

# React.js

## Tecnologie del Web

Prof. Fedeli Massimo

IIS Fermi Sacconi Cpia

November 30, 2025



# Cos'è React.js?

## Definizione

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

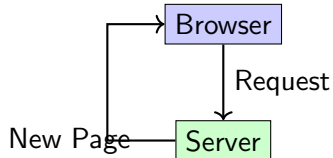
## Caratteristiche principali:

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



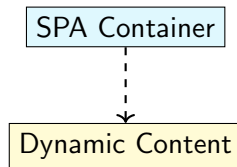
# Single Page Application (SPA)

## Approccio Tradizionale:



Caricamento di nuove pagine ad ogni interazione

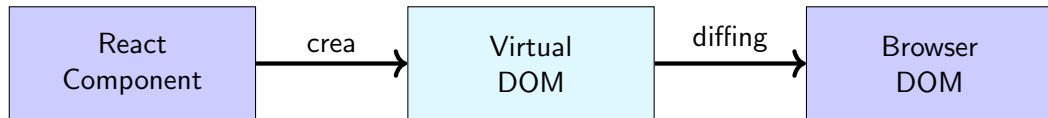
## Single Page Application:



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

## 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

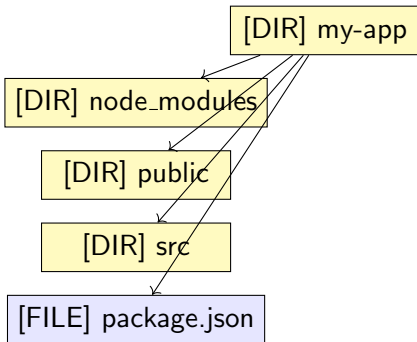
- 1 Accedere a C:\Applicativi\TecnologieWeb\
- 2 Copiare react.zip su pen drive USB
- 3 Scompattare il file (operazione lunga)
- 4 Risultato: directory my-app pronta

## 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

## Cos'è JSX

JSX è un'estensione sintattica di JavaScript che permette di descrivere l'interfaccia utente utilizzando una sintassi simile all'HTML. Viene compilato in chiamate a `React.createElement`.

- JSX non è obbligatorio, ma rende il codice più leggibile.
- Il browser non lo interpreta direttamente: serve una fase di trasformazione (Babel).
- Permette di combinare logica JavaScript e struttura dell'interfaccia.

# JSX - JavaScript XML

## Cos'è JSX?

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

### Senza JSX:

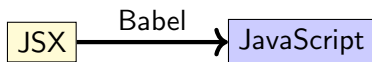
```
1  const element =  
2  
3  React.createElement('h1',  
4  {className: 'greeting'},  
5  'Hello, World!'  
                                     );
```

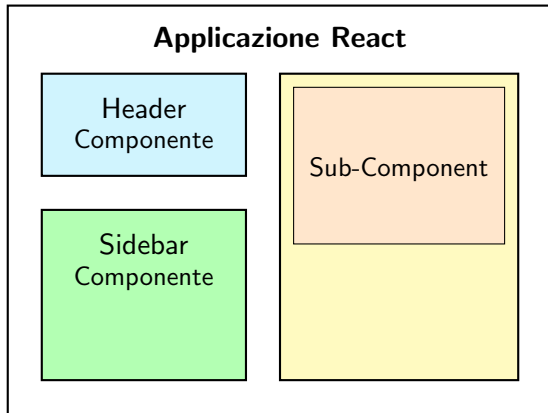
```
const element = React.createElement('h1',  
  className: 'greeting', 'Hello, World!');
```

### Con JSX:

```
1  const element = (<h1  
2  
3  className="greeting">  
4  Hello, World! </h1>  
                                     );
```

```
const element = (jh1 class-  
Name="greeting" i Hello, World! i/h1i );
```





# Vantaggi dei Componenti React

## Riusabilità

### Componenti utilizzabili ovunque nell'applicazione

- Scrivi una volta, usa molte volte
- Risparmio di tempo e codice duplicato
- Coerenza visiva e funzionale in tutta l'app

### Esempio:

```
// Definisci il componente una volta
function Button({ testo, onClick }) {
  return <button onClick={onClick}>{testo}</button>;
}

// Riutilizzalo ovunque
<Button testo="Salva" onClick={handleSave} />
<Button testo="Elimina" onClick={handleDelete} />
<Button testo="Annulla" onClick={handleCancel} />
```

