

# React

Sviluppo di Applicazioni Web Moderne con React

15 novembre 2025



# Indice



# Capitolo 1

## Prefazione

### 1.1 Benvenuti nel Mondo di React

React è una delle librerie JavaScript più popolari e influenti per la costruzione di interfacce utente moderne. Nata nei laboratori di Facebook (ora Meta) nel 2011 e rilasciata come progetto open source nel 2013, React ha rivoluzionato il modo in cui sviluppiamo applicazioni web, introducendo concetti innovativi che hanno cambiato il panorama dello sviluppo frontend.

#### 1.1.1 La Storia di React

React è stato creato da Jordan Walke, un ingegnere software di Facebook, per risolvere problemi specifici che l'azienda stava affrontando con le sue applicazioni web sempre più complesse. Il News Feed di Facebook richiedeva aggiornamenti dinamici continui, e i pattern tradizionali di manipolazione del DOM stavano diventando difficili da gestire e mantenere.

##### **Timeline Storica:**

- **2011:** Jordan Walke crea il primo prototipo di React chiamato "FaxJS"
- **2012:** React viene utilizzato internamente per Instagram
- **Maggio 2013:** React viene rilasciato come open source alla JSConf US
- **2014:** Introduzione del Flux pattern per la gestione dello stato
- **2015:** React Native viene rilasciato, portando React allo sviluppo mobile
- **2016:** React Fiber viene annunciato (nuova architettura di rendering)
- **2017:** React 16 viene rilasciato con supporto completo per Fiber
- **2019:** React Hooks vengono introdotti in React 16.8
- **2020:** Concurrent Mode (sperimentale) viene presentato
- **2022:** React 18 introduce Concurrent Rendering, Suspense migliorato
- **2024:** React 19 introduce Server Components e Actions

#### 1.1.2 La Rivoluzione React

Prima di React, lo sviluppo frontend seguiva principalmente due approcci:

##### **1. Manipolazione Diretta del DOM:**

Listing 1.1: Approccio tradizionale con jQuery

```

1 // Codice jQuery per aggiornare l'interfaccia
2 $('#counter').text('0');
3
4 $('#increment').click(function() {
5     var currentValue = parseInt($('#counter').text());
6     $('#counter').text(currentValue + 1);
7
8     // Gestione manuale della logica di visualizzazione
9     if (currentValue + 1 > 10) {
10         $('#counter').addClass('high-value');
11     }
12
13     // Aggiornamento di altri elementi correlati
14     $('#status').text('Counter incrementato');
15 });

```

## 2. Template Engine con Two-Way Binding:

Listing 1.2: Approccio AngularJS

```

1 // AngularJS controller
2 app.controller('CounterController', function($scope) {
3     $scope.counter = 0;
4
5     $scope.increment = function() {
6         $scope.counter++;
7     };
8 });

```

### La Soluzione React:

React ha introdotto un approccio completamente nuovo basato su componenti, dichiaratività e unidirezionalità dei dati:

Listing 1.3: Approccio React moderno

```

1 import { useState } from 'react';
2
3 function Counter() {
4     const [count, setCount] = useState(0);
5
6     return (
7         <div>
8             <p className={count > 10 ? 'high-value' : ''}>
9                 Contatore: {count}
10             </p>
11             <button onClick={() => setCount(count + 1)}>
12                 Incrementa
13             </button>
14             <p>Stato: Counter {count > 0 ? 'incrementato' : 'iniziale'}</p>
15         </div>
16     );
17 }

```

## 1.2 Perché Scegliere React?

### 1.2.1 1. Component-Based Architecture

React promuove la costruzione di UI come composizione di componenti riutilizzabili e indipendenti. Ogni componente gestisce il proprio stato e può essere combinato per creare interfacce complesse.

**Vantaggi:**

- **Riutilizzabilità:** Scrivi una volta, usa ovunque
- **Manutenibilità:** Ogni componente è isolato e testabile
- **Scalabilità:** Facile aggiungere nuove funzionalità
- **Collaborazione:** Team diversi possono lavorare su componenti diversi

Listing 1.4: Esempio di composizione di componenti

```
1 // Componente Button riutilizzabile
2 function Button({ onClick, children, variant = 'primary' }) {
3   return (
4     <button
5       className={`btn btn-${variant}`}
6       onClick={onClick}
7     >
8       {children}
9     </button>
10  );
11 }
12
13 // Componente Card riutilizzabile
14 function Card({ title, description, onAction }) {
15   return (
16     <div className="card">
17       <h3>{title}</h3>
18       <p>{description}</p>
19       <Button onClick={onAction} variant="secondary">
20         Leggi di più
21       </Button>
22     </div>
23   );
24 }
25
26 // Composizione per creare una dashboard
27 function Dashboard() {
28   const handleCardAction = (id) => {
29     console.log(`Azione su card ${id}`);
30   };
31
32   return (
33     <div className="dashboard">
34       <h1>Dashboard</h1>
35       <div className="cards-grid">
36         <Card
37           title="Vendite"
38           description="Visualizza le vendite mensili"
39           onAction={() => handleCardAction('sales')}
40         />
41         <Card
```

```

42         title="Utenti"
43         description="Gestisci gli utenti registrati"
44         onAction={() => handleCardAction('users')}
45     />
46     <Card
47         title="Analytics"
48         description="Analizza le metriche"
49         onAction={() => handleCardAction('analytics')}
50     />
51 </div>
52 </div>
53 );
54 }

```

### 1.2.2 2. Virtual DOM e Performance

Il Virtual DOM è una delle innovazioni chiave di React. Invece di manipolare direttamente il DOM (operazione costosa), React mantiene una rappresentazione virtuale leggera dell'UI in memoria.

#### Come Funziona:

1. Quando lo stato cambia, React crea un nuovo Virtual DOM tree
2. React confronta il nuovo tree con quello precedente (diffing)
3. React calcola il set minimo di cambiamenti necessari (reconciliation)
4. React applica solo questi cambiamenti al DOM reale (batch update)

#### Vantaggi in Termini di Performance:

- **Aggiornamenti Efficienti:** Solo le parti modificate vengono aggiornate
- **Batch Updates:** Più aggiornamenti vengono raggruppati
- **Predictable:** Il rendering è prevedibile e deterministico

Listing 1.5: Virtual DOM in azione

```

1 function TodoList() {
2     const [todos, setTodos] = useState([
3         { id: 1, text: 'Imparare React', completed: false },
4         { id: 2, text: 'Costruire un progetto', completed: false },
5         { id: 3, text: 'Deploy in produzione', completed: false }
6     ]);
7
8     // Quando toggli un todo, React:
9     // 1. Crea un nuovo Virtual DOM
10    // 2. Confronta con il precedente
11    // 3. Aggiorna solo il singolo elemento modificato nel DOM reale
12    const toggleTodo = (id) => {
13        setTodos(todos.map(todo =>
14            todo.id === id
15                ? { ...todo, completed: !todo.completed }
16                : todo
17        ));
18    };

```



```

19
20     return (
21       <ul>
22         {todos.map(todo => (
23           <li
24             key={todo.id}
25             onClick={() => toggleTodo(todo.id)}
26             style={{
27               textDecoration: todo.completed ? 'line-through'
28                 : 'none'
29             }}
30             >
31               {todo.text}
32             </li>
33           )}}
34       </ul>
35     );
  
```

### 1.2.3 3. Approccio Dichiarativo

React permette di descrivere *cosa* vuoi che l'interfaccia mostri, non *come* manipolare il DOM per ottenerlo. Questo rende il codice più leggibile, prevedibile e facile da debuggare.

#### Imperativo vs Dichiarativo:

Listing 1.6: Approccio Imperativo (Vanilla JS)

```

1  // Descrive COME fare qualcosa
2  const form = document.getElementById('login-form');
3  const errorDiv = document.getElementById('error');
4  const submitBtn = document.getElementById('submit');
5
6  form.addEventListener('submit', function(e) {
7    e.preventDefault();
8
9    // Rimuovi errori precedenti
10   errorDiv.innerHTML = '';
11   errorDiv.classList.remove('visible');
12
13   // Disabilita il bottone
14   submitBtn.disabled = true;
15   submitBtn.textContent = 'Caricamento...';
16
17   // Fai la richiesta
18   fetch('/api/login', { method: 'POST', body: new FormData(form) })
19     .then(response => {
20       if (!response.ok) {
21         // Mostra errore
22         errorDiv.textContent = 'Login fallito';
23         errorDiv.classList.add('visible');
24         // Riabilita bottone
25         submitBtn.disabled = false;
26         submitBtn.textContent = 'Login';
27       } else {
28         // Redirect
29         window.location.href = '/dashboard';
30       }
31     });
  
```

```
32 });
```

Listing 1.7: Approccio Dichiarativo (React)

```

1 // Descrive COSA mostrare
2 function LoginForm() {
3     const [isLoading, setIsLoading] = useState(false);
4     const [error, setError] = useState(null);
5
6     const handleSubmit = async (e) => {
7         e.preventDefault();
8         setIsLoading(true);
9         setError(null);
10
11         try {
12             const response = await fetch('/api/login', {
13                 method: 'POST',
14                 body: new FormData(e.target)
15             });
16
17             if (!response.ok) {
18                 setError('Login fallito');
19             } else {
20                 window.location.href = '/dashboard';
21             }
22         } finally {
23             setIsLoading(false);
24         }
25     };
26
27     // React si occupa di aggiornare il DOM basandosi sullo stato
28     return (
29         <form onSubmit={handleSubmit}>
30             {error && <div className="error visible">{error}</div>}
31
32             <input type="email" name="email" required />
33             <input type="password" name="password" required />
34
35             <button type="submit" disabled={isLoading}>
36                 {isLoading ? 'Caricamento...' : 'Login'}
37             </button>
38         </form>
39     );
40 }
```

#### 1.2.4 4. Ecosistema Ricco e Community Attiva

React beneficia di un ecosistema vasto e maturo:

##### Librerie e Tool Popolari:

- **State Management:** Redux, MobX, Zustand, Jotai, Recoil
- **Routing:** React Router, TanStack Router, Wouter
- **Data Fetching:** TanStack Query, SWR, Apollo Client
- **Forms:** React Hook Form, Formik, Final Form

- **UI Components:** Material-UI, Ant Design, Chakra UI, Shadcn/ui
- **Animation:** Framer Motion, React Spring, GSAP
- **Testing:** React Testing Library, Jest, Vitest
- **Dev Tools:** React DevTools, Storybook, ESLint, Prettier
- **Frameworks:** Next.js, Remix, Gatsby
- **Mobile:** React Native, Expo

### 1.2.5 5. Unidirezionalità dei Dati

React implementa un flusso di dati unidirezionale (one-way data flow), che rende il comportamento dell'applicazione più prevedibile e facile da debuggare.

Listing 1.8: Flusso dati unidirezionale

```

1 // I dati fluiscono sempre dall'alto verso il basso
2 function App() {
3     const [user, setUser] = useState({
4         name: 'Mario Rossi',
5         role: 'Admin'
6     });
7
8     // App passa dati ai componenti figli
9     return (
10         <div>
11             <Header user={user} />
12             <Dashboard user={user} onUpdateUser={setUser} />
13         </div>
14     );
15 }
16
17 function Header({ user }) {
18     // Header riceve dati e li visualizza
19     // Non può modificarli direttamente
20     return (
21         <header>
22             <h1>Benvenuto, {user.name}</h1>
23             <span>Ruolo: {user.role}</span>
24         </header>
25     );
26 }
27
28 function Dashboard({ user, onUpdateUser }) {
29     // Dashboard può richiedere modifiche tramite callback
30     const handleRoleChange = () => {
31         onUpdateUser({
32             ...user,
33             role: user.role === 'Admin' ? 'User' : 'Admin'
34         });
35     };
36
37     return (
38         <div>
39             <p>Dashboard per {user.name}</p>
40             <button onClick={handleRoleChange}>
41                 Cambia Ruolo

```

```
42         </button>
43     </div>
44 );
45 }
```

### 1.2.6 6. Learn Once, Write Anywhere

Con React puoi costruire:

- **Web Apps:** Single Page Applications con React
- **Mobile Apps:** iOS e Android con React Native
- **Desktop Apps:** Con Electron + React
- **Static Sites:** Con Gatsby o Next.js
- **Server-Side Rendering:** Con Next.js o Remix
- **VR/AR:** Con React 360 o React Three Fiber

Listing 1.9: Stesso componente per Web e Mobile

```
1 // components/Button.js - Funziona sia su web che mobile
2 import { TouchableOpacity, Text } from 'react-native'; // Mobile
3 // import './Button.css'; // Web
4
5 function Button({ onPress, title, style }) {
6     // Logica condivisa
7     const handlePress = () => {
8         console.log('Button pressed');
9         onPress?.();
10    };
11
12    // React Native (Mobile)
13    return (
14        <TouchableOpacity
15            onPress={handlePress}
16            style={[styles.button, style]}
17        >
18            <Text style={styles.text}>{title}</Text>
19        </TouchableOpacity>
20    );
21
22    // React DOM (Web)
23    // return (
24    //     <button onClick={handlePress} className="button" style={style}
25    //         {title}
26    //     </button>
27    // );
28 }
```

## 1.3 Quando Usare React

### 1.3.1 React è Ideale Per:

#### 1. Single Page Applications (SPA):

Applicazioni web complesse dove l'utente naviga senza ricaricare la pagina.

Listing 1.10: SPA con routing

```
1 import { BrowserRouter, Routes, Route } from 'react-router-dom';
2
3 function App() {
4   return (
5     <BrowserRouter>
6       <Navigation />
7       <Routes>
8         <Route path="/" element={<Home />} />
9         <Route path="/products" element={<Products />} />
10        <Route path="/products/:id" element={<ProductDetail />} />
11        <Route path="/cart" element={<Cart />} />
12        <Route path="/checkout" element={<Checkout />} />
13      </Routes>
14    </BrowserRouter>
15  );
16 }
```

#### 2. Dashboard e Admin Panels:

Interfacce con molti componenti interattivi e aggiornamenti in tempo reale.

Listing 1.11: Dashboard interattiva

```
1 function AdminDashboard() {
2   const [stats, setStats] = useState(null);
3   const [liveUsers, setLiveUsers] = useState([]);
4
5   // Polling per aggiornamenti in tempo reale
6   useEffect(() => {
7     const interval = setInterval(async () => {
8       const data = await fetchLiveStats();
9       setStats(data.stats);
10      setLiveUsers(data.users);
11    }, 5000);
12
13    return () => clearInterval(interval);
14  }, []);
15
16  return (
17    <div className="dashboard">
18      <StatsCards stats={stats} />
19      <LiveUsersTable users={liveUsers} />
20      <SalesChart data={stats?.sales} />
21      <RecentOrders limit={10} />
22    </div>
23  );
24 }
```

#### 3. Applicazioni con UI Complesse:

Interfacce con molti stati e interazioni.

Listing 1.12: UI complessa con drag and drop

```

1 import { DndContext, closestCenter } from '@dnd-kit/core';
2
3 function KanbanBoard() {
4   const [columns, setColumns] = useState({
5     todo: ['Task 1', 'Task 2'],
6     inProgress: ['Task 3'],
7     done: ['Task 4', 'Task 5']
8   });
9
10  const handleDragEnd = (event) => {
11    const { active, over } = event;
12
13    if (!over) return;
14
15    // Logica per spostare task tra colonne
16    const sourceColumn = findColumn(active.id);
17    const destColumn = over.id;
18
19    moveTask(active.id, sourceColumn, destColumn);
20  };
21
22  return (
23    <DndContext onDragEnd={handleDragEnd} collisionDetection={
24      closestCenter}>
25      <div className="kanban">
26        <Column id="todo" title="To Do" tasks={columns.todo} />
27        <Column id="inProgress" title="In Progress" tasks={
28          columns.inProgress} />
29        <Column id="done" title="Done" tasks={columns.done} />
30      </div>
31    </DndContext>
  );
}

```

#### 4. Applicazioni Real-Time:

Chat, collaboration tools, live updates.

Listing 1.13: Chat real-time con WebSocket

```

1 function ChatRoom({ roomId }) {
2   const [messages, setMessages] = useState([]);
3   const [ws, setWs] = useState(null);
4
5   useEffect(() => {
6     const websocket = new WebSocket(`ws://api.example.com/chat/${
7       roomId}`);
8
9     websocket.onmessage = (event) => {
10       const message = JSON.parse(event.data);
11       setMessages(prev => [...prev, message]);
12     };
13
14     setWs(websocket);
15
16     return () => websocket.close();
17   }, [roomId]);
18
19   const sendMessage = (text) => {

```

```

19         ws.send(JSON.stringify({ text, roomId }));
20     };
21
22     return (
23         <div className="chat">
24             <MessageList messages={messages} />
25             <MessageInput onSend={sendMessage} />
26         </div>
27     );
28 }

```

### 1.3.2 Quando Considerare Alternative:

#### 1. Siti Statici Semplici:

Per siti con contenuto prevalentemente statico, HTML/CSS puri o generatori statici potrebbero essere più appropriati.

#### 2. SEO-Critical con Vincoli di Performance:

Per siti dove ogni millisecondo conta e il SEO è critico, soluzioni SSR/SSG pure potrebbero essere migliori (comunque disponibili con Next.js).

#### 3. Applicazioni Molto Semplici:

Per form semplici o piccole interazioni, vanilla JavaScript potrebbe essere sufficiente.

## 1.4 L'Ecosistema React nel 2024-2025

### 1.4.1 React Server Components

Una delle innovazioni più recenti che sta cambiando il modo di costruire applicazioni React:

Listing 1.14: Server Component esempio

```

1 // app/products/page.js - Server Component (Next.js 13+)
2 async function ProductsPage() {
3     // Questo codice gira SOLO sul server
4     // Non aumenta il bundle JavaScript del client
5     const products = await db.products.findMany();
6
7     return (
8         <div>
9             <h1>I Nostri Prodotti</h1>
10             <ProductList products={products} />
11         </div>
12     );
13 }
14
15 // components/ProductList.js - Client Component
16 'use client'; // Direttiva per Client Component
17
18 import { useState } from 'react';
19
20 function ProductList({ products }) {
21     const [filter, setFilter] = useState('all');
22
23     // Questo codice gira sul client e può usare interattività
24     return (
25         <div>
26             <FilterBar value={filter} onChange={setFilter} />
27             {products}

```

```
28         .filter(p => filter === 'all' || p.category === filter)
29         .map(product => (
30             <ProductCard key={product.id} product={product} />
31         ))
32     }
33 </div>
34 );
35 }
```

### 1.4.2 React Compiler (in arrivo)

Un compilatore che ottimizza automaticamente le performance del codice React, eliminando la necessità di `useMemo` e `useCallback` manuali.

