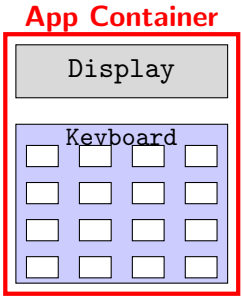
Versione 2: Arrow function

```
constructor(props) {
        super(props);
        this.state = {on
             : false};
        // NO bind
             necessario
}

handleClick() {
        this.setState({
             on: !
                  this
                  .
                  state
                  .on
        });
}

render() {
        return (
        <button onClick=
        {() => this.
             handleClick
             ()}>
```

Esempio: Calcolatrice React



Componenti necessari:

1. Display (figlio, stateless)
   - Visualizza espressione
   - Visualizza risultato
2. Keyboard (figlio, stateless)
   - Bottoni numerici
   - Operatori aritmetici
3. App (padre, stateful)
   - Gestisce lo state
   - Gestisce eventi
   - Compone i componenti

Calcolatrice: Componente Display

```
function Display(props) {
        return (
        <div className="display">
        <input
        type="text"
        value={props.value}
        readOnly
        />
        </div>
        );
}
```

Caratteristiche

- Componente **function** (stateless)
- Riceve il valore da visualizzare tramite **props**
- Campo di input in sola lettura
- Riutilizzabile

Calcolatrice: Componente Keyboard

```
function Keyboard(props) {
        return (
        <div className="keyboard">
        <button onClick={() => props.onButtonClick('7')}>7</button>
        <button onClick={() => props.onButtonClick('8')}>8</button>
        <button onClick={() => props.onButtonClick('9')}>9</button>
        <button onClick={() => props.onButtonClick('+')}>+</button>
        {/* altri bottoni... */}
        <button onClick={() => props.onButtonClick('=')}>=</button>
        </div>
        );
}
```

Caratteristiche

- Riceve funzione handler tramite props
- Ogni bottone invoca `props.onButtonClick`
- Non gestisce logica, solo presentazione

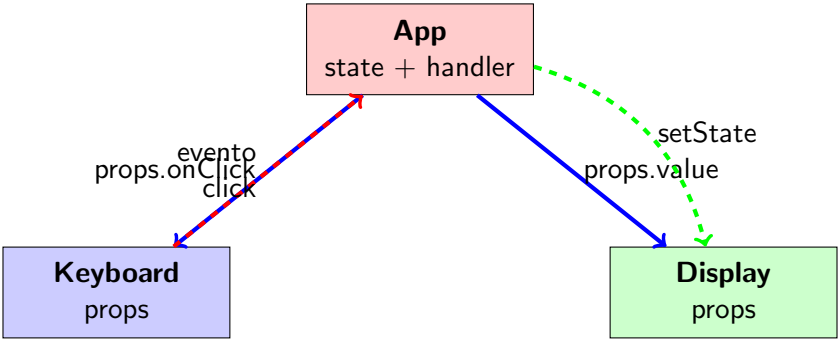
Calcolatrice: Componente App

```
class App extends React.Component {
        constructor(props) {
                super(props);
                this.state = { expression: '', result: '' };
        }

        handleButtonClick = (value) => {
                if (value === '=') {
                        try {
                                this.setState({ result: eval(this.state.expression) });
                        } catch (e) {
                                this.setState({ result: 'Errore' });
                        }
                } else if (value === 'C') {
                        this.setState({ expression: '', result: '' });
                } else {
                        this.setState({ expression: this.state.expression + value });
                }
        }

        render() {
                return (
                <div>
                <Display value={this.state.expression || this.state.result} />
                <Keyboard onButtonClick={this.handleButtonClick} />
                </div>
                );
        }
}
```

Flusso dei Dati nella Calcolatrice



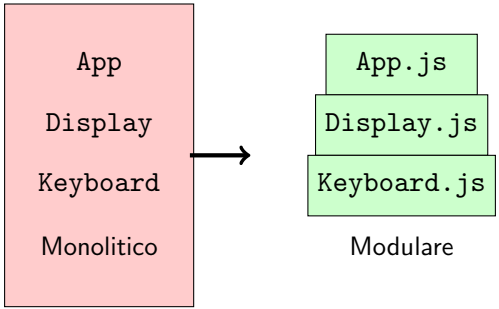
Pattern

Top-down data flow: i dati fluiscono dall'alto verso il basso tramite props

Modularità e Riusabilità

Problema

Applicazione monolitica in un singolo file = difficile riutilizzare componenti



✓ Riutilizzabile

Soluzione

Import ed Export

Display.js:

```
import React from 'react';

function Display(props) {
        return <div>{props.value}</div>;
}

export default Display;
```

App.js:

```
import React from 'react';
import Display from './Display';
import Keyboard from './Keyboard';

class App extends React.Component {
        // ...
}
```

Toolchain di Sviluppo

Cos'è una Toolchain?