# Vantaggi dei Componenti React (continua)

## Modularità

### Costruzione dell'interfaccia per composizione

- Suddivisione in unità piccole e gestibili
- Ogni componente ha una responsabilità specifica
- Facile combinare componenti per creare UI complesse
- Sviluppo parallelo e isolato

### Esempio di composizione:

```
function App() {  
  return (  
    <div>  
      <Header />  
      <Sidebar />  
      <MainContent>  
        <ArticleList />  
        <Pagination />  
      </MainContent>  
      <Footer />  
    </div>  
  )  
}
```



## Manutenibilità

### Focus separato su logica e layout

- Codice organizzato e facilmente comprensibile
- Bug più facili da individuare e correggere
- Testing semplificato (componenti isolati)
- Aggiornamenti localizzati senza effetti collaterali
- Documentazione più chiara

## Esempio: Componente Manutenibile

```
// Modifica solo il componente necessario
function UserCard({ user }) {
  // Logica isolata
  const formattedDate = formatDate(user.
    registrationDate);

  // Layout chiaro
  return (
    <div className="card">
      <img src={user.avatar} alt={user.name} />
      <h3>{user.name}</h3>
      <p>Iscritto dal: {formattedDate}</p>
    </div>
  );
}
```

## Vantaggi di questo approccio

- Logica separata dalla presentazione
- Facile da testare e modificare
- Codice chiaro e leggibile

# Riepilogo Vantaggi

Vantaggio	Beneficio Pratico
Riusabilità	Riduzione del codice duplicato e tempi di sviluppo più brevi
Modularità	Architettura scalabile e sviluppo team più efficiente
Manutenibilità	Debug più rapido e aggiornamenti sicuri

## Best Practice

Mantieni i componenti piccoli, focalizzati e con responsabilità singole (Single Responsibility Principle)

# 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`

# Cosa sono i Componenti React?

## Definizione

I componenti sono i **mattoni fondamentali** di un'applicazione React. Ogni componente è una funzione o classe JavaScript che accetta input (props) e restituisce elementi React che descrivono l'interfaccia utente.

## Caratteristiche:

- Indipendenti e riutilizzabili
- Accettano input (props)
- Restituiscono JSX
- Possono avere stato interno

## Analogia:

**Componenti = Funzioni**

Come le funzioni in matematica, i componenti trasformano input in output

# Tipi di Componenti

## Function Component

### Function

Props  
Return JSX

- Più semplici
- Moderni (con Hooks)
- Raccomandati oggi

## Class Component

### Class

Props + State  
Lifecycle  
Return JSX

- Più complessi
- Approccio tradizionale
- Meno usati oggi

# Function Component - Esempio

## Componente semplice:

```
function Benvenuto(props) {  
    return <h1>Ciao, {props.nome}!</h1>;  
}  
  
// Utilizzo  
<Benvenuto nome="Mario" />
```

## Con destructuring (più comune):

```
function Benvenuto({ nome }) {  
    return <h1>Ciao, {nome}!</h1>;  
}
```

## Arrow function (sintassi moderna):

```
const Benvenuto = ({ nome }) => {  
    return <h1>Ciao, {nome}!</h1>;  
}
```



# Function Component con Hooks

## Componente con stato (useState):

```
import { useState } from 'react';

function Contatore() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Hai cliccato {count} volte</p>
      <button onClick={() => setCount(count + 1)}>
        Incrementa
      </button>
    </div>
  );
}
```

# Class Component - Esempio

```
import React, { Component } from 'react';

class Contatore extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  incrementa = () => {
    this.setState({ count: this.
      state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>Hai cliccato {this.state.
```

# Confronto: Function vs Class Component

Aspetto	Function Component	Class Component
Sintassi	Più semplice	Più verbosa
Stato	Hooks (useState)	this.state
Props	Parametro funzione	this.props
Lifecycle	useEffect	componentDidMount, ecc.
Performance	Leggermente migliori	Buone
Trend	<b>Raccomandati</b>	Legacy

## Best Practice Attuale

React raccomanda l'uso di **Function Component con Hooks** per tutti i nuovi progetti.

# Anatomia di un Componente

```
// 1. Import necessari
import React, { useState } from 'react';

// 2. Definizione del componente
function MioComponente({ titolo, descrizione
  }) {
  // 3. Stato locale (opzionale)
  const [attivo, setAttivo] = useState
    (false);

  // 4. Funzioni helper (opzionale)
  const handleClick = () => {
    setAttivo(!attivo);
  };

  // 5. Return del JSX
  return (
```

# Props: Comunicazione tra Componenti

## Cosa sono le Props?

Le **props** (properties) sono gli argomenti passati ai componenti React. Funzionano come i parametri delle funzioni JavaScript.