### 1.4.3 Concurrent Features

React 18+ introduce rendering concorrente che migliora la responsiveness:

Listing 1.15: Concurrent Features

```
1 import { useTransition, useDeferredValue } from 'react';
2
3 function SearchPage() {
4     const [query, setQuery] = useState('');
5     const [isPending, startTransition] = useTransition();
6
7     // Gli aggiornamenti nella transition sono interrompibili
8     const handleChange = (e) => {
9         startTransition(() => {
10             setQuery(e.target.value);
11         });
12     };
13
14     return (
15         <div>
16             <input onChange={handleChange} />
17             {isPending && <Spinner />}
18             <SearchResults query={query} />
19         </div>
20     );
21 }
```

## 1.5 Prerequisiti per Questo Libro

### 1.5.1 Conoscenze Richieste

Per trarre il massimo da questo libro, dovresti avere:

- **JavaScript (ES6+)**: Arrow functions, destructuring, spread operator, modules
- **HTML/CSS**: Comprensione di base del DOM e dello styling
- **NPM/Yarn**: Package manager e gestione dipendenze
- **Concetti di Programmazione**: Funzioni, oggetti, array, asincronicità

**Ripasso JavaScript Essenziale:**



Listing 1.16: Concetti JavaScript per React

```
1 // 1. Arrow Functions
2 const greet = (name) => 'Hello, ${name}';
3 const add = (a, b) => a + b;
4
5 // 2. Destructuring
6 const user = { name: 'Mario', age: 30 };
7 const { name, age } = user;
8
9 const numbers = [1, 2, 3];
10 const [first, second] = numbers;
11
12 // 3. Spread Operator
13 const newUser = { ...user, city: 'Roma' };
14 const newNumbers = [...numbers, 4, 5];
15
16 // 4. Template Literals
17 const message = `${name} ha ${age} anni`;
18
19 // 5. Array Methods
20 const doubled = numbers.map(n => n * 2);
21 const evens = numbers.filter(n => n % 2 === 0);
22 const sum = numbers.reduce((acc, n) => acc + n, 0);
23
24 // 6. Async/Await
25 const fetchData = async () => {
26   try {
27     const response = await fetch('/api/data');
28     const data = await response.json();
29     return data;
30   } catch (error) {
31     console.error(error);
32   }
33 };
34
35 // 7. Modules
36 export const API_URL = 'https://api.example.com';
37 export default function Component() { /* ... */ }
38
39 // 8. Optional Chaining
40 const userName = user?.profile?.name ?? 'Guest';
41
42 // 9. Nullish Coalescing
43 const port = config.port ?? 3000;
```

## 1.5.2 Setup dell'Ambiente di Sviluppo

Avrai bisogno di:

- **Node.js** (v18 o superiore): Runtime JavaScript
- **Editor**: VS Code (raccomandato) con estensioni:
  - ES7+ React/Redux/React-Native snippets
  - ESLint
  - Prettier

– Auto Rename Tag

- **Browser:** Chrome o Firefox con React DevTools installato
- **Git:** Per version control

## 1.6 Struttura del Libro

Questo libro è organizzato in modo progressivo:

1. **Introduzione a React:** Virtual DOM, setup, primi passi
2. **JSX e Componenti:** Sintassi, functional components, composizione
3. **Props e State:** Gestione dati, props drilling, controlled components
4. **Hooks:** useState, useEffect, custom hooks e pattern avanzati
5. **Event Handling:** Gestione eventi e interazioni utente
6. **Form:** Form controllati, validazione, librerie

Ogni capitolo include:

- Spiegazioni teoriche dettagliate
- Esempi di codice completi e funzionanti
- Best practices e pattern comuni
- Esercizi pratici
- Diagrammi e visualizzazioni

## 1.7 Filosofia di Apprendimento

### 1.7.1 Imparare Facendo

Il modo migliore per imparare React è scrivere codice. Ogni esempio in questo libro è:

- **Funzionante:** Puoi copiarlo e provarlo subito
- **Incrementale:** Costruiamo concetti uno sopra l'altro
- **Pratico:** Esempi basati su casi d'uso reali
- **Commentato:** Spiegazioni inline nel codice

### 1.7.2 Best Practices dal Giorno Uno

Non imparerai solo come far funzionare le cose, ma come farle funzionare *bene*:

- Codice pulito e manutenibile
- Performance optimization
- Accessibilità (a11y)
- Testing
- Sicurezza

## 1.8 Oltre Questo Libro

### 1.8.1 Risorse per Continuare

Una volta completato questo libro, ecco dove proseguire:

**Documentazione Ufficiale:**

- <https://react.dev> - Nuova documentazione ufficiale
- <https://react.dev/learn> - Tutorial interattivi

**Framework e Tool:**

- Next.js per SSR e SSG
- Remix per web apps full-stack
- React Native per mobile

**Community:**

- Reddit: [r/reactjs](#)
- Discord: [Reactiflux](#)
- Twitter: [#ReactJS](#)
- Stack Overflow: tag [\[reactjs\]](#)

### 1.8.2 Il Futuro di React

React continua ad evolversi rapidamente:

- **React Forget:** Compilatore automatico per ottimizzazioni
- **Server Components:** Architettura ibrida client-server
- **Concurrent Features:** Rendering più fluido e responsivo
- **Suspense for Data:** Loading states dichiarativi
- **Actions:** Gestione form e mutations migliorata

## 1.9 Un Ultimo Pensiero

React non è solo una libreria da imparare: è una nuova mentalità per pensare alle interfacce utente. I concetti che apprenderai - componenti, immutabilità, composizione, flusso dati unidirezionale - ti serviranno indipendentemente dalle tecnologie che userai in futuro.

Non ti preoccupare se all'inizio qualcosa sembra complesso o contro-intuitivo. React ha una curva di apprendimento, ma una volta che "fa click", diventa uno strumento incredibilmente potente e piacevole da usare.

**Il viaggio inizia ora. Buon divertimento con React!**

Listing 1.17: Il tuo primo componente React ti aspetta...

```
1 import { useState } from 'react';
2
3 function Welcome() {
4   const [started, setStarted] = useState(false);
5
6   return (
7     <div className="welcome">
8       <h1>Benvenuto nel Mondo di React!</h1>
9       {!started ? (
10         <button onClick={() => setStarted(true)}>
11           Inizia il Viaggio
12         </button>
13       ) : (
14         <p>Sei pronto a costruire cose incredibili! </p>
15       )}
16     </div>
17   );
18 }
19
20 export default Welcome;
```

## Capitolo 2

# Introduzione a React

### 2.1 Cos'è React?

React è una libreria JavaScript dichiarativa, efficiente e flessibile per costruire interfacce utente. A differenza di framework completi come Angular, React si concentra esclusivamente sul layer di visualizzazione, lasciando agli sviluppatori la libertà di scegliere le soluzioni migliori per routing, state management e altre funzionalità.

#### 2.1.1 Caratteristiche Principali

##### 1. Libreria, Non Framework

React è una libreria focalizzata sulla UI, non un framework completo:

Listing 2.1: React fornisce solo la UI

```
1 // React fornisce
2 import { useState } from 'react';
3
4 function App() {
5   return <div>Hello React</div>;
6 }
7
8 // Tu scegli il resto:
9 // - Routing: React Router, TanStack Router, etc.
10 // - State: Redux, Zustand, MobX, etc.
11 // - Data fetching: TanStack Query, SWR, etc.
12 // - Forms: React Hook Form, Formik, etc.
```

##### 2. Component-Based

Tutto in React è un componente riutilizzabile:

Listing 2.2: Componenti come building blocks

```
1 // Componente Button
2 function Button({ children, onClick }) {
3   return <button onClick={onClick}>{children}</button>;
4 }
5
6 // Componente Form che usa Button
7 function LoginForm() {
8   return (
9     <form>
10       <input type="email" />
11       <input type="password" />
12       <Button onClick={() => console.log('Login')}>
```

```

13         Accedi
14       </Button>
15     </form>
16   );
17 }

```

### 3. Dichiarativo

Descrivi cosa vuoi vedere, React si occupa del resto:

Listing 2.3: Codice dichiarativo

```

1 function Counter() {
2   const [count, setCount] = useState(0);
3
4   // Dichiaro COSA mostrare in base allo stato
5   return (
6     <div>
7       <p>Hai cliccato {count} volte</p>
8       <button onClick={() => setCount(count + 1)}>
9         Incrementa
10      </button>
11    </div>
12  );
13  // React aggiorna il DOM automaticamente quando count cambia
14 }

```

## 2.2 Il Virtual DOM

Il Virtual DOM è uno dei concetti fondamentali che rendono React efficiente e performante.

### 2.2.1 Cos'è il Virtual DOM?

Il Virtual DOM è una rappresentazione leggera del DOM reale mantenuta in memoria come oggetto JavaScript. Invece di manipolare direttamente il DOM (operazione costosa), React lavora con questa rappresentazione virtuale.

**Struttura del Virtual DOM:**

Listing 2.4: Rappresentazione Virtual DOM

```

1 // JSX
2 <div className="container">
3   <h1>Titolo</h1>
4   <p>Paragrafo</p>
5 </div>
6
7 // Viene trasformato in Virtual DOM object
8 {
9   type: 'div',
10  props: {
11    className: 'container',
12    children: [
13      {
14        type: 'h1',
15        props: {
16          children: 'Titolo'
17        }
18      },
19      {

```

```

20         type: 'p',
21         props: {
22             children: 'Paragrafo'
23         }
24     }
25 ]
26 }
27 }

```

### 2.2.2 Come Funziona il Virtual DOM

Il processo di rendering in React segue questi passaggi:

#### Fase 1: Render Iniziale

Listing 2.5: Primo rendering

```

1 function App() {
2     const [items, setItems] = useState(['A', 'B', 'C']);
3
4     return (
5         <ul>
6             {items.map(item => <li key={item}>{item}</li>)}
7         </ul>
8     );
9 }
10
11 // React crea Virtual DOM:
12 {
13     type: 'ul',
14     props: {
15         children: [
16             { type: 'li', props: { children: 'A', key: 'A' } },
17             { type: 'li', props: { children: 'B', key: 'B' } },
18             { type: 'li', props: { children: 'C', key: 'C' } }
19         ]
20     }
21 }
22
23 // React crea il DOM reale basandosi sul Virtual DOM

```

#### Fase 2: Update

Listing 2.6: Aggiornamento stato

```

1 // Quando lo stato cambia (aggiungiamo 'D')
2 setItems(['A', 'B', 'C', 'D']);
3
4 // React crea un NUOVO Virtual DOM tree:
5 {
6     type: 'ul',
7     props: {
8         children: [
9             { type: 'li', props: { children: 'A', key: 'A' } },
10            { type: 'li', props: { children: 'B', key: 'B' } },
11            { type: 'li', props: { children: 'C', key: 'C' } },
12            { type: 'li', props: { children: 'D', key: 'D' } } // NUOVO
13        ]
14    }
15 }

```

**Fase 3: Diffing**

React confronta il nuovo Virtual DOM con quello precedente:

Listing 2.7: Algoritmo di Diffing

```

1 // Pseudo-codice del diffing
2 function diff(oldVTree, newVTree) {
3   // Confronta tipo
4   if (oldVTree.type !== newVTree.type) {
5     return { action: 'REPLACE', node: newVTree };
6   }
7
8   // Confronta props
9   const propsDiff = diffProps(oldVTree.props, newVTree.props);
10
11  // Confronta children ricorsivamente
12  const childrenDiff = diffChildren(
13    oldVTree.props.children,
14    newVTree.props.children
15  );
16
17  return {
18    action: 'UPDATE',
19    propChanges: propsDiff,
20    childChanges: childrenDiff
21  };
22 }
23
24 // Nel nostro esempio, React identifica:
25 // - 3 elementi uguali (A, B, C)
26 // - 1 elemento nuovo (D) da aggiungere

```

**Fase 4: Reconciliation**

React applica solo le modifiche necessarie al DOM reale:

Listing 2.8: Applicazione modifiche

```

1 // Invece di ricreare tutta la lista:
2 //   ul.innerHTML = '<li>A</li><li>B</li><li>C</li><li>D</li>';
3
4 // React fa solo:
5 //   ul.appendChild(document.createElement('li')); // Aggiunge solo D

```

**2.2.3 Diagramma del Processo Virtual DOM**

## VIRTUAL DOM WORKFLOW

```

1. STATE CHANGE
   > setItems(['A', 'B', 'C', 'D'])

```

```

2. NEW VIRTUAL DOM TREE CREATED

```

```

Virtual DOM v2
[A, B, C, D]

```



## 3. DIFFING ALGORITHM

Virtual DOM v1	vs	Virtual DOM v2
[A, B, C]		[A, B, C, D]

> Identifica: "Aggiungere D alla fine"

## 4. RECONCILIATION

Applica SOLO le modifiche minime al DOM reale  
 > appendChild(<li>D</li>)

## 2.2.4 Vantaggi del Virtual DOM

## 1. Performance

Listing 2.9: Confronto performance

```

1 //      Manipolazione diretta del DOM (LENTA)
2 function updateList(items) {
3   const ul = document.getElementById('list');
4   ul.innerHTML = ''; // Rimuove tutto
5
6   items.forEach(item => {
7     const li = document.createElement('li');
8     li.textContent = item;
9     ul.appendChild(li); // Ogni append causa reflow
10  });
11 }
12
13 //      Con React Virtual DOM (VELOCE)
14 function List({ items }) {
15   return (
16     <ul>
17       {items.map(item => <li key={item}>{item}</li>)}
18     </ul>
19   );
20   // React aggiorna solo gli elementi modificati
21 }

```

## 2. Batch Updates

React raggruppa più aggiornamenti:

Listing 2.10: Batching automatico

```

1 function MultiUpdate() {
2   const [count, setCount] = useState(0);
3   const [flag, setFlag] = useState(false);
4
5   const handleClick = () => {
6     setCount(count + 1); // Update 1
7     setFlag(!flag);      // Update 2
8     // React raggruppa entrambi e fa un solo re-render
9   };

```

```

10
11     console.log('Render!'); // Viene chiamato UNA sola volta
12
13     return (
14         <div>
15             <p>Count: {count}</p>
16             <p>Flag: {flag.toString()}</p>
17             <button onClick={handleClick}>Update</button>
18         </div>
19     );
20 }

```

### 3. Prevedibilità

Il rendering diventa deterministico:

Listing 2.11: Rendering prevedibile

```

1 // Stesso stato = stesso output SEMPRE
2 function UserCard({ user }) {
3     return (
4         <div>
5             <h3>{user.name}</h3>
6             <p>{user.email}</p>
7         </div>
8     );
9 }
10
11 // Se user è identico, output identico
12 // Facile da testare e debuggare

```

## 2.3 React vs Altri Framework

### 2.3.1 React vs Angular

#### Angular

Listing 2.12: Componente Angular

```

1 // user.component.ts
2 import { Component, Input } from '@angular/core';
3
4 @Component({
5     selector: 'app-user',
6     template: `
7         <div class="user-card">
8             <h3>{{ user.name }}</h3>
9             <p>{{ user.email }}</p>
10            <button (click)="onEdit()">Edit</button>
11        </div>
12    `,
13     styleUrls: ['./user.component.css']
14 })
15 export class UserComponent {
16     @Input() user: User;
17
18     onEdit() {
19         console.log('Editing', this.user);
20     }
21 }

```

## React

Listing 2.13: Componente React equivalente

```

1 // UserCard.jsx
2 import './UserCard.css';
3
4 function UserCard({ user }) {
5   const handleEdit = () => {
6     console.log('Editing', user);
7   };
8
9   return (
10     <div className="user-card">
11       <h3>{user.name}</h3>
12       <p>{user.email}</p>
13       <button onClick={handleEdit}>Edit</button>
14     </div>
15   );
16 }
17
18 export default UserCard;

```

### Confronto:

Aspetto	React	Angular
Tipo	Libreria	Framework completo
Linguaggio	JavaScript/TypeScript	TypeScript
Learning Curve	Media	Ripida
Bundle Size	~45KB	~150KB
Data Binding	Unidirezionale	Bidirezionale
Dependency Injection	No (context/props)	Sì (built-in)
Routing	Libreria esterna	Built-in
State Management	Libreria esterna	RxJS/Services
Template	JSX	HTML + direttive
Mobile	React Native	Ionic/NativeScript

### 2.3.2 React vs Vue

#### Vue

Listing 2.14: Componente Vue

```

1 <!-- UserCard.vue -->
2 <template>
3   <div class="user-card">
4     <h3>{{ user.name }}</h3>
5     <p>{{ user.email }}</p>
6     <button @click="handleEdit">Edit</button>
7   </div>
8 </template>
9
10 <script setup>
11 import { defineProps } from 'vue';
12
13 const props = defineProps({
14   user: Object
15 });
16

```

```

17  const handleEdit = () => {
18      console.log('Editing', props.user);
19  };
20  </script>
21
22  <style scoped>
23  .user-card {
24      border: 1px solid #ccc;
25  }
26  </style>

```

## React

Listing 2.15: Componente React equivalente

```

1  // UserCard.jsx
2  import styles from './UserCard.module.css';
3
4  function UserCard({ user }) {
5      const handleEdit = () => {
6          console.log('Editing', user);
7      };
8
9      return (
10         <div className={styles.userCard}>
11             <h3>{user.name}</h3>
12             <p>{user.email}</p>
13             <button onClick={handleEdit}>Edit</button>
14         </div>
15     );
16 }
17
18 export default UserCard;

```

## Confronto:

Aspetto	React	Vue
Tipo	Libreria	Framework progressivo
Learning Curve	Media	Facile
Template	JSX	HTML-based
Reattività	useState/useReducer	Proxy-based
Community	Molto grande	Grande
Job Market	Ampio	Buono
Ecosistema	Vasto	Integrato
Performance	Eccellente	Eccellente

### 2.3.3 Quando Scegliere React

#### Scegli React se:

- Vuoi massima flessibilità nella scelta delle librerie
- Hai bisogno di un'ottima integrazione con TypeScript
- Il mercato del lavoro è un fattore importante
- Vuoi costruire anche app mobile (React Native)
- Preferisci JavaScript/JSX ai template HTML

- Hai bisogno di un ecosistema vasto

**Scegli Vue se:**

- Vuoi una curva di apprendimento più dolce
- Preferisci template HTML con direttive
- Vuoi un framework più "opinionated" ma flessibile
- Apprezzi la documentazione eccellente

**Scegli Angular se:**

- Lavori su applicazioni enterprise di grandi dimensioni
- Hai bisogno di TypeScript robusto
- Vuoi un framework completo "tutto incluso"
- Apprezzi structure e convention rigide

## 2.4 Setup dell'Ambiente di Sviluppo

### 2.4.1 Prerequisiti

Prima di iniziare, assicurati di avere installato:

Listing 2.16: Verifica installazioni

```
1 # Node.js (versione 18 o superiore)
2 node --version
3 # Output: v20.10.0
4
5 # npm (viene installato con Node.js)
6 npm --version
7 # Output: 10.2.3
8
9 # Git (opzionale ma raccomandato)
10 git --version
11 # Output: git version 2.42.0
```

### 2.4.2 Opzione 1: Vite (Raccomandato 2024)

Vite è il build tool moderno più veloce per progetti React:

Listing 2.17: Creare progetto con Vite

```
1 # Crea nuovo progetto
2 npm create vite@latest my-react-app -- --template react
3
4 # Entra nella directory
5 cd my-react-app
6
7 # Installa dipendenze
8 npm install
9
10 # Avvia development server
11 npm run dev
12 # Server running at http://localhost:5173
```

**Struttura progetto Vite:**

```
my-react-app/  
  node_modules/  
  public/  
    vite.svg  
  src/  
    assets/  
    App.css  
    App.jsx  
    index.css  
    main.jsx  
  .gitignore  
  index.html  
  package.json  
  vite.config.js
```

**File principali:**

Listing 2.18: main.jsx - Entry point

```
1 import React from 'react'  
2 import ReactDOM from 'react-dom/client'  
3 import App from './App.jsx'  
4 import './index.css'  
5  
6 ReactDOM.createRoot(document.getElementById('root')).render(  
7   <React.StrictMode>  
8     <App />  
9   </React.StrictMode>,  
10 )
```

Listing 2.19: App.jsx - Componente principale

```
1 import { useState } from 'react'  
2 import './App.css'  
3  
4 function App() {  
5   const [count, setCount] = useState(0)  
6  
7   return (  
8     <div className="App">  
9       <h1>Vite + React</h1>  
10      <div className="card">  
11        <button onClick={() => setCount((count) => count + 1)}>  
12          count is {count}  
13        </button>  
14      </div>  
15    </div>  
16  )  
17 }  
18  
19 export default App
```

Listing 2.20: vite.config.js - Configurazione

```
1 import { defineConfig } from 'vite'  
2 import react from '@vitejs/plugin-react'
```

```
3
4 export default defineConfig({
5   plugins: [react()],
6   server: {
7     port: 3000, // Cambia porta se necessario
8     open: true  // Apri browser automaticamente
9   }
10 })
```

#### Vantaggi di Vite:

- **Velocissimo:** HMR (Hot Module Replacement) istantaneo
- **Leggero:** Build ottimizzate con Rollup
- **Moderno:** ESM nativo, supporto TypeScript out-of-the-box
- **Semplice:** Configurazione minimale

### 2.4.3 Opzione 2: Create React App (Legacy)

Create React App (CRA) è stato lo standard per anni, ma ora è meno raccomandato:

Listing 2.21: Creare progetto con CRA

```
1 # Crea nuovo progetto
2 npx create-react-app my-app
3
4 # Entra nella directory
5 cd my-app
6
7 # Avvia development server
8 npm start
9 # Server running at http://localhost:3000
```

**Nota:** Create React App è ora in maintenance mode. La documentazione ufficiale React raccomanda Vite o framework come Next.js.