Insieme di strumenti integrati che facilitano lo sviluppo professionale

Node.js	Webpack	Babel
Runtime JavaScript	Module Bundler	JSX Compiler

Create React App

Vantaggi

- Hot reload durante sviluppo
- Individuazione errori

Comandi NPM Principali

npm start

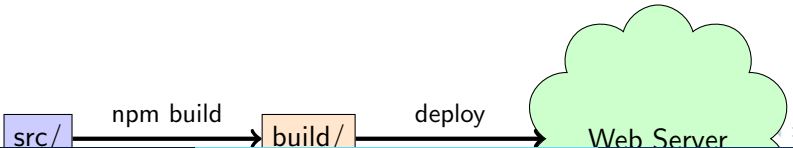
```
npm start
```

Avvia server di sviluppo su `localhost:3000`
Hot reload automatico delle modifiche

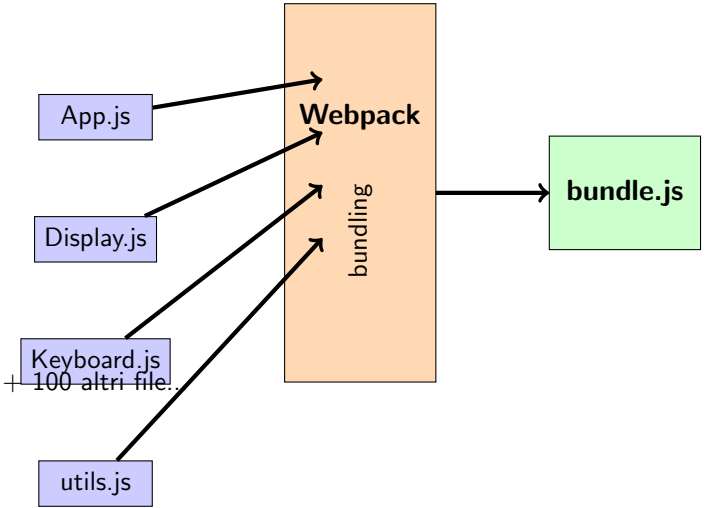
npm run build

```
npm run build
```

Crea directory `build/` con file ottimizzati per produzione
Pronta per deploy su web server



Application Bundling



Esercizi Proposti

Esercizio 1: Doppio Display

Modificare la calcolatrice per avere:

- Display 1: mostra l'espressione in composizione
- Display 2: mostra il risultato finale
- Tasto 'C': resetta entrambi i display

Suggerimento: riutilizzare il componente Display esistente!

Esercizio 2: Calcolatrice Scientifica

Aggiungere funzionalità scientifiche:

- Nuovo tastierino con: $\log_e(x)$, $\sqrt{x}$, e^x, $\frac{1}{x}$
- Applicare operatori all'espressione corrente
- Posizionare sotto la tastiera esistente

Best Practices React

Struttura:

- ✓ Componenti piccoli e focalizzati
- ✓ Separare logica e presentazione
- ✓ Un componente per file
- ✓ Naming chiaro e consistente

State Management:

- ✓ Minimizzare componenti stateful
- ✓ State il più in alto possibile
- ✓ Props per passare dati in basso

Performance:

- ✓ Evitare binding in render
- ✓ Usare keys nelle liste
- ✓ Componenti puri quando possibile

Codice:

- ✓ JSX leggibile e indentato
- ✓ Commenti dove necessario
- ✓ PropTypes per validazione
- ✓ Testing dei componenti

Risorse Utili

Documentazione Ufficiale

- **React**: `https://reactjs.org/docs`
- **Create React App**: `https://create-react-app.dev/`
- **Babel**: `https://babeljs.io/`

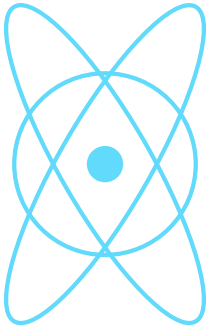
Tutorial e Guide

- React Tutorial ufficiale
- FreeCodeCamp React Course
- React Patterns
- Awesome React (GitHub)

Tools

- React Developer Tools (Chrome/Firefox)
- VS Code + ES7 React snippets

Conclusioni



Cosa abbiamo imparato

- Fondamenti di React.js e Single Page Applications
- Componenti function e class
- Props e State
- Gestione eventi

Tecnologie Web T

A.A. 2020–2021

Home Page del corso:
`http://lia.disi.unibo.it/Courses/twt2021-info/`

Versione elettronica:
`L.06.React.pdf`
`L.06.React-2p.pdf`