## Caratteristiche delle props:

- **Read-only:** non possono essere modificate dal componente
- **Unidirezionali:** fluiscono dal genitore al figlio
- **Qualsiasi tipo:** stringhe, numeri, oggetti, funzioni, componenti
- **Dinamiche:** possono cambiare nel tempo

## Regola d'oro

Un componente non deve mai modificare le proprie props. Deve comportarsi come una funzione pura rispetto ai suoi input.

# Esempio Completo: Card Component

```
function UserCard({ nome, email, avatar,
  isOnline }) {
  return (
    <div className="user-card">
      <img src={avatar} alt={nome} />
      <div className="user-info">
        <h3>{nome}</h3>
        <p>{email}</p>
        {isOnline && <span className="badge
          ">Online</span>}
      </div>
    </div>
  );
}

// Utilizzo
function App() {
```

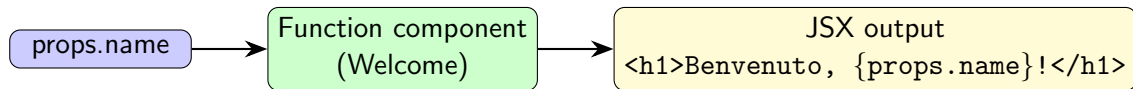
## Punti Chiave

- ❶ I componenti sono i **blocchi costruttivi** di React
- ❷ Possono essere **Function** o **Class** (usa Function!)
- ❸ Accettano **props** come input
- ❹ Restituiscono **JSX** come output
- ❺ Possono avere **stato interno** (con Hooks)
- ❻ Sono **componibili** e **riutilizzabili**

**Componente = Input (Props) + Logica + Output (JSX)**

# Function Component — Esempio

```
function Welcome(props) {  
  return <h1>Benvenuto, {props.name}</h1>;  
}  
  
// Utilizzo  
<Welcome name="Massimo" />
```



## Caratteristiche

- Componenti definite come funzioni: accettano props e restituiscono JSX.
- Il nome del componente deve iniziare con lettera maiuscola (convenzione di React).
- Ideali per componenti di presentazione; facilmente combinabili con Hooks.



# Class Component - Esempio

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

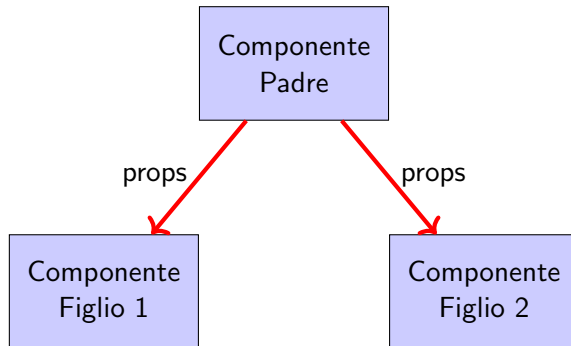
## Accesso alle props

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

**React.Component**

constructor()  
render()

# Props - Proprietà Immutabili



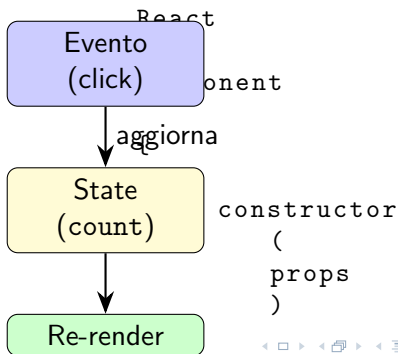
## Caratteristiche delle Props

- **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

## Cos'è lo state

Lo state è l'insieme di dati locali al componente che possono cambiare nel tempo e provocare il re-render del componente stesso.