### 2.4.4 Opzione 3: Setup Manuale

Per comprendere meglio come funziona:

Listing 2.22: Setup manuale da zero

```
1 # Crea directory e inizializza progetto
2 mkdir my-react-app
3 cd my-react-app
4 npm init -y
5
6 # Installa React e ReactDOM
7 npm install react react-dom
8
9 # Installa Vite come dev dependency
10 npm install -D vite @vitejs/plugin-react
11
12 # Crea struttura
13 mkdir src public
14 touch src/main.jsx src/App.jsx index.html vite.config.js
```

Listing 2.23: index.html

```
1 <!DOCTYPE html>
2 <html lang="it">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0
6     ">
7   <title>My React App</title>
8 </head>
9 <body>
10   <div id="root"></div>
11   <script type="module" src="/src/main.jsx"></script>
12 </body>
13 </html>
```

Listing 2.24: package.json - Aggiungi scripts

```
1 {
2   "name": "my-react-app",
3   "version": "1.0.0",
4   "scripts": {
5     "dev": "vite",
6     "build": "vite build",
7     "preview": "vite preview"
8   },
9   "dependencies": {
10     "react": "^18.2.0",
11     "react-dom": "^18.2.0"
12   },
13   "devDependencies": {
14     "vite": "^5.0.0",
15     "@vitejs/plugin-react": "^4.2.0"
16   }
17 }
```

### 2.4.5 Opzione 4: Framework Full-Stack

Per applicazioni più complesse, considera framework completi:

**Next.js (Raccomandato per produzione):**

Listing 2.25: Setup Next.js

```
1 npx create-next-app@latest my-next-app
2
3 # Scegli opzioni:
4 #   TypeScript? No/Yes
5 #   ESLint? Yes
6 #   Tailwind CSS? No/Yes
7 #   App Router? Yes (raccomandato)
8 #   Customize import alias? No
9
10 cd my-next-app
11 npm run dev
```

**Remix:**

Listing 2.26: Setup Remix

```
1 npx create-remix@latest my-remix-app
```



```
2 cd my-remix-app
3 npm run dev
```

## 2.5 Il Primo Progetto React

Creiamo una semplice applicazione TODO per esplorare i concetti base:

Listing 2.27: Setup progetto TODO

```
1 npm create vite@latest react-todo -- --template react
2 cd react-todo
3 npm install
4 npm run dev
```

### 2.5.1 App.jsx - TODO List Completa

Listing 2.28: Applicazione TODO completa

```
1 import { useState } from 'react';
2 import './App.css';
3
4 function App() {
5   const [todos, setTodos] = useState([
6     { id: 1, text: 'Imparare React', completed: false },
7     { id: 2, text: 'Costruire un progetto', completed: false }
8   ]);
9   const [inputValue, setInputValue] = useState('');
10
11   // Aggiungi nuovo todo
12   const addTodo = () => {
13     if (inputValue.trim() === '') return;
14
15     const newTodo = {
16       id: Date.now(),
17       text: inputValue,
18       completed: false
19     };
20
21     setTodos([...todos, newTodo]);
22     setInputValue('');
23   };
24
25   // Toggle completamento
26   const toggleTodo = (id) => {
27     setTodos(todos.map(todo =>
28       todo.id === id
29         ? { ...todo, completed: !todo.completed }
30         : todo
31     ));
32   };
33
34   // Elimina todo
35   const deleteTodo = (id) => {
36     setTodos(todos.filter(todo => todo.id !== id));
37   };
38
39   // Gestisci Enter key
```

```

40   const handleKeyPress = (e) => {
41     if (e.key === 'Enter') {
42       addTodo();
43     }
44   };
45
46   // Statistiche
47   const totalTodos = todos.length;
48   const completedTodos = todos.filter(t => t.completed).length;
49   const pendingTodos = totalTodos - completedTodos;
50
51   return (
52     <div className="app">
53       <div className="container">
54         <h1>      My TODO List</h1>
55
56         {/* Statistiche */}
57         <div className="stats">
58           <span>Totale: {totalTodos}</span>
59           <span>Completati: {completedTodos}</span>
60           <span>Da fare: {pendingTodos}</span>
61         </div>
62
63         {/* Input nuovo todo */}
64         <div className="input-section">
65           <input
66             type="text"
67             value={inputValue}
68             onChange={(e) => setInputValue(e.target.value)}
69             onKeyPress={handleKeyPress}
70             placeholder="Aggiungi un nuovo task..."
71           />
72           <button onClick={addTodo}>Aggiungi</button>
73         </div>
74
75         {/* Lista todos */}
76         <ul className="todo-list">
77           {todos.length === 0 ? (
78             <li className="empty">Nessun task. Aggiungine
79               uno!</li>
80           ) : (
81             todos.map(todo => (
82               <li
83                 key={todo.id}
84                 className={todo.completed ? 'completed' : ''}
85               >
86                 <input
87                   type="checkbox"
88                   checked={todo.completed}
89                   onChange={() => toggleTodo(todo.id)}
90                 />
91                 <span onClick={() => toggleTodo(todo.id)}>
92                   {todo.text}
93                 </span>
94                 <button
95                   className="delete-btn"

```

```

95         onClick={() => deleteTodo(todo.id)}
96     >
97
98     </button>
99 </li>
100 ))
101 }}
102 </ul>
103
104 {/* Clear completed */}
105 {completedTodos > 0 && (
106     <button
107         className="clear-completed"
108         onClick={() => setTodos(todos.filter(t => !t.
109             completed))}
110     >
111         Rimuovi completati ({completedTodos})
112     </button>
113 )}
114 </div>
115 );
116 }
117
118 export default App;

```

## 2.5.2 App.css - Styling

Listing 2.29: Stili per TODO app

```

1  .app {
2      min-height: 100vh;
3      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
4      display: flex;
5      justify-content: center;
6      align-items: center;
7      padding: 20px;
8  }
9
10 .container {
11     background: white;
12     border-radius: 15px;
13     padding: 30px;
14     max-width: 500px;
15     width: 100%;
16     box-shadow: 0 10px 40px rgba(0,0,0,0.2);
17 }
18
19 h1 {
20     text-align: center;
21     color: #333;
22     margin-bottom: 20px;
23 }
24
25 .stats {
26     display: flex;
27     justify-content: space-around;

```

```
28     margin-bottom: 20px;
29     padding: 15px;
30     background: #f5f5f5;
31     border-radius: 8px;
32 }
33
34 .stats span {
35     font-size: 14px;
36     color: #666;
37 }
38
39 .input-section {
40     display: flex;
41     gap: 10px;
42     margin-bottom: 20px;
43 }
44
45 .input-section input {
46     flex: 1;
47     padding: 12px;
48     border: 2px solid #e0e0e0;
49     border-radius: 8px;
50     font-size: 16px;
51 }
52
53 .input-section input:focus {
54     outline: none;
55     border-color: #667eea;
56 }
57
58 .input-section button {
59     padding: 12px 24px;
60     background: #667eea;
61     color: white;
62     border: none;
63     border-radius: 8px;
64     cursor: pointer;
65     font-weight: 600;
66 }
67
68 .input-section button:hover {
69     background: #5568d3;
70 }
71
72 .todo-list {
73     list-style: none;
74     padding: 0;
75     margin: 0;
76 }
77
78 .todo-list li {
79     display: flex;
80     align-items: center;
81     padding: 15px;
82     border-bottom: 1px solid #e0e0e0;
83     gap: 10px;
84 }
85
```

```
86 .todo-list li.completed span {
87     text-decoration: line-through;
88     color: #999;
89 }
90
91 .todo-list li span {
92     flex: 1;
93     cursor: pointer;
94 }
95
96 .todo-list li input[type="checkbox"] {
97     width: 20px;
98     height: 20px;
99     cursor: pointer;
100 }
101
102 .delete-btn {
103     background: none;
104     border: none;
105     cursor: pointer;
106     font-size: 20px;
107     opacity: 0.6;
108     transition: opacity 0.2s;
109 }
110
111 .delete-btn:hover {
112     opacity: 1;
113 }
114
115 .clear-completed {
116     width: 100%;
117     padding: 12px;
118     margin-top: 15px;
119     background: #f44336;
120     color: white;
121     border: none;
122     border-radius: 8px;
123     cursor: pointer;
124     font-weight: 600;
125 }
126
127 .clear-completed:hover {
128     background: #da190b;
129 }
130
131 .empty {
132     text-align: center;
133     color: #999;
134     padding: 40px;
135     font-style: italic;
136 }
```

## 2.6 React Developer Tools

### 2.6.1 Installazione

React DevTools è un'estensione essenziale per il debugging:

- **Chrome:** Chrome Web Store → "React Developer Tools"
- **Firefox:** Firefox Add-ons → "React DevTools"
- **Edge:** Edge Add-ons → "React Developer Tools"

## 2.6.2 Funzionalità Principali

### 1. Components Tab

Visualizza l'albero dei componenti e le loro props/state:

Components

App

  TodoList

    props: { todos: [...] }

    state: { filter: 'all' }

  TodoItem (x3)

    props: { todo: {...}, onToggle: f }

### 2. Profiler Tab

Analizza le performance dei rendering:

Listing 2.30: Usare il Profiler

```

1 import { Profiler } from 'react';
2
3 function App() {
4   const onRenderCallback = (
5     id,
6     phase,
7     actualDuration,
8     baseDuration,
9     startTime,
10    commitTime
11  ) => {
12    console.log(`${id} (${phase}) took ${actualDuration}ms`);
13  };
14
15  return (
16    <Profiler id="App" onRender={onRenderCallback}>
17      <TodoList />
18    </Profiler>
19  );
20 }
```

## 2.7 Best Practices Iniziali

### 2.7.1 1. Organizzazione File

src/

  components/

    TodoList/

      TodoList.jsx

      TodoList.css

      index.js

    TodoItem/

```
    TodoItem.jsx
    TodoItem.css
    index.js
hooks/
  useTodos.js
utils/
  helpers.js
App.jsx
main.jsx
```

## 2.7.2 2. Naming Convention

Listing 2.31: Convenzioni di naming

```
1 // Componenti: PascalCase
2 function TodoList() {}
3 function UserProfile() {}
4
5 // File componenti: stesso nome del componente
6 // TodoList.jsx, UserProfile.jsx
7
8 // Funzioni/variabili: camelCase
9 const handleClick = () => {};
10 const userData = {};
11
12 // Costanti: UPPER_SNAKE_CASE
13 const API_URL = 'https://api.example.com';
14 const MAX_ITEMS = 100;
15
16 // Custom hooks: use + PascalCase
17 function useTodos() {}
18 function useLocalStorage() {}
```

## 2.7.3 3. Componenti Puliti

Listing 2.32: Componente ben strutturato

```
1 import { useState, useEffect } from 'react';
2 import PropTypes from 'prop-types';
3 import './TodoList.css';
4
5 // 1. Imports
6
7 // 2. Costanti
8 const FILTER_OPTIONS = ['all', 'active', 'completed'];
9
10 // 3. Componente
11 function TodoList({ initialTodos = [] }) {
12   // 3.1 State
13   const [todos, setTodos] = useState(initialTodos);
14   const [filter, setFilter] = useState('all');
15
16   // 3.2 Effects
17   useEffect(() => {
18     console.log('Todos changed:', todos);
19   }, [todos]);
```

```
20
21 // 3.3 Handlers
22 const handleAddTodo = (text) => {
23   // ...
24 };
25
26 // 3.4 Computed values
27 const filteredTodos = todos.filter(todo => {
28   if (filter === 'active') return !todo.completed;
29   if (filter === 'completed') return todo.completed;
30   return true;
31 });
32
33 // 3.5 Render
34 return (
35   <div className="todo-list">
36     {/* JSX */}
37   </div>
38 );
39 }
40
41 // 4. PropTypes
42 TodoList.propTypes = {
43   initialTodos: PropTypes.arrayOf(PropTypes.shape({
44     id: PropTypes.number.isRequired,
45     text: PropTypes.string.isRequired,
46     completed: PropTypes.bool.isRequired
47   })),
48 };
49
50 // 5. Export
51 export default TodoList;
```

## 2.8 Conclusione

In questo capitolo abbiamo esplorato:

- Cos'è React e le sue caratteristiche principali
- Il funzionamento del Virtual DOM
- Confronto con Angular e Vue
- Setup dell'ambiente con Vite
- Creazione del primo progetto React
- React DevTools e best practices

Ora sei pronto per approfondire JSX e i componenti nel prossimo capitolo!



## Capitolo 3

# JSX e Componenti

### 3.1 Introduzione a JSX

JSX (JavaScript XML) è un'estensione della sintassi JavaScript che permette di scrivere markup simile a HTML all'interno di JavaScript. È una delle caratteristiche più distintive di React.

#### 3.1.1 Cos'è JSX?

JSX sembra HTML, ma è JavaScript:

Listing 3.1: Esempio JSX base

```
1 // Questo è JSX
2 const element = <h1>Hello, World!</h1>;
3
4 // Viene trasformato in JavaScript puro:
5 const element = React.createElement('h1', null, 'Hello, World!');
```

**JSX NON è:**

- HTML (anche se somiglia)
- Un template engine
- Obbligatorio (ma altamente raccomandato)

**JSX È:**

- JavaScript con sintassi estesa
- Type-safe (con TypeScript)
- Più espressivo e leggibile

#### 3.1.2 Sintassi JSX Fondamentale

##### 1. Espressioni JavaScript in JSX

Usa le parentesi graffe {} per inserire espressioni JavaScript:

Listing 3.2: Espressioni in JSX

```
1 function Greeting() {
2   const name = 'Mario';
3   const age = 25;
4   const hobbies = ['coding', 'reading', 'gaming'];
5 }
```

```

6   return (
7     <div>
8       {/* Variabili */}
9       <h1>Ciao, {name}!</h1>
10
11      {/* Operazioni */}
12      <p>Tra 5 anni avrai {age + 5} anni</p>
13
14      {/* Condizioni ternarie */}
15      <p>Sei {age} >= 18 ? 'maggiorenne' : 'minorenne'</p>
16
17      {/* Chiamate a funzioni */}
18      <p>Nome uppercase: {name.toUpperCase()}</p>
19
20      {/* Map per liste */}
21      <ul>
22        {hobbies.map((hobby, index) => (
23          <li key={index}>{hobby}</li>
24        ))}
25      </ul>
26
27      {/* Template literals */}
28      <p>{'${name} ha ${age} anni'}</p>
29    </div>
30  );
31 }

```

## 2. Attributi in JSX

Gli attributi seguono la convenzione camelCase:

Listing 3.3: Attributi JSX

```

1  function AttributesExample() {
2    const imageUrl = 'https://example.com/image.jpg';
3    const altText = 'Una bella immagine';
4    const isActive = true;
5
6    return (
7      <div>
8        {/* className invece di class */}
9        <div className="container"></div>
10
11       {/* htmlFor invece di for */}
12       <label htmlFor="input-id">Nome:</label>
13       <input id="input-id" type="text" />
14
15       {/* Attributi booleani */}
16       <input type="checkbox" checked={isActive} />
17       <input type="text" disabled />
18       <button type="submit" autoFocus>Invia</button>
19
20       {/* onClick invece di onclick */}
21       <button onClick={() => console.log('Click!')}>
22         Click me
23       </button>
24
25       {/* Style come oggetto */}
26       <div style={{
27         color: 'red',

```

```

28         fontSize: '20px',
29         backgroundColor: '#f0f0f0'
30     }}>
31     Testo stilizzato
32 </div>
33
34     {/* Attributi dinamici */}
35     <img src={imageUrl} alt={altText} />
36
37     {/* Data attributes */}
38     <div data-user-id="123" data-role="admin">
39         User info
40     </div>
41
42     {/* Spread attributes */}
43     <input {...{type: 'text', placeholder: 'Nome'}} />
44 </div>
45 );
46 }

```

### 3. Differenze HTML vs JSX

HTML	JSX	Motivo
class	className	class è keyword JS
for	htmlFor	for è keyword JS
onclick	onClick	camelCase convention
tabindex	tabIndex	camelCase convention
style="..."	style={{}}	Oggetto JS
<!-- -->	{/* */}	Commenti JS
 	 	Tag auto-chiudenti

### 4. Rendering Condizionale

Listing 3.4: Condizioni in JSX

```

1 function ConditionalRendering({ isLoggedIn, username, role }) {
2     // 1. If-else con ternario
3     return (
4         <div>
5             {isLoggedIn ? (
6                 <h1>Benvenuto, {username}!</h1>
7             ) : (
8                 <h1>Per favore, effettua il login</h1>
9             )}
10        </div>
11    );
12
13    // 2. Logical AND (&&) per rendering condizionale
14    return (
15        <div>
16            {isLoggedIn && <h1>Benvenuto, {username}!</h1>}
17            {role === 'admin' && <button>Pannello Admin</button>}
18        </div>
19    );
20
21    // 3. Variabile intermedia
22    let content;
23    if (isLoggedIn) {

```

```

24     content = <h1>Benvenuto, {username}!</h1>;
25   } else {
26     content = <h1>Per favore, effettua il login</h1>;
27   }
28   return <div>{content}</div>;
29
30   // 4. Early return
31   if (!isLoggedIn) {
32     return <LoginForm />;
33   }
34   return <Dashboard username={username} />;
35
36   // 5. Switch con oggetto
37   const statusMessages = {
38     loading: <Spinner />,
39     error: <ErrorMessage />,
40     success: <SuccessMessage />,
41     idle: null
42   };
43   return statusMessages[status];
44 }

```

## 5. Liste e Keys

Listing 3.5: Rendering liste

```

1  function ListRendering() {
2    const users = [
3      { id: 1, name: 'Mario', age: 25 },
4      { id: 2, name: 'Laura', age: 30 },
5      { id: 3, name: 'Giuseppe', age: 28 }
6    ];
7
8    return (
9      <div>
10        /*      Con key unica (id) */
11        <ul>
12          {users.map(user => (
13            <li key={user.id}>
14              {user.name} - {user.age} anni
15            </li>
16          ))}
17        </ul>
18
19        /*      Senza key (warning in console) */
20        <ul>
21          {users.map(user => (
22            <li>{user.name}</li>
23          ))}
24        </ul>
25
26        /*      Con index come key (evitare se possibile) */
27        <ul>
28          {users.map((user, index) => (
29            <li key={index}>{user.name}</li>
30          ))}
31        </ul>
32
33        /*      Liste complesse */

```

```

34         <div>
35             {users
36                 .filter(user => user.age >= 28)
37                 .sort((a, b) => a.name.localeCompare(b.name))
38                 .map(user => (
39                     <UserCard key={user.id} user={user} />
40                 ))
41             }
42         </div>
43     </div>
44 );
45 }

```

### Perché le Keys sono Importanti:

Listing 3.6: Importanza delle keys

```

1  // Senza keys corrette, React può confondersi
2  // nei riordinamenti
3
4  function TodoList() {
5      const [todos, setTodos] = useState([
6          { id: 1, text: 'Primo' },
7          { id: 2, text: 'Secondo' },
8          { id: 3, text: 'Terzo' }
9      ]);
10
11     // Se riordini e usi index come key
12     // React potrebbe aggiornare l'elemento sbagliato
13
14     // Male - usa index
15     return (
16         <ul>
17             {todos.map((todo, index) => (
18                 <li key={index}>
19                     <input type="checkbox" />
20                     {todo.text}
21                 </li>
22             ))}
23         </ul>
24     );
25
26     // Bene - usa id univoco
27     return (
28         <ul>
29             {todos.map(todo => (
30                 <li key={todo.id}>
31                     <input type="checkbox" />
32                     {todo.text}
33                 </li>
34             ))}
35         </ul>
36     );
37 }

```

### 3.1.3 JSX Avanzato

#### 1. Fragments

Raggruppa elementi senza aggiungere nodi DOM:

Listing 3.7: React Fragments

```

1 import { Fragment } from 'react';
2
3 function Table() {
4   // Aggiunge un div extra nel DOM
5   return (
6     <div>
7       <td>Cella 1</td>
8       <td>Cella 2</td>
9     </div>
10  );
11
12   // Sintassi lunga
13   return (
14     <Fragment>
15       <td>Cella 1</td>
16       <td>Cella 2</td>
17     </Fragment>
18  );
19
20   // Sintassi corta (più comune)
21   return (
22     <>
23       <td>Cella 1</td>
24       <td>Cella 2</td>
25     </>
26  );
27
28   // Con key (necessario per liste)
29   return items.map(item => (
30     <Fragment key={item.id}>
31       <td>{item.name}</td>
32       <td>{item.value}</td>
33     </Fragment>
34  ));
35 }

```

## 2. Spread Props

Listing 3.8: Spread di props

```

1 function Button({ type = 'button', className, ...rest }) {
2   // Passa tutte le altre props al button
3   return (
4     <button
5       type={type}
6       className={'btn ${className}'}
7       {...rest}
8     />
9   );
10 }
11
12 // Uso
13 function App() {
14   return (
15     <Button
16       onClick={() => console.log('Click')}
17       disabled
18       aria-label="Submit button"

```

```

19         data-testid="submit-btn"
20     >
21         Invia
22     </Button>
23 );
24 }

```

### 3. Children come Function

Listing 3.9: Render Props pattern

```

1 function DataFetcher({ url, children }) {
2     const [data, setData] = useState(null);
3     const [loading, setLoading] = useState(true);
4
5     useEffect(() => {
6         fetch(url)
7             .then(res => res.json())
8             .then(data => {
9                 setData(data);
10                setLoading(false);
11            });
12    }, [url]);
13
14    // Children è una funzione
15    return children({ data, loading });
16 }
17
18 // Uso
19 function App() {
20     return (
21         <DataFetcher url="/api/users">
22             ({ data, loading }) => (
23                 loading ? <Spinner /> : <UserList users={data} />
24             )
25         </DataFetcher>
26     );
27 }

```

## 3.2 Componenti Funzionali

I componenti funzionali sono il modo moderno e raccomandato di creare componenti React.

### 3.2.1 Anatomia di un Componente

Listing 3.10: Componente funzionale completo

```

1 // 1. Imports
2 import { useState, useEffect } from 'react';
3 import PropTypes from 'prop-types';
4 import './UserCard.css';
5
6 // 2. Componente
7 function UserCard({ user, onEdit, onDelete }) {
8     // 3. State locale
9     const [isExpanded, setIsExpanded] = useState(false);
10

```

```

11 // 4. Effects
12 useEffect(() => {
13     console.log('UserCard montato');
14     return () => console.log('UserCard smontato');
15 }, []);
16
17 // 5. Event handlers
18 const handleToggle = () => {
19     setIsExpanded(!isExpanded);
20 };
21
22 // 6. Computed values
23 const fullName = `${user.firstName} ${user.lastName}`;
24 const initials = `${user.firstName[0]}${user.lastName[0]}`;
25
26 // 7. Render
27 return (
28     <div className="user-card">
29         <div className="avatar">{initials}</div>
30
31         <div className="info">
32             <h3>{fullName}</h3>
33             <p>{user.email}</p>
34
35             {isExpanded && (
36                 <div className="details">
37                     <p>Ruolo: {user.role}</p>
38                     <p>Registrato: {user.registeredAt}</p>
39                 </div>
40             )}
41         </div>
42
43         <div className="actions">
44             <button onClick={handleToggle}>
45                 {isExpanded ? 'Nascondi' : 'Mostra'}
46             </button>
47             <button onClick={() => onEdit(user.id)}>
48                 Modifica
49             </button>
50             <button onClick={() => onDelete(user.id)}>
51                 Elimina
52             </button>
53         </div>
54     </div>
55 );
56 }
57
58 // 8. PropTypes
59 UserCard.propTypes = {
60     user: PropTypes.shape({
61         id: PropTypes.number.isRequired,
62         firstName: PropTypes.string.isRequired,
63         lastName: PropTypes.string.isRequired,
64         email: PropTypes.string.isRequired,
65         role: PropTypes.string,
66         registeredAt: PropTypes.string
67     }).isRequired,
68     onEdit: PropTypes.func.isRequired,

```



```
69     onDelete: PropTypes.func.isRequired
70 };
71
72 // 9. Export
73 export default UserCard;
```

### 3.2.2 Props: Comunicazione tra Componenti

Props (properties) sono il meccanismo per passare dati dai componenti parent ai componenti child.

#### 1. Props Base

Listing 3.11: Props fondamentali

```
1 // Parent component
2 function App() {
3     const user = {
4         name: 'Mario Rossi',
5         age: 30,
6         email: 'mario@example.com'
7     };
8
9     return (
10         <div>
11             {/* Passaggio props */}
12             <Greeting name="Mario" />
13             <UserInfo user={user} />
14             <Button label="Click me" onClick={() => alert('Clicked!')} />
15         </div>
16     );
17 }
18
19 // Child components
20 function Greeting({ name }) {
21     return <h1>Ciao, {name}!</h1>;
22 }
23
24 function UserInfo({ user }) {
25     return (
26         <div>
27             <p>Nome: {user.name}</p>
28             <p>Età: {user.age}</p>
29             <p>Email: {user.email}</p>
30         </div>
31     );
32 }
33
34 function Button({ label, onClick }) {
35     return <button onClick={onClick}>{label}</button>;
36 }
```

#### 2. Props Destructuring

Listing 3.12: Destructuring di props

```
1 // Senza destructuring
2 function UserCard(props) {
3     return (
```

```

4      <div>
5          <h3>{props.name}</h3>
6          <p>{props.email}</p>
7      </div>
8  );
9  }
10
11  //      Con destructuring (raccomandato)
12  function UserCard({ name, email }) {
13      return (
14          <div>
15              <h3>{name}</h3>
16              <p>{email}</p>
17          </div>
18      );
19  }
20
21  //      Con valori di default
22  function UserCard({
23      name = 'Guest',
24      email = 'No email',
25      role = 'user',
26      isActive = true
27  }) {
28      return (
29          <div>
30              <h3>{name}</h3>
31              <p>{email}</p>
32              <span>{role}</span>
33              <span>{isActive ? 'Active' : 'Inactive'}</span>
34          </div>
35      );
36  }
37
38  //      Rest props
39  function Button({ children, className, ...rest }) {
40      return (
41          <button className={`btn ${className}`} {...rest}>
42              {children}
43          </button>
44      );
45  }

```

### 3. Children Prop

Listing 3.13: Children prop

```

1  // Children come contenuto
2  function Card({ children, title }) {
3      return (
4          <div className="card">
5              <h2>{title}</h2>
6              <div className="card-body">
7                  {children}
8              </div>
9          </div>
10     );
11 }
12

```

```

13 // Uso
14 function App() {
15     return (
16         <Card title="User Info">
17             <p>Nome: Mario</p>
18             <p>Email: mario@example.com</p>
19             <button>Contatta</button>
20         </Card>
21     );
22 }
23
24 // Children multipli
25 function Layout({ header, sidebar, children, footer }) {
26     return (
27         <div className="layout">
28             <header>{header}</header>
29             <aside>{sidebar}</aside>
30             <main>{children}</main>
31             <footer>{footer}</footer>
32         </div>
33     );
34 }
35
36 // Uso
37 function App() {
38     return (
39         <Layout
40             header={<Header />}
41             sidebar={<Sidebar />}
42             footer={<Footer />}
43         >
44             <h1>Contenuto principale</h1>
45             <p>Lorem ipsum...</p>
46         </Layout>
47     );
48 }

```

## 4. Callback Props

Listing 3.14: Props come callbacks

```

1 // Parent passa funzioni ai child
2 function App() {
3     const [count, setCount] = useState(0);
4
5     const handleIncrement = () => {
6         setCount(count + 1);
7     };
8
9     const handleDecrement = () => {
10         setCount(count - 1);
11     };
12
13     const handleReset = () => {
14         setCount(0);
15     };
16
17     return (
18         <div>

```

```

19         <Display value={count} />
20         <Controls
21             onIncrement={handleIncrement}
22             onDecrement={handleDecrement}
23             onReset={handleReset}
24         />
25     </div>
26 );
27 }
28
29 function Display({ value }) {
30     return <h1>Contatore: {value}</h1>;
31 }
32
33 function Controls({ onIncrement, onDecrement, onReset }) {
34     return (
35         <div>
36             <button onClick={onDecrement}>-</button>
37             <button onClick={onReset}>Reset</button>
38             <button onClick={onIncrement}>+</button>
39         </div>
40     );
41 }

```

### 3.2.3 Composizione di Componenti

La composizione è il pattern principale per costruire UI complesse in React.

#### 1. Composizione Base

Listing 3.15: Composizione semplice

```

1 // Componenti atomici
2 function Avatar({ src, alt }) {
3     return <img src={src} alt={alt} className="avatar" />;
4 }
5
6 function UserName({ name }) {
7     return <h3 className="username">{name}</h3>;
8 }
9
10 function UserEmail({ email }) {
11     return <p className="email">{email}</p>;
12 }
13
14 // Componente composto
15 function UserCard({ user }) {
16     return (
17         <div className="user-card">
18             <Avatar src={user.avatar} alt={user.name} />
19             <div>
20                 <UserName name={user.name} />
21                 <UserEmail email={user.email} />
22             </div>
23         </div>
24     );
25 }
26
27 // Lista di UserCard

```

```

28 function UserList({ users }) {
29   return (
30     <div className="user-list">
31       {users.map(user => (
32         <UserCard key={user.id} user={user} />
33       ))}
34     </div>
35   );
36 }

```

## 2. Container/Presentational Pattern

Listing 3.16: Container e Presentational components

```

1  // Presentational Component (UI pura)
2  function TodoListView({ todos, onToggle, onDelete }) {
3    return (
4      <ul className="todo-list">
5        {todos.map(todo => (
6          <TodoItem
7            key={todo.id}
8            todo={todo}
9            onToggle={onToggle}
10           onDelete={onDelete}
11         />
12       ))}
13     </ul>
14   );
15 }
16
17 function TodoItem({ todo, onToggle, onDelete }) {
18   return (
19     <li className={todo.completed ? 'completed' : ''}>
20       <input
21         type="checkbox"
22         checked={todo.completed}
23         onChange={() => onToggle(todo.id)}
24       />
25       <span>{todo.text}</span>
26       <button onClick={() => onDelete(todo.id)}>Delete</button>
27     </li>
28   );
29 }
30
31 // Container Component (logica)
32 function TodoListContainer() {
33   const [todos, setTodos] = useState([]);
34   const [loading, setLoading] = useState(true);
35
36   useEffect(() => {
37     fetchTodos().then(data => {
38       setTodos(data);
39       setLoading(false);
40     });
41   }, []);
42
43   const handleToggle = (id) => {
44     setTodos(todos.map(todo =>
45       todo.id === id

```

```

46         ? { ...todo, completed: !todo.completed }
47         : todo
48     ));
49 };
50
51     const handleDelete = (id) => {
52         setTodos(todos.filter(todo => todo.id !== id));
53     };
54
55     if (loading) return <Spinner />;
56
57     return (
58         <TodoListView
59             todos={todos}
60             onToggle={handleToggle}
61             onDelete={handleDelete}
62         />
63     );
64 }

```

### 3. Specializzazione

Listing 3.17: Componenti specializzati

```

1  // Componente generico
2  function Dialog({ title, message, onClose, children }) {
3      return (
4          <div className="dialog">
5              <div className="dialog-header">
6                  <h2>{title}</h2>
7                  <button onClick={onClose}> </button>
8              </div>
9              <div className="dialog-body">
10                 {message} && <p>{message}</p>
11                 {children}
12             </div>
13         </div>
14     );
15 }
16
17 // Componenti specializzati
18 function WelcomeDialog({ onClose }) {
19     return (
20         <Dialog
21             title="Benvenuto"
22             message="Grazie per esserti registrato!"
23             onClose={onClose}
24         />
25     );
26 }
27
28 function ConfirmDialog({ title, message, onConfirm, onCancel }) {
29     return (
30         <Dialog title={title} message={message} onClose={onCancel}>
31             <div className="dialog-actions">
32                 <button onClick={onCancel}>Annulla</button>
33                 <button onClick={onConfirm}>Conferma</button>
34             </div>
35         </Dialog>

```

```

36     );
37 }
38
39 function AlertDialog({ message, onClose }) {
40     return (
41         <Dialog title="Attenzione" message={message} onClose={onClose}>
42             <button onClick={onClose}>OK</button>
43         </Dialog>
44     );
45 }

```

#### 4. Compound Components

Listing 3.18: Compound Components pattern

```

1 // Tabs compound component
2 function Tabs({ children, defaultActive = 0 }) {
3     const [activeIndex, setActiveIndex] = useState(defaultActive);
4
5     return (
6         <div className="tabs">
7             {React.Children.map(children, (child, index) => {
8                 return React.cloneElement(child, {
9                     isActive: index === activeIndex,
10                    onClick: () => setActiveIndex(index),
11                    index
12                });
13            })}
14         </div>
15     );
16 }
17
18 function Tab({ label, children, isActive, onClick }) {
19     return (
20         <div className={`tab ${isActive ? 'active' : ''}`}>
21             <button onClick={onClick}>{label}</button>
22             {isActive && <div className="tab-content">{children}</div>}
23         </div>
24     );
25 }
26
27 // Uso
28 function App() {
29     return (
30         <Tabs defaultActive={1}>
31             <Tab label="Tab 1">
32                 <p>Contenuto Tab 1</p>
33             </Tab>
34             <Tab label="Tab 2">
35                 <p>Contenuto Tab 2</p>
36             </Tab>
37             <Tab label="Tab 3">
38                 <p>Contenuto Tab 3</p>
39             </Tab>
40         </Tabs>
41     );
42 }

```

### 3.3 Component Tree

#### 3.3.1 Diagramma Component Tree

Esempio di albero componenti per un'applicazione TODO:

```

App
  Header
    Logo
    UserMenu
      Avatar
      Dropdown
        MenuItem (Profile)
        MenuItem (Settings)
        MenuItem (Logout)

  TodoApp
    TodoInput
      Input
      Button (Add)

    FilterBar
      FilterButton (All)
      FilterButton (Active)
      FilterButton (Completed)

    TodoList
      TodoItem (x5)
        Checkbox
        Text
        Button (Delete)
      EmptyState (se vuoto)

    TodoStats
      Stat (Total)
      Stat (Completed)
      Stat (Pending)

  Footer
    Copyright
    Links
      Link (Privacy)
      Link (Terms)

```

#### 3.3.2 Implementazione Component Tree

Listing 3.19: Implementazione albero completo

```

1 // App - Root component
2 function App() {
3   const [user, setUser] = useState({
4     name: 'Mario Rossi',
5     avatar: '/avatar.jpg'
6   });

```



```

7
8     return (
9         <div className="app">
10             <Header user={user} onLogout={() => setUser(null)} />
11             <TodoApp />
12             <Footer />
13         </div>
14     );
15 }
16
17 // Header component
18 function Header({ user, onLogout }) {
19     return (
20         <header className="header">
21             <Logo />
22             <UserMenu user={user} onLogout={onLogout} />
23         </header>
24     );
25 }
26
27 function Logo() {
28     return <div className="logo">      TodoApp</div>;
29 }
30
31 function UserMenu({ user, onLogout }) {
32     const [isOpen, setIsOpen] = useState(false);
33
34     return (
35         <div className="user-menu">
36             <Avatar src={user.avatar} onClick={() => setIsOpen(!isOpen)} />
37             {isOpen && (
38                 <Dropdown>
39                     <MenuItem label="Profile" onClick={() => {}} />
40                     <MenuItem label="Settings" onClick={() => {}} />
41                     <MenuItem label="Logout" onClick={onLogout} />
42                 </Dropdown>
43             )}
44         </div>
45     );
46 }
47
48 // TodoApp component
49 function TodoApp() {
50     const [todos, setTodos] = useState([]);
51     const [filter, setFilter] = useState('all');
52
53     const addTodo = (text) => {
54         setTodos([...todos, {
55             id: Date.now(),
56             text,
57             completed: false
58         }]);
59     };
60
61     const toggleTodo = (id) => {
62         setTodos(todos.map(todo =>
63             todo.id === id

```

```

64         ? { ...todo, completed: !todo.completed }
65         : todo
66     ));
67 };
68
69     const deleteTodo = (id) => {
70         setTodos(todos.filter(todo => todo.id !== id));
71     };
72
73     const filteredTodos = todos.filter(todo => {
74         if (filter === 'active') return !todo.completed;
75         if (filter === 'completed') return todo.completed;
76         return true;
77     });
78
79     return (
80         <main className="todo-app">
81             <TodoInput onAdd={addTodo} />
82             <FilterBar filter={filter} onFilterChange={setFilter} />
83             <TodoList
84                 todos={filteredTodos}
85                 onToggle={toggleTodo}
86                 onDelete={deleteTodo}
87             />
88             <TodoStats todos={todos} />
89         </main>
90     );
91 }
92
93 // Componenti foglia (leaf components)
94 function TodoInput({ onAdd }) {
95     const [text, setText] = useState('');
96
97     const handleSubmit = (e) => {
98         e.preventDefault();
99         if (text.trim()) {
100             onAdd(text);
101             setText('');
102         }
103     };
104
105     return (
106         <form onSubmit={handleSubmit} className="todo-input">
107             <Input
108                 value={text}
109                 onChange={setText}
110                 placeholder="Nuovo task..."
111             />
112             <Button type="submit">Aggiungi</Button>
113         </form>
114     );
115 }
116
117 function TodoList({ todos, onToggle, onDelete }) {
118     if (todos.length === 0) {
119         return <EmptyState />;
120     }
121 }

```

```

122     return (
123       <ul className="todo-list">
124         {todos.map(todo => (
125           <TodoItem
126             key={todo.id}
127             todo={todo}
128             onToggle={onToggle}
129             onDelete={onDelete}
130           />
131         ))}
132       </ul>
133     );
134   }
135
136   function TodoStats({ todos }) {
137     const total = todos.length;
138     const completed = todos.filter(t => t.completed).length;
139     const pending = total - completed;
140
141     return (
142       <div className="todo-stats">
143         <Stat label="Totale" value={total} />
144         <Stat label="Completati" value={completed} />
145         <Stat label="Da fare" value={pending} />
146       </div>
147     );
148   }