# State vs Props

## Props

- Immutabili

↓ Dal padre

★ Configurazione

[FILE] Read-only

## State

↔ Mutabile

- Interno

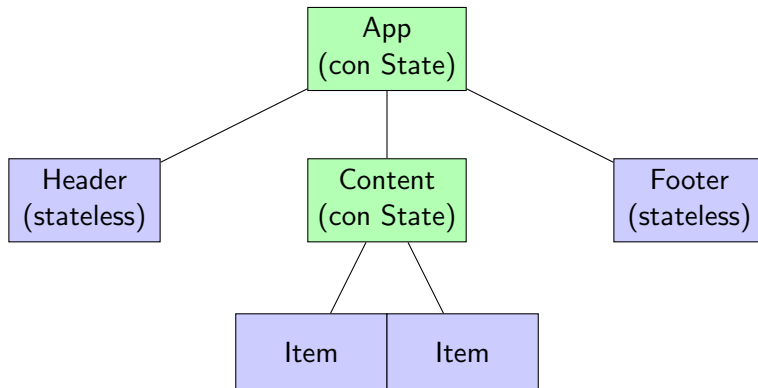
🔄 Dinamico

- 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>  
    );  
  }  
}
```

Elemento JSX

onClick

Handler

## Eventi comuni

# Binding del this negli Eventi

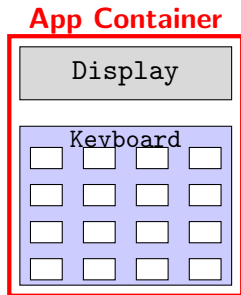
**Problema:** accedere allo state nell'handler

## 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  
          ()}>  
    </button>  
  );  
}
```



## 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

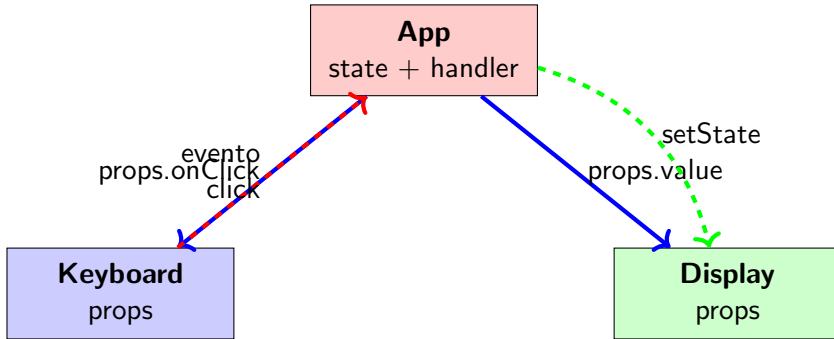
# Calcolatrice: Componente App

```
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 onClick={this.handleClick} />
      </div>
    );
  }
}
```

# Flusso dei Dati nella Calcolatrice



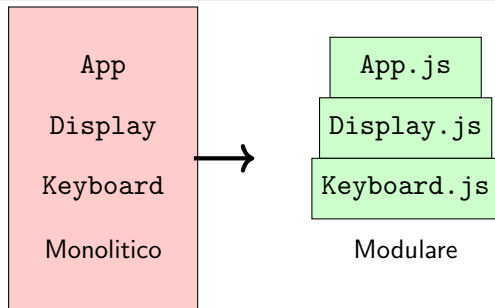
## Pattern

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

# Modularità e Riutilizzabilità

## 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:

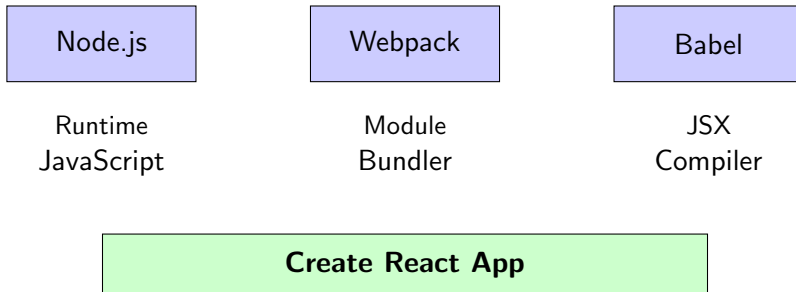
```
import React from 'react';
import Display from './Display';
import Keyboard from './Keyboard';

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

# Toolchain di Sviluppo

## Cos'è una Toolchain?

Insieme di strumenti integrati che facilitano lo sviluppo professionale



## Vantaggi

- Hot reload durante sviluppo
- Individuazione errori

# Comandi NPM Principali

## npm start

```
npm start
```

Avvia server di sviluppo su localhost:3000

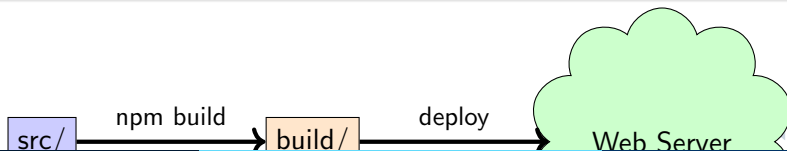
Hot reload automatico delle modifiche

## npm run build