```

## 3.4 Best Practices

### 3.4.1 1. Naming e Organizzazione

Listing 3.20: Convenzioni di naming

```

1  // Componenti: PascalCase
2  function UserProfile() {}
3  function TodoList() {}
4
5  // File: stesso nome del componente
6  // UserProfile.jsx
7  // TodoList.jsx
8
9  // Props: camelCase descrittivo
10 function Button({ onClick, isDisabled, primaryColor }) {}
11
12 // Event handlers: handle + Event
13 const handleClick = () => {};
14 const handleSubmit = () => {};
15 const handleInputChange = () => {};
16
17 // Boolean props: is/has/should prefix
18 const isLoading = false;
19 const hasError = true;
20 const shouldRender = true;
21
22 // Evitare nomi generici
23 function Component() {} // Troppo generico

```

```
24 function MyComponent() {} // "My" non aggiunge informazioni
```

### 3.4.2 2. Componenti Piccoli e Focalizzati

Listing 3.21: Single Responsibility

```
1 //      Componente che fa troppo
2 function UserDashboard() {
3     // 100+ righe di logica mista
4     // Fetching dati, gestione form, rendering complesso
5 }
6
7 //      Dividi responsabilità
8 function UserDashboard() {
9     return (
10         <div>
11             <UserStats />
12             <UserActivity />
13             <UserSettings />
14         </div>
15     );
16 }
17
18 function UserStats() {
19     // Solo statistiche
20 }
21
22 function UserActivity() {
23     // Solo attività recente
24 }
25
26 function UserSettings() {
27     // Solo impostazioni
28 }
```

### 3.4.3 3. Props Validation

Listing 3.22: PropTypes per validazione

```
1 import PropTypes from 'prop-types';
2
3 function UserCard({ user, onEdit, onDelete, isSelected }) {
4     // ...
5 }
6
7 UserCard.propTypes = {
8     user: PropTypes.shape({
9         id: PropTypes.number.isRequired,
10         name: PropTypes.string.isRequired,
11         email: PropTypes.string.isRequired,
12         avatar: PropTypes.string,
13         role: PropTypes.oneOf(['admin', 'user', 'guest'])
14     }).isRequired,
15     onEdit: PropTypes.func.isRequired,
16     onDelete: PropTypes.func,
17     isSelected: PropTypes.bool
```

```
18 };
19
20 UserCard.defaultProps = {
21   isSelected: false,
22   onDelete: () => {}
23 };
```

### 3.4.4 4. Evitare Prop Drilling Eccessivo

Listing 3.23: Problema prop drilling

```
1 //      Prop drilling eccessivo
2 function App() {
3   const [theme, setTheme] = useState('light');
4   return <Layout theme={theme} setTheme={setTheme} />;
5 }
6
7 function Layout({ theme, setTheme }) {
8   return <Sidebar theme={theme} setTheme={setTheme} />;
9 }
10
11 function Sidebar({ theme, setTheme }) {
12   return <ThemeToggle theme={theme} setTheme={setTheme} />;
13 }
14
15 function ThemeToggle({ theme, setTheme }) {
16   return <button onClick={() => setTheme(/* ... */)}>Toggle</button>;
17 }
18
19 //      Usa Context (vedi capitoli successivi)
20 const ThemeContext = createContext();
21
22 function App() {
23   const [theme, setTheme] = useState('light');
24   return (
25     <ThemeContext.Provider value={{ theme, setTheme }}>
26       <Layout />
27     </ThemeContext.Provider>
28   );
29 }
30
31 function ThemeToggle() {
32   const { theme, setTheme } = useContext(ThemeContext);
33   return <button onClick={() => setTheme(/* ... */)}>Toggle</button>;
34 }
```

## 3.5 Conclusione

In questo capitolo abbiamo coperto:

- Sintassi JSX completa e avanzata
- Componenti funzionali moderni
- Props e comunicazione tra componenti

- Patterns di composizione
- Component tree e architettura
- Best practices fondamentali

Ora sei pronto per approfondire props, state e la gestione dei dati!

## Capitolo 4

# Props e State Management

### 4.1 Props: Dati Immutabili

Le props (properties) sono il meccanismo primario per passare dati in React. Sono immutabili dal punto di vista del componente che le riceve.

#### 4.1.1 Props Immutability

Listing 4.1: Props sono read-only

```
1 function Welcome({ name }) {  
2   //      ERRORE: Non puoi modificare le props  
3   // name = 'Nuovo nome'; // TypeError in Strict Mode  
4  
5   //      Le props si leggono soltanto  
6   return <h1>Benvenuto, {name}</h1>;  
7 }  
8  
9 function App() {  
10   const [userName, setUserName] = useState('Mario');  
11  
12   // Per cambiare i dati, cambi lo state nel parent  
13   return (  
14     <div>  
15       <Welcome name={userName} />  
16       <button onClick={() => setUserName('Luigi')}>  
17         Cambia nome  
18       </button>  
19     </div>  
20   );  
21 }
```

#### 4.1.2 Tipi di Props

##### 1. Props Primitive

Listing 4.2: Props primitive

```
1 function UserProfile({  
2   name,           // string  
3   age,            // number  
4   isActive,       // boolean  
5   score           // number  
6 }) {
```

```
7     return (
8         <div>
9             <h2>{name}</h2>
10            <p>Età: {age}</p>
11            <p>Stato: {isActive ? 'Attivo' : 'Inattivo'}</p>
12            <p>Punteggio: {score}</p>
13        </div>
14    );
15 }
16
17 // Uso
18 <UserProfile
19     name="Mario"
20     age={30}
21     isActive={true}
22     score={95.5}
23 />
```

## 2. Props Oggetti e Array

Listing 4.3: Props complesse

```
1 function ProductCard({ product, tags, seller }) {
2     return (
3         <div className="product-card">
4             <h3>{product.name}</h3>
5             <p>    {product.price}</p>
6             <p>{product.description}</p>
7
8             <div className="tags">
9                 {tags.map(tag => (
10                     <span key={tag} className="tag">{tag}</span>
11                 ))}
12             </div>
13
14             <div className="seller">
15                 <img src={seller.avatar} alt={seller.name} />
16                 <span>{seller.name}</span>
17             </div>
18         </div>
19     );
20 }
21
22 // Uso
23 function App() {
24     const product = {
25         id: 1,
26         name: 'Laptop',
27         price: 999,
28         description: 'Un ottimo laptop'
29     };
30
31     const tags = ['Elettronica', 'Computer', 'Offerta'];
32
33     const seller = {
34         id: 10,
35         name: 'TechStore',
36         avatar: '/seller.jpg'
37     };
```



```

38
39     return (
40         <ProductCard
41             product={product}
42             tags={tags}
43             seller={seller}
44         />
45     );
46 }

```

### 3. Props Funzioni (Callbacks)

Listing 4.4: Callback props

```

1 function SearchBar({ onSearch, onClear, onFilterChange }) {
2     const [query, setQuery] = useState('');
3     const [filter, setFilter] = useState('all');
4
5     const handleSearch = () => {
6         onSearch(query, filter);
7     };
8
9     const handleClear = () => {
10         setQuery('');
11         onClear();
12     };
13
14     const handleFilterChange = (newFilter) => {
15         setFilter(newFilter);
16         onFilterChange(newFilter);
17     };
18
19     return (
20         <div className="search-bar">
21             <input
22                 value={query}
23                 onChange={(e) => setQuery(e.target.value)}
24                 placeholder="Cerca..."
25             />
26             <button onClick={handleSearch}>Cerca</button>
27             <button onClick={handleClear}>Cancella</button>
28
29             <select
30                 value={filter}
31                 onChange={(e) => handleFilterChange(e.target.value)}
32             >
33                 <option value="all">Tutti</option>
34                 <option value="active">Attivi</option>
35                 <option value="archived">Archiviati</option>
36             </select>
37         </div>
38     );
39 }
40
41 // Uso nel parent
42 function App() {
43     const handleSearch = (query, filter) => {
44         console.log('Ricerca:', query, 'Filtro:', filter);
45         // Fetch dati...

```

```

46     };
47
48     const handleClear = () => {
49         console.log('Reset ricerca');
50     };
51
52     const handleFilterChange = (filter) => {
53         console.log('Filtro cambiato:', filter);
54     };
55
56     return (
57         <SearchBar
58             onSearch={handleSearch}
59             onClear={handleClear}
60             onFilterChange={handleFilterChange}
61         />
62     );
63 }

```

#### 4. Props come Componenti (Render Props)

Listing 4.5: Componenti come props

```

1  function Layout({ header, sidebar, footer, children }) {
2      return (
3          <div className="layout">
4              <header className="header">{header}</header>
5              <div className="main-content">
6                  <aside className="sidebar">{sidebar}</aside>
7                  <main className="content">{children}</main>
8              </div>
9              <footer className="footer">{footer}</footer>
10         </div>
11     );
12 }
13
14 // Uso
15 function App() {
16     return (
17         <Layout
18             header={<AppHeader />}
19             sidebar={<Navigation />}
20             footer={<AppFooter />}
21         >
22             <h1>Contenuto principale</h1>
23             <p>Contenuto della pagina...</p>
24         </Layout>
25     );
26 }

```

## 4.2 Props Drilling

Props drilling si verifica quando passi props attraverso molti livelli di componenti intermedi.

### 4.2.1 Il Problema del Props Drilling

Listing 4.6: Props drilling eccessivo

```
1 // Livello 1: App
2 function App() {
3   const [user, setUser] = useState({
4     name: 'Mario',
5     theme: 'dark',
6     language: 'it'
7   });
8
9   return <Layout user={user} setUser={setUser} />;
10 }
11
12 // Livello 2: Layout (non usa user, solo passa)
13 function Layout({ user, setUser }) {
14   return (
15     <div>
16       <Header user={user} setUser={setUser} />
17       <Main user={user} setUser={setUser} />
18     </div>
19   );
20 }
21
22 // Livello 3: Header (non usa user, solo passa)
23 function Header({ user, setUser }) {
24   return (
25     <header>
26       <Navigation user={user} setUser={setUser} />
27     </header>
28   );
29 }
30
31 // Livello 4: Navigation (non usa user, solo passa)
32 function Navigation({ user, setUser }) {
33   return (
34     <nav>
35       <UserMenu user={user} setUser={setUser} />
36     </nav>
37   );
38 }
39
40 // Livello 5: UserMenu (FINALMENTE usa user!)
41 function UserMenu({ user, setUser }) {
42   return (
43     <div>
44       <span>{user.name}</span>
45       <button onClick={() => setUser({ ...user, theme: 'light' })}>
46         Cambia tema
47       </button>
48     </div>
49   );
50 }
```

## 4.2.2 Soluzioni al Props Drilling

### 1. Component Composition

Listing 4.7: Composizione invece di drilling

```

1 //      Passa il componente completo, non le props
2 function App() {
3     const [user, setUser] = useState({
4         name: 'Mario',
5         theme: 'dark'
6     });
7
8     // Costruisci UserMenu qui, dove hai accesso a user
9     const userMenu = <UserMenu user={user} setUser={setUser} />;
10
11     return <Layout userMenu={userMenu} />;
12 }
13
14 function Layout({ userMenu }) {
15     return (
16         <div>
17             <Header userMenu={userMenu} />
18             <Main />
19         </div>
20     );
21 }
22
23 function Header({ userMenu }) {
24     return (
25         <header>
26             <Navigation userMenu={userMenu} />
27         </header>
28     );
29 }
30
31 function Navigation({ userMenu }) {
32     return (
33         <nav>
34             <Logo />
35             {userMenu}  {/* Renderizza direttamente */}
36         </nav>
37     );
38 }
39
40 function UserMenu({ user, setUser }) {
41     return (
42         <div>
43             <span>{user.name}</span>
44             <button onClick={() => setUser({ ...user, theme: 'light' })}
45                 Cambia tema
46             </button>
47         </div>
48     );
49 }

```

## 2. Context API

Listing 4.8: Context per evitare drilling

```

1 import { createContext, useContext, useState } from 'react';
2
3 // 1. Crea Context

```

```
4  const UserContext = createContext();
5
6  // 2. Provider nel componente root
7  function App() {
8      const [user, setUser] = useState({
9          name: 'Mario',
10         theme: 'dark'
11     });
12
13     return (
14         <UserContext.Provider value={{ user, setUser }}>
15             <Layout />
16         </UserContext.Provider>
17     );
18 }
19
20 // 3. Componenti intermedi non hanno bisogno delle props
21 function Layout() {
22     return (
23         <div>
24             <Header />
25             <Main />
26         </div>
27     );
28 }
29
30 function Header() {
31     return (
32         <header>
33             <Navigation />
34         </header>
35     );
36 }
37
38 function Navigation() {
39     return (
40         <nav>
41             <Logo />
42             <UserMenu />
43         </nav>
44     );
45 }
46
47 // 4. Consuma il context dove serve
48 function UserMenu() {
49     const { user, setUser } = useContext(UserContext);
50
51     return (
52         <div>
53             <span>{user.name}</span>
54             <button onClick={() => setUser({ ...user, theme: 'light' })}>
55                 Cambia tema
56             </button>
57         </div>
58     );
59 }
```

### 3. Custom Hook per Context

Listing 4.9: Custom hook per context

```
1 // hooks/useUser.js
2 import { createContext, useContext, useState } from 'react';
3
4 const UserContext = createContext();
5
6 export function UserProvider({ children }) {
7   const [user, setUser] = useState({
8     name: 'Mario',
9     theme: 'dark',
10    language: 'it'
11  });
12
13  const updateUser = (updates) => {
14    setUser(prev => ({ ...prev, ...updates }));
15  };
16
17  const toggleTheme = () => {
18    setUser(prev => ({
19      ...prev,
20      theme: prev.theme === 'dark' ? 'light' : 'dark'
21    }));
22  };
23
24  return (
25    <UserContext.Provider value={{
26      user,
27      setUser,
28      updateUser,
29      toggleTheme
30    }}>
31      {children}
32    </UserContext.Provider>
33  );
34 }
35
36 export function useUser() {
37   const context = useContext(UserContext);
38   if (!context) {
39     throw new Error('useUser must be used within UserProvider');
40   }
41   return context;
42 }
43
44 // App.jsx
45 import { UserProvider } from '../hooks/useUser';
46
47 function App() {
48   return (
49     <UserProvider>
50       <Layout />
51     </UserProvider>
52   );
53 }
54
55 // Qualsiasi componente
```

```
56 import { useUser } from './hooks/useUser';
57
58 function UserMenu() {
59   const { user, toggleTheme } = useUser();
60
61   return (
62     <div>
63       <span>{user.name}</span>
64       <button onClick={toggleTheme}>
65         Tema: {user.theme}
66       </button>
67     </div>
68   );
69 }
```

## 4.3 State Management

Lo state è dati mutabili gestiti dal componente. Quando lo state cambia, React re-renderizza il componente.

### 4.3.1 State Locale con useState

Listing 4.10: useState base

```
1 import { useState } from 'react';
2
3 function Counter() {
4   // Dichiarazione state
5   const [count, setCount] = useState(0);
6   //      ^           ^           ^
7   //      |           |           |
8   //  valore setter   valore iniziale
9
10  // Aggiorna state
11  const increment = () => {
12    setCount(count + 1);
13  };
14
15  const decrement = () => {
16    setCount(count - 1);
17  };
18
19  const reset = () => {
20    setCount(0);
21  };
22
23  return (
24    <div>
25      <h1>Contatore: {count}</h1>
26      <button onClick={decrement}>-</button>
27      <button onClick={reset}>Reset</button>
28      <button onClick={increment}>+</button>
29    </div>
30  );
31 }
```

### 4.3.2 State con Oggetti

Listing 4.11: State oggetto

```

1 function UserForm() {
2   const [user, setUser] = useState({
3     firstName: '',
4     lastName: '',
5     email: '',
6     age: ''
7   });
8
9   //      SBAGLIATO: Muta direttamente lo state
10  const handleChangeBad = (field, value) => {
11    user[field] = value; // NO!
12    setUser(user);      // React non rileva il cambiamento
13  };
14
15  //      CORRETTO: Crea nuovo oggetto
16  const handleChange = (field, value) => {
17    setUser({
18      ...user,           // Copia proprietà esistenti
19      [field]: value     // Sovrascrivi campo specifico
20    });
21  };
22
23  //      Alternativa con funzione updater
24  const handleChangeUpdater = (field, value) => {
25    setUser(prevUser => ({
26      ...prevUser,
27      [field]: value
28    }));
29  };
30
31  const handleSubmit = (e) => {
32    e.preventDefault();
33    console.log('User data:', user);
34  };
35
36  return (
37    <form onSubmit={handleSubmit}>
38      <input
39        type="text"
40        value={user.firstName}
41        onChange={(e) => handleChange('firstName', e.target.
42          value)}
43        placeholder="Nome"
44      />
45      <input
46        type="text"
47        value={user.lastName}
48        onChange={(e) => handleChange('lastName', e.target.value
49          )}
50        placeholder="Cognome"
51      />
52      <input
53        type="email"
54        value={user.email}
55        onChange={(e) => handleChange('email', e.target.value)}

```



```

54         placeholder="Email"
55     />
56     <input
57         type="number"
58         value={user.age}
59         onChange={(e) => handleChange('age', e.target.value)}
60         placeholder="Età"
61     />
62     <button type="submit">Invia</button>
63
64     {/* Preview dati */}
65     <pre>{JSON.stringify(user, null, 2)}</pre>
66 </form>
67 );
68 }

```

### 4.3.3 State con Array

Listing 4.12: State array

```

1 function TodoList() {
2     const [todos, setTodos] = useState([
3         { id: 1, text: 'Imparare React', completed: false },
4         { id: 2, text: 'Costruire app', completed: false }
5     ]);
6
7     // Aggiungi elemento
8     const addTodo = (text) => {
9         const newTodo = {
10             id: Date.now(),
11             text,
12             completed: false
13         };
14
15         // SBAGLIATO
16         // todos.push(newTodo);
17         // setTodos(todos);
18
19         // CORRETTO
20         setTodos([...todos, newTodo]);
21         // oppure
22         setTodos(prevTodos => [...prevTodos, newTodo]);
23     };
24
25     // Rimuovi elemento
26     const deleteTodo = (id) => {
27         // Filter crea nuovo array
28         setTodos(todos.filter(todo => todo.id !== id));
29     };
30
31     // Aggiorna elemento
32     const toggleTodo = (id) => {
33         // Map crea nuovo array
34         setTodos(todos.map(todo =>
35             todo.id === id
36             ? { ...todo, completed: !todo.completed }
37             : todo

```

```

38     ));
39   };
40
41   // Aggiorna elemento con indice
42   const updateTodoAtIndex = (index, newText) => {
43     setTodos(todos.map((todo, i) =>
44       i === index
45         ? { ...todo, text: newText }
46         : todo
47     ));
48   };
49
50   // Inserisci elemento a inizio
51   const prependTodo = (text) => {
52     const newTodo = { id: Date.now(), text, completed: false };
53     setTodos([newTodo, ...todos]);
54   };
55
56   // Ordina array
57   const sortTodos = () => {
58     setTodos([...todos].sort((a, b) =>
59       a.text.localeCompare(b.text)
60     ));
61   };
62
63   // Rimuovi tutti completati
64   const clearCompleted = () => {
65     setTodos(todos.filter(todo => !todo.completed));
66   };
67
68   return (
69     <div>
70       {/* UI componenti... */}
71     </div>
72   );
73 }

```

#### 4.3.4 Multiple State Updates

Listing 4.13: Aggiornamenti multipli

```

1 function ShoppingCart() {
2   const [items, setItems] = useState([]);
3   const [total, setTotal] = useState(0);
4   const [discount, setDiscount] = useState(0);
5
6   // Problema: Usa state precedente
7   const addItem = (item) => {
8     setItems([...items, item]);
9     setTotal(total + item.price); // total potrebbe essere stale
10  };
11
12  // Usa funzione updater per accesso allo state più recente
13  const addItemCorrect = (item) => {
14    setItems(prevItems => [...prevItems, item]);
15    setTotal(prevTotal => prevTotal + item.price);
16  };

```

```

17
18 //      Aggiornamenti batch automatici in React 18+
19 const handleCheckout = () => {
20     setItems([]);
21     setTotal(0);
22     setDiscount(0);
23     // Tutti e tre gli aggiornamenti causano un solo re-render
24 };
25
26 // Calcola totale da items (computed value)
27 const calculatedTotal = items.reduce((sum, item) => sum + item.price
28     , 0);
29 const finalTotal = calculatedTotal - discount;
30
31 return (
32     <div>
33         <p>Items: {items.length}</p>
34         <p>Totale:    {finalTotal}</p>
35     </div>
36 );
37 }

```

### 4.3.5 Lifting State Up

Quando più componenti devono condividere lo stesso state, "alza" lo state al parent comune.