```
npm run build
```

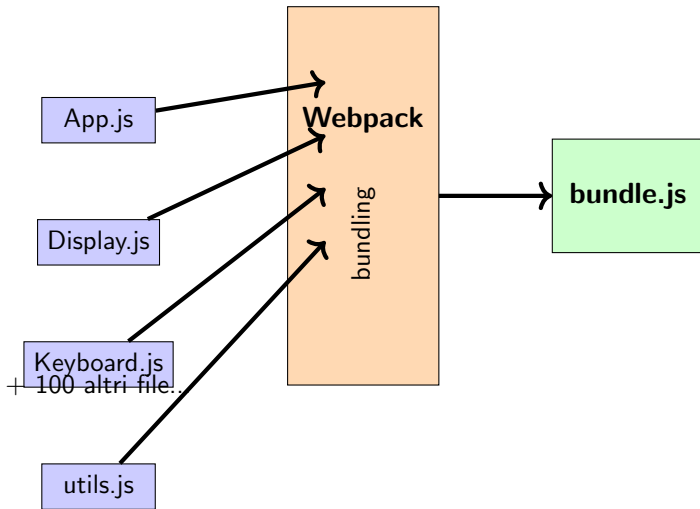
Crea directory build/ con file ottimizzati per produzione

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



## Documentazione Ufficiale

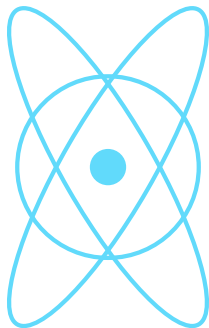
- **React:** <https://reactjs.org/docs>
- **Create React App:** <https://create-react-app.dev/>
- **Babel:** <https://babeljs.io/>

## 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



## Cosa abbiamo imparato

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

## 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



# React.js

## Tecnologie del Web

Prof. Fedeli Massimo

IIS Fermi Sacconi Cpia

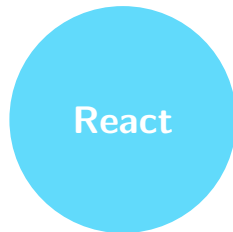
November 30, 2025



- 1 Introduzione a React
- 2 Concetti Fondamentali
- 3 Hooks Principali
- 4 Applicazione Pratica
- 5 Prossimi Passi

# Cos'è React?

- **Libreria JavaScript** creata da Facebook (Meta) nel 2013
- Per costruire **interfacce utente** interattive
- Basata su **componenti riutilizzabili**
- Approccio **dichiarativo**
- Utilizzata da: Facebook, Instagram, Netflix, Airbnb



# Perché React?

## Vantaggi

- + **Componenti riutilizzabili**: scrivi una volta, usa ovunque
- + **Virtual DOM**: aggiornamenti efficienti
- + **Flusso dati unidirezionale**: più facile da debuggare
- + **Ecosistema ricco**: migliaia di librerie
- + **Community enorme**: supporto e risorse abbondanti

## Definizione

I componenti sono funzioni che restituiscono JSX

```
// Componente semplice
function Benvenuto() @\{@
return <h1>Ciao, benvenuto in React
    !</h1>;
@\}@

// Componente con props
function Saluto(@\{@ nome @\}@) @\{@
return <h1>Ciao, @\{@nome@\}@!</h1>;
@\}@
```

## Definizione

Le **props** sono dati passati dal padre al figlio. Sono **immutabili**.

```
function TaskItem(@\{@ task,  
    onDelete @\}@) @\{@  
    return (  
    <div>  
    <span>@\{@task.text@\}@</span>  
    <button onClick=@\{@() => onDelete(  
        task.id)\}@>  
        Elimina  
    </button>  
    </div>  
    );  
    @\}@
```

## Definizione

Lo **state** contiene dati che cambiano e causano il ri-rendering.