Listing 4.14: Lifting state up

```

1 //      State duplicato in componenti separati
2 function TemperatureInput() {
3     const [temperature, setTemperature] = useState('');
4     return <input value={temperature} onChange={e => setTemperature(e.
5         target.value)} />;
6 }
7
8 function App() {
9     return (
10         <div>
11             <TemperatureInput /> {/* Celsius */}
12             <TemperatureInput /> {/* Fahrenheit */}
13             {/* I due input non sono sincronizzati! */}
14         </div>
15     );
16 }
17
18 //      State condiviso nel parent
19 function TemperatureInput({ scale, temperature, onTemperatureChange }) {
20     const handleChange = (e) => {
21         onTemperatureChange(e.target.value);
22     };
23
24     const scaleNames = {
25         c: 'Celsius',
26         f: 'Fahrenheit'
27     };
28
29     return (
30         <fieldset>

```

```

30         <legend>Temperatura in {scaleNames[scale]}:</legend>
31         <input value={temperature} onChange={handleChange} />
32     </fieldset>
33 );
34 }
35
36 function App() {
37     const [temperature, setTemperature] = useState('');
38     const [scale, setScale] = useState('c');
39
40     const handleCelsiusChange = (temp) => {
41         setScale('c');
42         setTemperature(temp);
43     };
44
45     const handleFahrenheitChange = (temp) => {
46         setScale('f');
47         setTemperature(temp);
48     };
49
50     // Conversioni
51     const celsius = scale === 'f'
52       ? ((temperature - 32) * 5 / 9).toFixed(1)
53       : temperature;
54
55     const fahrenheit = scale === 'c'
56       ? (temperature * 9 / 5 + 32).toFixed(1)
57       : temperature;
58
59     return (
60       <div>
61         <TemperatureInput
62           scale="c"
63           temperature={celsius}
64           onTemperatureChange={handleCelsiusChange}
65         />
66         <TemperatureInput
67           scale="f"
68           temperature={fahrenheit}
69           onTemperatureChange={handleFahrenheitChange}
70         />
71         <BoilingVerdict celsius={parseFloat(celsius)} />
72       </div>
73     );
74 }
75
76 function BoilingVerdict({ celsius }) {
77     if (celsius >= 100) {
78         return <p>L'acqua bollirebbe.</p>;
79     }
80     return <p>L'acqua non bollirebbe.</p>;
81 }

```

## 4.4 Controlled Components

Un controlled component è un elemento form il cui valore è controllato da React state.

### 4.4.1 Input Controllati

Listing 4.15: Controlled input

```
1 function ControlledInput() {
2   const [value, setValue] = useState('');
3
4   //      Controlled: React controlla il valore
5   return (
6     <div>
7       <input
8         type="text"
9         value={value}
10        onChange={(e) => setValue(e.target.value)}
11      />
12      <p>Valore: {value}</p>
13    </div>
14  );
15
16  //      Uncontrolled: DOM controlla il valore
17  // return <input type="text" />;
18 }
```

### 4.4.2 Form Completo Controllato

Listing 4.16: Form completamente controllato

```
1 function RegistrationForm() {
2   const [formData, setFormData] = useState({
3     username: '',
4     email: '',
5     password: '',
6     confirmPassword: '',
7     age: '',
8     country: '',
9     terms: false,
10    newsletter: false,
11    gender: '',
12    interests: []
13  });
14
15  const [errors, setErrors] = useState({});
16
17  // Handler generico per input text/email/password/number
18  const handleInputChange = (e) => {
19    const { name, value } = e.target;
20    setFormData(prev => ({
21      ...prev,
22      [name]: value
23    }));
24  };
25
26  // Handler per checkbox
27  const handleCheckboxChange = (e) => {
28    const { name, checked } = e.target;
29    setFormData(prev => ({
30      ...prev,
31      [name]: checked
32    }));
33  };
34}
```

```
32     }));
33 };
34
35 // Handler per checkbox multipli (interests)
36 const handleInterestChange = (interest) => {
37     setFormData(prev => ({
38         ...prev,
39         interests: prev.interests.includes(interest)
40             ? prev.interests.filter(i => i !== interest)
41             : [...prev.interests, interest]
42     }));
43 };
44
45 // Validazione
46 const validate = () => {
47     const newErrors = {};
48
49     if (!formData.username) {
50         newErrors.username = 'Username richiesto';
51     } else if (formData.username.length < 3) {
52         newErrors.username = 'Username troppo corto';
53     }
54
55     if (!formData.email) {
56         newErrors.email = 'Email richiesta';
57     } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
58         newErrors.email = 'Email non valida';
59     }
60
61     if (!formData.password) {
62         newErrors.password = 'Password richiesta';
63     } else if (formData.password.length < 8) {
64         newErrors.password = 'Password troppo corta (min 8 caratteri)';
65     }
66
67     if (formData.password !== formData.confirmPassword) {
68         newErrors.confirmPassword = 'Le password non coincidono';
69     }
70
71     if (!formData.terms) {
72         newErrors.terms = 'Devi accettare i termini';
73     }
74
75     setErrors(newErrors);
76     return Object.keys(newErrors).length === 0;
77 };
78
79 const handleSubmit = (e) => {
80     e.preventDefault();
81
82     if (validate()) {
83         console.log('Form valido:', formData);
84         // Invia dati al server...
85     } else {
86         console.log('Form non valido:', errors);
87     }
88 };
```

```

89
90     return (
91         <form onSubmit={handleSubmit}>
92             {/* Text Input */}
93             <div>
94                 <label htmlFor="username">Username:</label>
95                 <input
96                     id="username"
97                     type="text"
98                     name="username"
99                     value={formData.username}
100                     onChange={handleInputChange}
101                 />
102                 {errors.username && <span className="error">{errors.
                    username}</span>}
103             </div>
104
105             {/* Email Input */}
106             <div>
107                 <label htmlFor="email">Email:</label>
108                 <input
109                     id="email"
110                     type="email"
111                     name="email"
112                     value={formData.email}
113                     onChange={handleInputChange}
114                 />
115                 {errors.email && <span className="error">{errors.email
                    }</span>}
116             </div>
117
118             {/* Password Input */}
119             <div>
120                 <label htmlFor="password">Password:</label>
121                 <input
122                     id="password"
123                     type="password"
124                     name="password"
125                     value={formData.password}
126                     onChange={handleInputChange}
127                 />
128                 {errors.password && <span className="error">{errors.
                    password}</span>}
129             </div>
130
131             {/* Confirm Password */}
132             <div>
133                 <label htmlFor="confirmPassword">Conferma Password:</
                    label>
134                 <input
135                     id="confirmPassword"
136                     type="password"
137                     name="confirmPassword"
138                     value={formData.confirmPassword}
139                     onChange={handleInputChange}
140                 />
141                 {errors.confirmPassword && <span className="error">{
                    errors.confirmPassword}</span>}

```

```

142     </div>
143
144     { /* Number Input */ }
145     <div>
146         <label htmlFor="age">Età:</label>
147         <input
148             id="age"
149             type="number"
150             name="age"
151             value={formData.age}
152             onChange={handleInputChange}
153             min="18"
154             max="120"
155         />
156     </div>
157
158     { /* Select */ }
159     <div>
160         <label htmlFor="country">Paese:</label>
161         <select
162             id="country"
163             name="country"
164             value={formData.country}
165             onChange={handleInputChange}
166         >
167             <option value="">Seleziona...</option>
168             <option value="IT">Italia</option>
169             <option value="US">USA</option>
170             <option value="UK">UK</option>
171         </select>
172     </div>
173
174     { /* Radio Buttons */ }
175     <div>
176         <label>Genere:</label>
177         <label>
178             <input
179                 type="radio"
180                 name="gender"
181                 value="male"
182                 checked={formData.gender === 'male'}
183                 onChange={handleInputChange}
184             />
185             Maschio
186         </label>
187         <label>
188             <input
189                 type="radio"
190                 name="gender"
191                 value="female"
192                 checked={formData.gender === 'female'}
193                 onChange={handleInputChange}
194             />
195             Femmina
196         </label>
197         <label>
198             <input
199                 type="radio"

```



```

200         name="gender"
201         value="other"
202         checked={formData.gender === 'other'}
203         onChange={handleInputChange}
204     />
205     Altro
206   </label>
207 </div>
208
209 {/* Checkboxes Multiple */}
210 <div>
211   <label>Interessi:</label>
212   {[ 'Sport', 'Musica', 'Lettura', 'Viaggi' ].map(interest
213     => (
214       <label key={interest}>
215         <input
216           type="checkbox"
217           checked={formData.interests.includes(
218             interest)}
219           onChange={() => handleInterestChange(
220             interest)}
221         />
222         {interest}
223       </label>
224     )
225   )}
226 </div>
227
228 {/* Checkbox Singola */}
229 <div>
230   <label>
231     <input
232       type="checkbox"
233       name="terms"
234       checked={formData.terms}
235       onChange={handleCheckboxChange}
236     />
237     Accetto i termini e condizioni
238   </label>
239   {errors.terms && <span className="error">{errors.terms}
240     </span>}
241 </div>
242
243 <div>
244   <label>
245     <input
246       type="checkbox"
247       name="newsletter"
248       checked={formData.newsletter}
249       onChange={handleCheckboxChange}
250     />
251     Iscriviti alla newsletter
252   </label>
253 </div>
254
255 {/* Submit */}
256 <button type="submit">Registrati</button>
257
258 {/* Debug */}

```

```

254         <pre>{JSON.stringify(formData, null, 2)}</pre>
255     </form>
256     );
257 }

```

### 4.4.3 Controlled vs Uncontrolled

Listing 4.17: Controlled vs Uncontrolled

```

1  import { useRef } from 'react';
2
3  function ComparisonExample() {
4      // Controlled
5      const [controlledValue, setControlledValue] = useState('');
6
7      // Uncontrolled
8      const uncontrolledRef = useRef();
9
10     const handleControlledSubmit = (e) => {
11         e.preventDefault();
12         console.log('Controlled:', controlledValue);
13     };
14
15     const handleUncontrolledSubmit = (e) => {
16         e.preventDefault();
17         console.log('Uncontrolled:', uncontrolledRef.current.value);
18     };
19
20     return (
21         <div>
22             {/* Controlled Input */}
23             <form onSubmit={handleControlledSubmit}>
24                 <h3>Controlled</h3>
25                 <input
26                     type="text"
27                     value={controlledValue}
28                     onChange={(e) => setControlledValue(e.target.value)}
29                 />
30                 <button type="submit">Invia</button>
31                 <p>Live value: {controlledValue}</p>
32             </form>
33
34             {/* Uncontrolled Input */}
35             <form onSubmit={handleUncontrolledSubmit}>
36                 <h3>Uncontrolled</h3>
37                 <input
38                     type="text"
39                     ref={uncontrolledRef}
40                     defaultValue="Initial"
41                 />
42                 <button type="submit">Invia</button>
43                 <p>Value disponibile solo al submit</p>
44             </form>
45         </div>
46     );
47 }
48

```

```

49 // Quando usare controlled vs uncontrolled:
50
51 // Usa CONTROLLED quando:
52 // - Hai bisogno di validazione in tempo reale
53 // - Vuoi formattare l'input mentre digiti
54 // - Devi disabilitare il submit in base all'input
55 // - Hai input dipendenti tra loro
56 // - Vuoi mostrare l'input in tempo reale
57
58 // Usa UNCONTROLLED quando:
59 // - Form molto grandi con molti campi
60 // - Integri con librerie non-React
61 // - Performance critiche
62 // - File uploads
63 // - Semplicemente leggi il valore al submit

```

## 4.5 State Management Patterns

### 4.5.1 Single Source of Truth

Listing 4.18: Single source of truth

```

1 function ShoppingApp() {
2   // State centralizzato
3   const [cart, setCart] = useState({
4     items: [],
5     total: 0,
6     itemCount: 0
7   });
8
9   // Computed values derivati dallo state
10  const subtotal = cart.items.reduce((sum, item) => sum + item.price *
11    item.quantity, 0);
12  const tax = subtotal * 0.22;
13  const total = subtotal + tax;
14
15  const addItem = (product) => {
16    setCart(prev => {
17      const existingItem = prev.items.find(item => item.id ===
18        product.id);
19
20      if (existingItem) {
21        // Incrementa quantità
22        return {
23          ...prev,
24          items: prev.items.map(item =>
25            item.id === product.id
26              ? { ...item, quantity: item.quantity + 1 }
27              : item
28          );
29      }
30      // Aggiungi nuovo item
31      return {
32        ...prev,
33        items: [...prev.items, { ...product, quantity: 1 }]
34      };
35    });
36  };
37 }

```

```

34         }
35     });
36 };
37
38     return (
39         <div>
40             <ProductList onAddToCart={addItem} />
41             <Cart
42                 items={cart.items}
43                 subtotal={subtotal}
44                 tax={tax}
45                 total={total}
46             />
47         </div>
48     );
49 }

```

### 4.5.2 Derived State (Computed Values)

Listing 4.19: Evita state ridondante

```

1  function UserList() {
2      const [users, setUsers] = useState([...]);
3      const [searchQuery, setSearchQuery] = useState('');
4      const [sortBy, setSortBy] = useState('name');
5
6      //      SBAGLIATO: State ridondante
7      // const [filteredUsers, setFilteredUsers] = useState([]);
8      // const [sortedUsers, setSortedUsers] = useState([]);
9
10     //      CORRETTO: Calcola al render
11     const filteredUsers = users.filter(user =>
12         user.name.toLowerCase().includes(searchQuery.toLowerCase())
13     );
14
15     const sortedUsers = [...filteredUsers].sort((a, b) => {
16         if (sortBy === 'name') return a.name.localeCompare(b.name);
17         if (sortBy === 'age') return a.age - b.age;
18         return 0;
19     });
20
21     return (
22         <div>
23             <input
24                 value={searchQuery}
25                 onChange={(e) => setSearchQuery(e.target.value)}
26                 placeholder="Cerca..."
27             />
28             <select value={sortBy} onChange={(e) => setSortBy(e.target.
29                 value)}>
30                 <option value="name">Nome</option>
31                 <option value="age">Età</option>
32             </select>
33
34             <ul>
35                 {sortedUsers.map(user => (

```

```

36         )))}
37     </ul>
38 </div>
39 );
40 }

```

### 4.5.3 State Reducer Pattern

Listing 4.20: Pattern reducer per state complesso

```

1  import { useReducer } from 'react';
2
3  // Reducer function
4  function cartReducer(state, action) {
5      switch (action.type) {
6          case 'ADD_ITEM':
7              const existingIndex = state.items.findIndex(
8                  item => item.id === action.payload.id
9              );
10
11              if (existingIndex >= 0) {
12                  const newItems = [...state.items];
13                  newItems[existingIndex].quantity += 1;
14                  return { ...state, items: newItems };
15              }
16
17              return {
18                  ...state,
19                  items: [...state.items, { ...action.payload, quantity: 1
20                      }]}];
21
22          case 'REMOVE_ITEM':
23              return {
24                  ...state,
25                  items: state.items.filter(item => item.id !== action.
26                      payload)
27              };
28
29          case 'UPDATE_QUANTITY':
30              return {
31                  ...state,
32                  items: state.items.map(item =>
33                      item.id === action.payload.id
34                      ? { ...item, quantity: action.payload.quantity }
35                      : item
36                  )
37              };
38
39          case 'CLEAR_CART':
40              return { ...state, items: [] };
41
42          default:
43              return state;
44      }
45  }

```

```
46 function ShoppingCart() {
47   const [state, dispatch] = useReducer(cartReducer, { items: [] });
48
49   const addItem = (product) => {
50     dispatch({ type: 'ADD_ITEM', payload: product });
51   };
52
53   const removeItem = (id) => {
54     dispatch({ type: 'REMOVE_ITEM', payload: id });
55   };
56
57   const updateQuantity = (id, quantity) => {
58     dispatch({ type: 'UPDATE_QUANTITY', payload: { id, quantity } });
59   };
60
61   const clearCart = () => {
62     dispatch({ type: 'CLEAR_CART' });
63   };
64
65   return (
66     <div>
67       {/* UI usando dispatch */}
68     </div>
69   );
70 }
```

## 4.6 Best Practices

### 4.6.1 1. Mantieni State Minimo

Listing 4.21: State minimo necessario

```
1  //      State ridondante
2  function BadExample() {
3    const [firstName, setFirstName] = useState('');
4    const [lastName, setLastName] = useState('');
5    const [fullName, setFullName] = useState(''); // Ridondante!
6
7    const handleFirstNameChange = (name) => {
8      setFirstName(name);
9      setFullName(`${name} ${lastName}`); // Duplicazione logica
10   };
11 }
12
13 //      Calcola fullName
14 function GoodExample() {
15   const [firstName, setFirstName] = useState('');
16   const [lastName, setLastName] = useState('');
17
18   // Computed value
19   const fullName = `${firstName} ${lastName}.trim();
20
21   return <h1>{fullName}</h1>;
22 }
```

### 4.6.2 2. Normalizza State Complesso

Listing 4.22: Normalizzazione state

```
1 //      State denormalizzato
2 const [data, setData] = useState({
3   posts: [
4     {
5       id: 1,
6       title: 'Post 1',
7       author: { id: 10, name: 'Mario' },
8       comments: [
9         { id: 100, text: 'Comment 1', author: { id: 10, name: '
10         Mario' } }
11       ]
12     }
13   ]
14 });
15 //      State normalizzato
16 const [data, setData] = useState({
17   posts: {
18     1: { id: 1, title: 'Post 1', authorId: 10, commentIds: [100] }
19   },
20   users: {
21     10: { id: 10, name: 'Mario' }
22   },
23   comments: {
24     100: { id: 100, text: 'Comment 1', authorId: 10 }
25   }
26 });
```

### 4.6.3 3. Usa Funzione Updater

Listing 4.23: Updater function

```
1 //      Può causare problemi
2 const handleClick = () => {
3   setCount(count + 1);
4   setCount(count + 1); // Usa lo stesso valore di count!
5   // count aumenta solo di 1, non 2
6 };
7
8 //      Usa updater function
9 const handleClick = () => {
10   setCount(prev => prev + 1);
11   setCount(prev => prev + 1);
12   // count aumenta di 2
13 };
```

## 4.7 Conclusione

In questo capitolo abbiamo esplorato:

- Props immutabili e tipi di props

- Props drilling e soluzioni
- State management con useState
- Controlled components per form
- Patterns di state management
- Best practices per props e state

Nel prossimo capitolo approfondiremo gli Hooks di React!



## Capitolo 5

# React Hooks

### 5.1 Introduzione agli Hooks

Gli Hooks sono funzioni speciali che permettono di "agganciare" funzionalità di React (state, lifecycle, context, ecc.) nei componenti funzionali. Introdotti in React 16.8, hanno rivoluzionato il modo di scrivere componenti React.

#### 5.1.1 Regole degli Hooks

**Le 2 Regole Fondamentali:**

1. **Chiama gli Hooks solo al top level:** Non chiamarli dentro loop, condizioni o funzioni annidate
2. **Chiama gli Hooks solo da React functions:** Componenti funzionali o custom hooks

Listing 5.1: Regole degli Hooks

```
1 function Component() {
2   //      CORRETTO: Top level
3   const [count, setCount] = useState(0);
4   const [name, setName] = useState('');
5
6   //      SBAGLIATO: Dentro condizione
7   if (count > 5) {
8     const [error, setError] = useState(''); // NO!
9   }
10
11  //      SBAGLIATO: Dentro loop
12  for (let i = 0; i < 10; i++) {
13    const [value, setValue] = useState(i); // NO!
14  }
15
16  //      SBAGLIATO: Dentro funzione normale
17  function handleClick() {
18    const [temp, setTemp] = useState(0); // NO!
19  }
20
21  //      CORRETTO: Hook chiamato al top level
22  useEffect(() => {
23    // Effetto qui dentro è OK
24  }, [count]);
25 }
```

```
26     return <div>{count}</div>;
27 }
28
29 //      CORRETTO: Custom hook
30 function useCustomHook() {
31     const [state, setState] = useState(0);
32     return [state, setState];
33 }
```

## 5.2 useState: State Management

useState è l'hook fondamentale per aggiungere state ai componenti funzionali.

### 5.2.1 Sintassi Base

Listing 5.2: useState base

```
1  import { useState } from 'react';
2
3  function Counter() {
4      // Dichiarazione
5      const [count, setCount] = useState(0);
6      //      ^           ^           ^
7      //      |           |           |
8      //  valore  setter   valore iniziale
9
10     return (
11         <div>
12             <p>Count: {count}</p>
13             <button onClick={() => setCount(count + 1)}>
14                 Incrementa
15             </button>
16         </div>
17     );
18 }
```

### 5.2.2 useState con Lazy Initialization

Listing 5.3: Lazy initialization

```
1  function ExpensiveComponent() {
2      //      Funzione eseguita ad ogni render
3      const [data, setData] = useState(expensiveComputation());
4
5      //      Funzione eseguita solo al mount
6      const [data, setData] = useState(() => expensiveComputation());
7
8      // Esempio pratico
9      const [user, setUser] = useState(() => {
10         const savedUser = localStorage.getItem('user');
11         return savedUser ? JSON.parse(savedUser) : null;
12     });
13
14     return <div>{user?.name}</div>;
15 }
```

```
16
17 function expensiveComputation() {
18   console.log('Computing...');
19   let result = 0;
20   for (let i = 0; i < 1000000; i++) {
21     result += i;
22   }
23   return result;
24 }
```

### 5.2.3 useState con Oggetti Complessi

Listing 5.4: State oggetti

```
1 function UserProfile() {
2   const [user, setUser] = useState({
3     name: '',
4     email: '',
5     age: 0,
6     preferences: {
7       theme: 'light',
8       notifications: true
9     }
10  });
11
12  // Aggiorna campo top-level
13  const updateName = (name) => {
14    setUser(prev => ({ ...prev, name }));
15  };
16
17  // Aggiorna campo nested
18  const updateTheme = (theme) => {
19    setUser(prev => ({
20      ...prev,
21      preferences: {
22        ...prev.preferences,
23        theme
24      }
25    }));
26  };
27
28  // Aggiorna multipli campi
29  const updateMultiple = () => {
30    setUser(prev => ({
31      ...prev,
32      name: 'Mario',
33      age: 30
34    }));
35  };
36
37  return (
38    <div>
39      <input
40        value={user.name}
41        onChange={(e) => updateName(e.target.value)}
42      />
43      <select
```

```
44         value={user.preferences.theme}
45         onChange={(e) => updateTheme(e.target.value)}
46     >
47         <option value="light">Light</option>
48         <option value="dark">Dark</option>
49     </select>
50 </div>
51 );
52 }
```

## 5.3 useEffect: Side Effects

`useEffect` permette di eseguire side effects nei componenti funzionali: data fetching, subscriptions, DOM manipulation, ecc.

### 5.3.1 useEffect Base

Listing 5.5: `useEffect` sintassi

```
1 import { useState, useEffect } from 'react';
2
3 function Component() {
4     const [count, setCount] = useState(0);
5
6     // 1. Esegue ad ogni render
7     useEffect(() => {
8         console.log('Eseguito ad ogni render');
9     });
10
11    // 2. Esegue solo al mount
12    useEffect(() => {
13        console.log('Eseguito solo al mount');
14    }, []);
15
16    // 3. Esegue quando count cambia
17    useEffect(() => {
18        console.log('Count è cambiato:', count);
19    }, [count]);
20
21    // 4. Con cleanup function
22    useEffect(() => {
23        console.log('Setup');
24        return () => {
25            console.log('Cleanup');
26        };
27    }, []);
28
29    return <div>{count}</div>;
30 }
```

### 5.3.2 Component Lifecycle con `useEffect`

Component Lifecycle:

## COMPONENT LIFECYCLE

1. MOUNT (prima volta che il componente appare)
  - > `useEffect(() => { ... }, [])`
  - > Eseguito UNA SOLA VOLTA
2. UPDATE (quando props/state cambiano)
  - > `useEffect(() => { ... }, [dep1, dep2])`
  - > Eseguito quando dep1 o dep2 cambiano
3. UNMOUNT (componente rimosso dal DOM)
  - > `useEffect(() => {`
  - `return () => { cleanup };`
  - `}, [])`

Listing 5.6: Lifecycle completo

```

1 function UserProfile({ userId }) {
2   const [user, setUser] = useState(null);
3   const [loading, setLoading] = useState(true);
4
5   // componentDidMount + componentDidUpdate (quando userId cambia)
6   useEffect(() => {
7     console.log('Fetching user:', userId);
8     setLoading(true);
9
10    fetch(`/api/users/${userId}`)
11      .then(res => res.json())
12      .then(data => {
13        setUser(data);
14        setLoading(false);
15      });
16
17    // componentWillUnmount / cleanup
18    return () => {
19      console.log('Cleanup per userId:', userId);
20    };
21  }, [userId]); // Re-esegue quando userId cambia
22
23  if (loading) return <div>Loading...</div>;
24  return <div>{user?.name}</div>;
25 }

```

## 5.3.3 useEffect: Data Fetching

Listing 5.7: Data fetching pattern

```

1 function PostsList() {
2   const [posts, setPosts] = useState([]);
3   const [loading, setLoading] = useState(true);
4   const [error, setError] = useState(null);
5
6   useEffect(() => {
7     let cancelled = false;
8

```

```

9      const fetchPosts = async () => {
10        try {
11          setLoading(true);
12          setError(null);
13
14          const response = await fetch('/api/posts');
15          if (!response.ok) {
16            throw new Error('Failed to fetch');
17          }
18
19          const data = await response.json();
20
21          // Evita state update se componente è unmounted
22          if (!cancelled) {
23            setPosts(data);
24            setLoading(false);
25          }
26        } catch (err) {
27          if (!cancelled) {
28            setError(err.message);
29            setLoading(false);
30          }
31        }
32      };
33
34      fetchPosts();
35
36      // Cleanup: segna come cancelled
37      return () => {
38        cancelled = true;
39      };
40    }, []);
41
42    if (loading) return <div>Loading...</div>;
43    if (error) return <div>Error: {error}</div>;
44
45    return (
46      <ul>
47        {posts.map(post => (
48          <li key={post.id}>{post.title}</li>
49        ))}
50      </ul>
51    );
52  }

```

### 5.3.4 useEffect: Subscriptions

Listing 5.8: Event subscriptions

```

1  function WindowSize() {
2    const [size, setSize] = useState({
3      width: window.innerWidth,
4      height: window.innerHeight
5    });
6
7    useEffect(() => {
8      // Setup subscription

```

```

9      const handleResize = () => {
10          setSize({
11              width: window.innerWidth,
12              height: window.innerHeight
13          });
14      };
15
16      window.addEventListener('resize', handleResize);
17
18      // Cleanup subscription
19      return () => {
20          window.removeEventListener('resize', handleResize);
21      };
22  }, []); // Array vuoto = solo mount/unmount
23
24  return (
25      <div>
26          Window size: {size.width} x {size.height}
27      </div>
28  );
29  }
30
31  // WebSocket subscription
32  function ChatRoom({ roomId }) {
33      const [messages, setMessages] = useState([]);
34
35      useEffect(() => {
36          const ws = new WebSocket('ws://api.example.com/chat/${roomId}');
37
38          ws.onmessage = (event) => {
39              const message = JSON.parse(event.data);
40              setMessages(prev => [...prev, message]);
41          };
42
43          ws.onerror = (error) => {
44              console.error('WebSocket error:', error);
45          };
46
47          // Cleanup: chiudi connessione
48          return () => {
49              ws.close();
50          };
51      }, [roomId]); // Riconnetti quando roomId cambia
52
53      return (
54          <ul>
55              {messages.map((msg, i) => (
56                  <li key={i}>{msg.text}</li>
57              ))}
58          </ul>
59      );
60  }

```

### 5.3.5 useEffect: Timers

```

1 function Timer() {
2   const [seconds, setSeconds] = useState(0);
3   const [isRunning, setIsRunning] = useState(false);
4
5   useEffect(() => {
6     if (!isRunning) return;
7
8     const interval = setInterval(() => {
9       setSeconds(prev => prev + 1);
10    }, 1000);
11
12    // Cleanup: clear interval
13    return () => clearInterval(interval);
14  }, [isRunning]); // Re-esegue quando isRunning cambia
15
16  return (
17    <div>
18      <p>Secondi: {seconds}</p>
19      <button onClick={() => setIsRunning(!isRunning)}>
20        {isRunning ? 'Pausa' : 'Avvia'}
21      </button>
22      <button onClick={() => setSeconds(0)}>Reset</button>
23    </div>
24  );
25 }
26
27 // Countdown timer
28 function Countdown({ initialSeconds }) {
29   const [timeLeft, setTimeLeft] = useState(initialSeconds);
30
31   useEffect(() => {
32     if (timeLeft <= 0) return;
33
34     const timeout = setTimeout(() => {
35       setTimeLeft(timeLeft - 1);
36     }, 1000);
37
38     return () => clearTimeout(timeout);
39   }, [timeLeft]);
40
41   return (
42     <div>
43       {timeLeft > 0 ? (
44         <p>Tempo rimanente: {timeLeft}s</p>
45       ) : (
46         <p>Tempo scaduto!</p>
47       )}
48     </div>
49   );
50 }

```

## 5.4 useContext: Context API

useContext permette di consumare context senza wrapper component.

### 5.4.1 Creazione e Uso del Context



Listing 5.10: Context pattern completo

```

1 import { createContext, useContext, useState } from 'react';
2
3 // 1. Crea Context
4 const ThemeContext = createContext();
5
6 // 2. Provider Component
7 export function ThemeProvider({ children }) {
8   const [theme, setTheme] = useState('light');
9
10  const toggleTheme = () => {
11    setTheme(prev => prev === 'light' ? 'dark' : 'light');
12  };
13
14  const value = {
15    theme,
16    setTheme,
17    toggleTheme
18  };
19
20  return (
21    <ThemeContext.Provider value={value}>
22      {children}
23    </ThemeContext.Provider>
24  );
25 }
26
27 // 3. Custom Hook per consumare Context
28 export function useTheme() {
29   const context = useContext(ThemeContext);
30   if (context === undefined) {
31     throw new Error('useTheme must be used within ThemeProvider');
32   }
33   return context;
34 }
35
36 // 4. Uso in App
37 function App() {
38   return (
39     <ThemeProvider>
40       <Navbar />
41       <Content />
42     </ThemeProvider>
43   );
44 }
45
46 // 5. Consume Context in qualsiasi componente
47 function Navbar() {
48   const { theme, toggleTheme } = useTheme();
49
50   return (
51     <nav className={`navbar-${theme}`}>
52       <button onClick={toggleTheme}>
53         Tema: {theme}
54       </button>
55     </nav>
56   );
57 }

```

```
58
59 function Content() {
60     const { theme } = useTheme();
61
62     return (
63         <div className={`content-${theme}`}>
64             <p>Contenuto con tema {theme}</p>
65         </div>
66     );
67 }
```

## 5.4.2 Context Multipli

Listing 5.11: Multipli context providers

```
1 // contexts/AuthContext.js
2 const AuthContext = createContext();
3
4 export function AuthProvider({ children }) {
5     const [user, setUser] = useState(null);
6
7     const login = async (credentials) => {
8         const userData = await api.login(credentials);
9         setUser(userData);
10    };
11
12    const logout = () => {
13        setUser(null);
14    };
15
16    return (
17        <AuthContext.Provider value={{ user, login, logout }}>
18            {children}
19        </AuthContext.Provider>
20    );
21 }
22
23 export const useAuth = () => useContext(AuthContext);
24
25 // contexts/SettingsContext.js
26 const SettingsContext = createContext();
27
28 export function SettingsProvider({ children }) {
29     const [settings, setSettings] = useState({
30         language: 'it',
31         notifications: true
32    });
33
34    const updateSettings = (updates) => {
35        setSettings(prev => ({ ...prev, ...updates }));
36    };
37
38    return (
39        <SettingsContext.Provider value={{ settings, updateSettings }}>
40            {children}
41        </SettingsContext.Provider>
42    );
43 }
```

```
43 }
44
45 export const useSettings = () => useContext(SettingsContext);
46
47 // App.jsx - Componenti multipli providers
48 function App() {
49   return (
50     <AuthProvider>
51       <SettingsProvider>
52         <ThemeProvider>
53           <Router>
54             <Routes />
55           </Router>
56         </ThemeProvider>
57       </SettingsProvider>
58     </AuthProvider>
59   );
60 }
61
62 // Uso in componenti
63 function UserDashboard() {
64   const { user, logout } = useAuth();
65   const { settings } = useSettings();
66   const { theme } = useTheme();
67
68   return (
69     <div className={theme}>
70       <h1>Ciao, {user.name}</h1>
71       <p>Lingua: {settings.language}</p>
72       <button onClick={logout}>Logout</button>
73     </div>
74   );
75 }
```

## 5.5 useRef: Riferimenti Mutabili

useRef crea un riferimento mutabile che persiste tra i render senza causare re-rendering.

### 5.5.1 useRef per DOM Access

Listing 5.12: Accesso al DOM con useRef

```
1 import { useRef, useEffect } from 'react';
2
3 function FocusInput() {
4   const inputRef = useRef(null);
5
6   useEffect(() => {
7     // Focus automatico al mount
8     inputRef.current.focus();
9   }, []);
10
11   const handleFocus = () => {
12     inputRef.current.focus();
13   };
14 }
```

```

15     return (
16         <div>
17             <input ref={inputRef} type="text" />
18             <button onClick={handleFocus}>Focus Input</button>
19         </div>
20     );
21 }
22
23 // Scroll to element
24 function ScrollToSection() {
25     const sectionRef = useRef(null);
26
27     const scrollToSection = () => {
28         sectionRef.current.scrollIntoView({ behavior: 'smooth' });
29     };
30
31     return (
32         <div>
33             <button onClick={scrollToSection}>Vai alla sezione</button>
34             <div style={{ height: '1000px' }}>Contenuto...</div>
35             <div ref={sectionRef}>
36                 <h2>Sezione Target</h2>
37             </div>
38         </div>
39     );
40 }
41
42 // Media playback control
43 function VideoPlayer({ src }) {
44     const videoRef = useRef(null);
45     const [isPlaying, setIsPlaying] = useState(false);
46
47     const togglePlay = () => {
48         if (isPlaying) {
49             videoRef.current.pause();
50         } else {
51             videoRef.current.play();
52         }
53         setIsPlaying(!isPlaying);
54     };
55
56     return (
57         <div>
58             <video ref={videoRef} src={src} />
59             <button onClick={togglePlay}>
60                 {isPlaying ? 'Pausa' : 'Play'}
61             </button>
62         </div>
63     );
64 }

```

### 5.5.2 useRef per Valori Mutabili

Listing 5.13: useRef per valori persistenti

```

1 function Timer() {
2     const [seconds, setSeconds] = useState(0);

```

```

3      const intervalRef = useRef(null);
4
5      const start = () => {
6          if (intervalRef.current) return; // Già in esecuzione
7
8          intervalRef.current = setInterval(() => {
9              setSeconds(prev => prev + 1);
10         }, 1000);
11     };
12
13     const stop = () => {
14         clearInterval(intervalRef.current);
15         intervalRef.current = null;
16     };
17
18     const reset = () => {
19         stop();
20         setSeconds(0);
21     };
22
23     useEffect(() => {
24         // Cleanup on unmount
25         return () => {
26             if (intervalRef.current) {
27                 clearInterval(intervalRef.current);
28             }
29         };
30     }, []);
31
32     return (
33         <div>
34             <p>Secondi: {seconds}</p>
35             <button onClick={start}>Start</button>
36             <button onClick={stop}>Stop</button>
37             <button onClick={reset}>Reset</button>
38         </div>
39     );
40 }
41
42 // Tracciare valore precedente
43 function usePrevious(value) {
44     const ref = useRef();
45
46     useEffect(() => {
47         ref.current = value;
48     }, [value]);
49
50     return ref.current;
51 }
52
53 // Uso
54 function Counter() {
55     const [count, setCount] = useState(0);
56     const prevCount = usePrevious(count);
57
58     return (
59         <div>
60             <p>Attuale: {count}</p>

```

```

61         <p>Precedente: {prevCount}</p>
62         <button onClick={() => setCount(count + 1)}>
63             Incrementa
64         </button>
65     </div>
66 );
67 }
68
69 // Evitare re-render inutili
70 function SearchComponent() {
71     const [query, setQuery] = useState('');
72     const timeoutRef = useRef(null);
73
74     const handleSearch = (value) => {
75         setQuery(value);
76
77         // Debounce: cancella timeout precedente
78         clearTimeout(timeoutRef.current);
79
80         timeoutRef.current = setTimeout(() => {
81             console.log('Searching for:', value);
82             // API call...
83         }, 500);
84     };
85
86     return (
87         <input
88             value={query}
89             onChange={(e) => handleSearch(e.target.value)}
90         />
91     );
92 }

```

## 5.6 useMemo: Memoizzazione Valori

useMemo memoizza il risultato di un calcolo costoso, ricalcolandolo solo quando le dipendenze cambiano.