```
import @\{@ useState @\}@ from '
  react';

function Contatore() @\{@
  const [conteggio, setConteggio] =
    useState(0);

  return (
    <div>
      <p>Hai cliccato @\{@conteggio@\}@
        volte</p>
      <button onClick=@\{@() =>
        setConteggio(conteggio + 1)\}@>
        Incrementa
```

# State vs Props

Caratteristica	Props	State
Mutabilità	Immutabili	Mutabili
Origine	Passate dal padre	Gestite dal componente
Modifica	NO	SI (con setState)
Re-render	NO	SI (quando cambia)

## Regola d'oro

Se un dato **cambia** → useState

Se un dato è **passato dal padre** → props

## Sintassi

```
const [stato, setStato] = useState(valoreIniziale);
```

```
function FormEsempio() @\{@  
  const [nome, setNome] = useState('')  
  ;  
  const [eta, setEta] = useState(0);  
  
  return (  
    <input  
      value=@\{@nome@\}  
      onChange=@\{@(e) => setNome(e.target  
        .value)\}  
    />  
  );  
@}\}
```



## Scopo

Eeguire **effetti collaterali**: chiamate API, sottoscrizioni, timer

```
import @\{@ useState, useEffect @\}@
  from 'react';

function Esempio() @\{@
  const [count, setCount] = useState
    (0);

  // Si esegue dopo ogni render
  useEffect(() => @\{@
    document.title = 'Cliccato $@\{
      @count@\}@ volte';
    @\}@);

  // Si esegue solo al mount
```

# Task Manager - Esempio Completo

```
function App() @\{@
  const [tasks, setTasks] = useState([]);
  const [inputValue, setInputValue] = useState
    ('');

  const addTask = () => @\{@
    if (inputValue.trim() === '') return;
    setTasks([...tasks, @\{@
      id: Date.now(),
      text: inputValue,
      completed: false
    @\}@]);
    setInputValue('');
    @\}@;

  const deleteTask = (id) => @\{@
    setTasks(tasks.filter(task => task.id !== id
    ));
    @\}@;
```

## Regola Importante

**NON** modificare mai direttamente lo state!

**SBAGLIATO**

**CORRETTO**

## ① Un componente, una responsabilità

- Mantieni i componenti piccoli e focalizzati

## ② Denomina chiaramente

- Componenti: PascalCase (TaskItem)
- Props/variabili: camelCase (onDelete)

## ③ Immutabilità dello state

- Non modificare mai direttamente lo state

## ④ Key univoche nelle liste

- Mai usare l'indice come key

# Errori Comuni da Evitare

## 1. Modificare direttamente lo state

Non fare: `tasks[0] = newTask;`

## 2. Chiamare hooks condizionalmente

Gli hook vanno sempre in cima alla funzione

## 3. Dimenticare le dipendenze in `useEffect`

Porta a bug difficili da trovare

## 4. Usare l'indice come key

Usa ID univoci invece di indici

## Hooks Avanzati:

- useContext
- useReducer
- useMemo
- useCallback
- useRef

## Concetti:

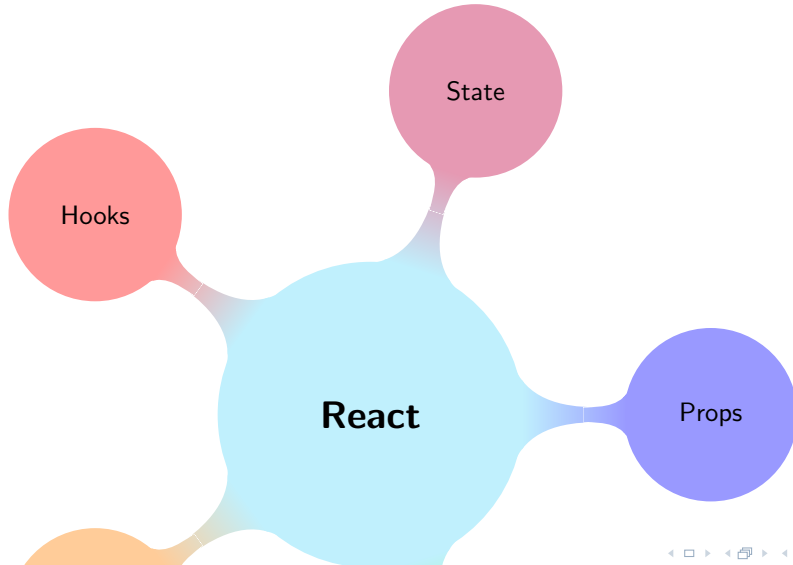
- React Router
- Form avanzati
- Chiamate API
- State Management
- Testing

## Documentazione Ufficiale

<https://react.dev/> - Tutorial interattivi

## Pratica

- Crea progetti reali (TODO app, blog)
- Contribuisci a progetti open source
- Partecipa a coding challenges





# Domande?

Email: [fedeli.massimo@iisfermisacconiceciap.edu.it](mailto:fedeli.massimo@iisfermisacconiceciap.edu.it)

GitHub: [github.com/massimof79](https://github.com/massimof79)

Grazie per l'attenzione!