### 5.6.1 useMemo Base

Listing 5.14: useMemo per ottimizzazioni

```

1  import { useMemo, useState } from 'react';
2
3  function ExpensiveList({ items, filter }) {
4      // Senza useMemo: filtra ad ogni render
5      const filteredItems = items.filter(item =>
6          item.name.includes(filter)
7      );
8
9      // Con useMemo: filtra solo quando items o filter cambiano
10     const filteredItems = useMemo(() => {
11         console.log('Filtering items...');
12         return items.filter(item => item.name.includes(filter));
13     }, [items, filter]);
14 }

```

```

15     return (
16         <ul>
17             {filteredItems.map(item => (
18                 <li key={item.id}>{item.name}</li>
19             ))}
20         </ul>
21     );
22 }
23
24 // Calcoli complessi
25 function DataAnalysis({ data }) {
26     const statistics = useMemo(() => {
27         console.log('Calculating statistics...');
28
29         const sum = data.reduce((acc, val) => acc + val, 0);
30         const avg = sum / data.length;
31         const min = Math.min(...data);
32         const max = Math.max(...data);
33
34         return { sum, avg, min, max };
35     }, [data]);
36
37     return (
38         <div>
39             <p>Somma: {statistics.sum}</p>
40             <p>Media: {statistics.avg}</p>
41             <p>Min: {statistics.min}</p>
42             <p>Max: {statistics.max}</p>
43         </div>
44     );
45 }

```

### 5.6.2 useMemo per Reference Equality

Listing 5.15: useMemo per stabilità riferimenti

```

1 function Parent() {
2     const [count, setCount] = useState(0);
3     const [name, setName] = useState('Mario');
4
5     // Nuovo oggetto ad ogni render
6     const user = {
7         name: name,
8         age: 30
9     };
10
11     // Stesso oggetto se name non cambia
12     const user = useMemo(() => ({
13         name: name,
14         age: 30
15     })), [name]);
16
17     // Child non re-renderizza inutilmente quando count cambia
18     return (
19         <div>
20             <p>Count: {count}</p>
21             <button onClick={() => setCount(count + 1)}>Inc</button>

```

```

22         <ExpensiveChild user={user} />
23     </div>
24 );
25 }
26
27 const ExpensiveChild = React.memo(({ user }) => {
28     console.log('ExpensiveChild render');
29     return <div>{user.name}</div>;
30 });

```

## 5.7 useCallback: Memoizzazione Funzioni

useCallback memoizza una funzione, restituendo la stessa istanza finché le dipendenze non cambiano.

### 5.7.1 useCallback Base

Listing 5.16: useCallback per callbacks

```

1  import { useCallback, useState } from 'react';
2
3  function Parent() {
4      const [count, setCount] = useState(0);
5      const [name, setName] = useState('Mario');
6
7      // Nuova funzione ad ogni render
8      const handleClick = () => {
9          console.log('Clicked');
10     };
11
12     // Stessa funzione tra i render
13     const handleClick = useCallback(() => {
14         console.log('Clicked');
15     }, []); // Nessuna dipendenza
16
17     // Con dipendenze
18     const handleChange = useCallback((newName) => {
19         console.log('Old name:', name);
20         setName(newName);
21     }, [name]); // Ricreata solo quando name cambia
22
23     return (
24         <div>
25             <p>Count: {count}</p>
26             <button onClick={() => setCount(count + 1)}>Inc</button>
27             <ExpensiveButton onClick={handleClick} />
28         </div>
29     );
30 }
31
32 const ExpensiveButton = React.memo(({ onClick }) => {
33     console.log('ExpensiveButton render');
34     return <button onClick={onClick}>Click me</button>;
35 });

```



### 5.7.2 useCallback vs useMemo

Listing 5.17: Differenza useCallback vs useMemo

```

1 function Component() {
2   // useCallback: memoizza la FUNZIONE
3   const handleClick = useCallback(() => {
4     console.log('Clicked');
5   }, []);
6
7   // useMemo: memoizza il RISULTATO della funzione
8   const value = useMemo(() => {
9     return expensiveCalculation();
10  }, []);
11
12  // Equivalenze:
13  // useCallback(fn, deps) === useMemo(() => fn, deps)
14
15  const fn1 = useCallback(() => console.log('Hi'), []);
16  const fn2 = useMemo(() => () => console.log('Hi'), []);
17  // fn1 === fn2
18
19  return <div />;
20 }

```

### 5.7.3 useCallback: Casi d'Uso

Listing 5.18: useCallback pattern comuni

```

1 function TodoList() {
2   const [todos, setTodos] = useState([]);
3
4   // Callback passata a child
5   const handleToggle = useCallback((id) => {
6     setTodos(prev => prev.map(todo =>
7       todo.id === id
8         ? { ...todo, completed: !todo.completed }
9         : todo
10    ));
11  }, []); // Usa funzione updater, nessuna dipendenza
12
13  const handleDelete = useCallback((id) => {
14    setTodos(prev => prev.filter(todo => todo.id !== id));
15  }, []);
16
17  return (
18    <div>
19      {todos.map(todo => (
20        <TodoItem
21          key={todo.id}
22          todo={todo}
23          onToggle={handleToggle}
24          onDelete={handleDelete}
25        />
26      ))}
27    </div>
28  );
29 }

```

```

30
31 const TodoItem = React.memo(({ todo, onToggle, onDelete }) => {
32   console.log('TodoItem render:', todo.id);
33
34   return (
35     <div>
36       <input
37         type="checkbox"
38         checked={todo.completed}
39         onChange={() => onToggle(todo.id)}
40       />
41       <span>{todo.text}</span>
42       <button onClick={() => onDelete(todo.id)}>Delete</button>
43     </div>
44   );
45 });

```

## 5.8 Custom Hooks

I custom hooks permettono di estrarre logica riutilizzabile in funzioni separate.

### 5.8.1 Custom Hook Base

Listing 5.19: Primo custom hook

```

1 // hooks/useCounter.js
2 import { useState } from 'react';
3
4 function useCounter(initialValue = 0) {
5   const [count, setCount] = useState(initialValue);
6
7   const increment = () => setCount(prev => prev + 1);
8   const decrement = () => setCount(prev => prev - 1);
9   const reset = () => setCount(initialValue);
10
11   return { count, increment, decrement, reset };
12 }
13
14 // Uso
15 function Counter() {
16   const { count, increment, decrement, reset } = useCounter(0);
17
18   return (
19     <div>
20       <p>Count: {count}</p>
21       <button onClick={decrement}>-</button>
22       <button onClick={reset}>Reset</button>
23       <button onClick={increment}>+</button>
24     </div>
25   );
26 }

```

### 5.8.2 useLocalStorage

Listing 5.20: Custom hook localStorage

```

1  import { useState, useEffect } from 'react';
2
3  function useLocalStorage(key, initialValue) {
4      // State con lazy initialization
5      const [storedValue, setStoredValue] = useState(() => {
6          try {
7              const item = window.localStorage.getItem(key);
8              return item ? JSON.parse(item) : initialValue;
9          } catch (error) {
10             console.error(error);
11             return initialValue;
12         }
13     });
14
15     // Aggiorna localStorage quando il valore cambia
16     useEffect(() => {
17         try {
18             window.localStorage.setItem(key, JSON.stringify(storedValue)
19             );
20         } catch (error) {
21             console.error(error);
22         }
23     }, [key, storedValue]);
24
25     return [storedValue, setStoredValue];
26 }
27
28 // Uso
29 function App() {
30     const [name, setName] = useLocalStorage('name', 'Guest');
31     const [theme, setTheme] = useLocalStorage('theme', 'light');
32
33     return (
34         <div>
35             <input
36                 value={name}
37                 onChange={(e) => setName(e.target.value)}
38             />
39             <p>Nome salvato: {name}</p>
40
41             <button onClick={() => setTheme(theme === 'light' ? 'dark' :
42                 'light')}>
43                 Tema: {theme}
44             </button>
45         </div>
46     );
47 }

```

### 5.8.3 useFetch

Listing 5.21: Custom hook data fetching

```

1  import { useState, useEffect } from 'react';
2
3  function useFetch(url, options = {}) {
4      const [data, setData] = useState(null);

```

```

5     const [loading, setLoading] = useState(true);
6     const [error, setError] = useState(null);
7
8     useEffect(() => {
9         let cancelled = false;
10
11         const fetchData = async () => {
12             try {
13                 setLoading(true);
14                 setError(null);
15
16                 const response = await fetch(url, options);
17
18                 if (!response.ok) {
19                     throw new Error('HTTP error! status: ${response.
20                         status}');
21                 }
22
23                 const json = await response.json();
24
25                 if (!cancelled) {
26                     setData(json);
27                     setLoading(false);
28                 }
29             } catch (err) {
30                 if (!cancelled) {
31                     setError(err.message);
32                     setLoading(false);
33                 }
34             }
35
36             fetchData();
37
38             return () => {
39                 cancelled = true;
40             };
41         }, [url]);
42
43         return { data, loading, error };
44     }
45
46     // Uso
47     function UserProfile({ userId }) {
48         const { data: user, loading, error } = useFetch(`/api/users/${userId}
49             `);
50
51         if (loading) return <div>Loading...</div>;
52         if (error) return <div>Error: {error}</div>;
53
54         return (
55             <div>
56                 <h1>{user.name}</h1>
57                 <p>{user.email}</p>
58             </div>
59         );
60     }

```

### 5.8.4 useDebounce

Listing 5.22: Custom hook debounce

```

1 import { useState, useEffect } from 'react';
2
3 function useDebounce(value, delay = 500) {
4   const [debouncedValue, setDebouncedValue] = useState(value);
5
6   useEffect(() => {
7     const timer = setTimeout(() => {
8       setDebouncedValue(value);
9     }, delay);
10
11     return () => {
12       clearTimeout(timer);
13     };
14   }, [value, delay]);
15
16   return debouncedValue;
17 }
18
19 // Uso
20 function SearchComponent() {
21   const [searchTerm, setSearchTerm] = useState('');
22   const debouncedSearchTerm = useDebounce(searchTerm, 500);
23
24   useEffect(() => {
25     if (debouncedSearchTerm) {
26       console.log('Searching for:', debouncedSearchTerm);
27       // API call...
28     }
29   }, [debouncedSearchTerm]);
30
31   return (
32     <input
33       value={searchTerm}
34       onChange={(e) => setSearchTerm(e.target.value)}
35       placeholder="Cerca..."
36     />
37   );
38 }

```

### 5.8.5 useToggle

Listing 5.23: Custom hook toggle

```

1 import { useState, useCallback } from 'react';
2
3 function useToggle(initialValue = false) {
4   const [value, setValue] = useState(initialValue);
5
6   const toggle = useCallback(() => {
7     setValue(prev => !prev);
8   }, []);
9
10   const setTrue = useCallback(() => {
11     setValue(true);

```

```

12     }, []);
13
14     const setFalse = useCallback(() => {
15         setValue(false);
16     }, []);
17
18     return [value, toggle, setTrue, setFalse];
19 }
20
21 // Uso
22 function Modal() {
23     const [isOpen, toggle, open, close] = useToggle(false);
24
25     return (
26         <div>
27             <button onClick={open}>Apri Modal</button>
28
29             {isOpen && (
30                 <div className="modal">
31                     <h2>Modal</h2>
32                     <button onClick={close}>Chiudi</button>
33                     <button onClick={toggle}>Toggle</button>
34                 </div>
35             )}
36         </div>
37     );
38 }

```

### 5.8.6 useWindowSize

Listing 5.24: Custom hook window size

```

1  import { useState, useEffect } from 'react';
2
3  function useWindowSize() {
4      const [windowSize, setWindowSize] = useState({
5          width: window.innerWidth,
6          height: window.innerHeight
7      });
8
9      useEffect(() => {
10         const handleResize = () => {
11             setWindowSize({
12                 width: window.innerWidth,
13                 height: window.innerHeight
14             });
15         };
16
17         window.addEventListener('resize', handleResize);
18         return () => window.removeEventListener('resize', handleResize);
19     }, []);
20
21     return windowSize;
22 }
23
24 // Uso
25 function ResponsiveComponent() {

```

```

26     const { width, height } = useWindowSize();
27
28     return (
29       <div>
30         <p>Window size: {width} x {height}</p>
31         {width < 768 ? (
32           <MobileLayout />
33         ) : (
34           <DesktopLayout />
35         )}
36       </div>
37     );
38   }

```

### 5.8.7 useClickOutside

Listing 5.25: Custom hook click outside

```

1  import { useEffect, useRef } from 'react';
2
3  function useClickOutside(callback) {
4    const ref = useRef(null);
5
6    useEffect(() => {
7      const handleClick = (event) => {
8        if (ref.current && !ref.current.contains(event.target)) {
9          callback();
10        }
11      };
12
13      document.addEventListener('mousedown', handleClick);
14      return () => {
15        document.removeEventListener('mousedown', handleClick);
16      };
17    }, [callback]);
18
19    return ref;
20  }
21
22  // Uso
23  function Dropdown() {
24    const [isOpen, setIsOpen] = useState(false);
25    const dropdownRef = useClickOutside(() => setIsOpen(false));
26
27    return (
28      <div ref={dropdownRef}>
29        <button onClick={() => setIsOpen(!isOpen)}>
30          Menu
31        </button>
32
33        {isOpen && (
34          <ul className="dropdown-menu">
35            <li>Option 1</li>
36            <li>Option 2</li>
37            <li>Option 3</li>
38          </ul>
39        )}

```

```

40     </div>
41   );
42 }

```

## 5.9 Hook Composition

Combina hooks multipli per funzionalità complesse.

Listing 5.26: Composizione di hooks

```

1  // Custom hook che usa altri custom hooks
2  function useForm(initialValues, validate) {
3    const [values, setValues] = useState(initialValues);
4    const [errors, setErrors] = useState({});
5    const [touched, setTouched] = useState({});
6    const [isSubmitting, setIsSubmitting] = useState(false);
7
8    // Usa useCallback
9    const handleChange = useCallback((e) => {
10      const { name, value } = e.target;
11      setValues(prev => ({ ...prev, [name]: value }));
12    }, []);
13
14    const handleBlur = useCallback((e) => {
15      const { name } = e.target;
16      setTouched(prev => ({ ...prev, [name]: true }));
17    }, []);
18
19    // Usa useEffect
20    useEffect(() => {
21      if (Object.keys(touched).length > 0) {
22        const validationErrors = validate(values);
23        setErrors(validationErrors);
24      }
25    }, [values, touched, validate]);
26
27    const handleSubmit = useCallback(async (onSubmit) => {
28      setIsSubmitting(true);
29      const validationErrors = validate(values);
30
31      if (Object.keys(validationErrors).length === 0) {
32        await onSubmit(values);
33      }
34
35      setIsSubmitting(false);
36    }, [values, validate]);
37
38    return {
39      values,
40      errors,
41      touched,
42      isSubmitting,
43      handleChange,
44      handleBlur,
45      handleSubmit
46    };
47  }
48

```



```

49 // Uso
50 function LoginForm() {
51   const validate = (values) => {
52     const errors = {};
53     if (!values.email) errors.email = 'Email richiesta';
54     if (!values.password) errors.password = 'Password richiesta';
55     return errors;
56   };
57
58   const form = useForm(
59     { email: '', password: '' },
60     validate
61   );
62
63   const handleSubmit = (e) => {
64     e.preventDefault();
65     form.handleSubmit(async (values) => {
66       console.log('Login:', values);
67       // API call...
68     });
69   };
70
71   return (
72     <form onSubmit={handleSubmit}>
73       <input
74         name="email"
75         value={form.values.email}
76         onChange={form.handleChange}
77         onBlur={form.handleBlur}
78       />
79       {form.touched.email && form.errors.email && (
80         <span>{form.errors.email}</span>
81       )}
82
83       <input
84         type="password"
85         name="password"
86         value={form.values.password}
87         onChange={form.handleChange}
88         onBlur={form.handleBlur}
89       />
90       {form.touched.password && form.errors.password && (
91         <span>{form.errors.password}</span>
92       )}
93
94       <button type="submit" disabled={form.isSubmitting}>
95         Login
96       </button>
97     </form>
98   );
99 }

```

## 5.10 Best Practices

### 5.10.1 1. Naming Convention

Listing 5.27: Naming dei custom hooks

```
1 //      Inizia sempre con "use"
2 function useCounter() {}
3 function useFetch() {}
4 function useLocalStorage() {}
5
6 //      Non usare "use" per funzioni normali
7 function useHelper() {} // Se non è un hook, non chiamarlo "use..."
8 function formatDate() {} //      Meglio
```

### 5.10.2 2. Dependency Array

Listing 5.28: Gestione dependencies

```
1 function Component({ userId }) {
2   const [user, setUser] = useState(null);
3
4   //      Missing dependency
5   useEffect(() => {
6     fetchUser(userId).then(setUser);
7   }, []); // userId dovrebbe essere nelle deps!
8
9   //      Include tutte le dependencies
10  useEffect(() => {
11    fetchUser(userId).then(setUser);
12  }, [userId]);
13
14  //      Oppure usa ESLint plugin
15  // eslint-plugin-react-hooks ti avviserà
16 }
```

### 5.10.3 3. Evita Premature Optimization

Listing 5.29: Non abusare di useMemo/useCallback

```
1 //      Overuse di useMemo
2 function Component() {
3   const value = useMemo(() => 2 + 2, []); // Inutile!
4   const name = useMemo(() => 'Mario', []); // Inutile!
5
6   //      Usa solo per calcoli costosi
7   const expensiveValue = useMemo(() => {
8     return data.reduce((acc, item) => {
9       // Calcolo complesso...
10     }, 0);
11   }, [data]);
12 }
```

## 5.11 Conclusione

In questo capitolo abbiamo esplorato:

- useState per state management
- useEffect per side effects e lifecycle

- useContext per condividere dati globalmente
- useRef per riferimenti mutabili
- useMemo per memoizzazione valori
- useCallback per memoizzazione funzioni
- Custom hooks per logica riutilizzabile
- Best practices e pattern avanzati

Ora sei pronto per gestire eventi e interazioni utente!



# Capitolo 6

## Event Handling

### 6.1 Eventi in React

React gestisce gli eventi in modo simile al DOM nativo, ma con alcune differenze importanti. Gli eventi in React sono chiamati "Synthetic Events" (eventi sintetici).

#### 6.1.1 Differenze tra Eventi React e DOM

Listing 6.1: React vs DOM eventi

```
1 //      DOM nativo (HTML)
2 <button onclick="handleClick()">Click me</button>
3
4 //      React (JSX)
5 <button onClick={handleClick}>Click me</button>
6
7 // Differenze principali:
8 // 1. camelCase invece di lowercase (onClick vs onclick)
9 // 2. Funzione invece di stringa ({handleClick} vs "handleClick()")
10 // 3. preventDefault() invece di return false
```

Aspetto	DOM	React
Naming	lowercase	camelCase
Handler	Stringa	Funzione
Prevent default	return false	e.preventDefault()
Event object	Event nativo	SyntheticEvent

### 6.2 Synthetic Events

React wrappa gli eventi nativi del browser in un SyntheticEvent cross-browser.

#### 6.2.1 Cos'è un SyntheticEvent?

Listing 6.2: SyntheticEvent structure

```
1 function Component() {
2   const handleClick = (event) => {
3     // event è un SyntheticEvent
4     console.log(event);
5
6     // Proprietà comuni
```

```

7      console.log(event.type);           // "click"
8      console.log(event.target);         // Elemento DOM che ha
          generato l'evento
9      console.log(event.currentTarget);   // Elemento con l'handler
10     console.log(event.timeStamp);        // Timestamp
11     console.log(event.bubbles);          // true/false
12
13     // Metodi
14     event.preventDefault(); // Previene comportamento default
15     event.stopPropagation(); // Ferma propagazione
16
17     // Accesso all'evento nativo
18     const nativeEvent = event.nativeEvent;
19     console.log(nativeEvent);
20 };
21
22     return <button onClick={handleClick}>Click</button>;
23 }

```

## 6.2.2 Event Pooling (React < 17)

Listing 6.3: Event pooling in React 16

```

1 // In React 16 e precedenti, gli eventi venivano "pooled"
2 function OldComponent() {
3     const handleClick = (event) => {
4         console.log(event.type); // "click"
5
6         setTimeout(() => {
7             // event è null qui!
8             console.log(event.type); // null
9         }, 1000);
10
11         // Soluzione: persist()
12         event.persist();
13         setTimeout(() => {
14             console.log(event.type); // "click"
15         }, 1000);
16     };
17
18     return <button onClick={handleClick}>Click</button>;
19 }
20
21 // React 17+: Event pooling rimosso!
22 function NewComponent() {
23     const handleClick = (event) => {
24         setTimeout(() => {
25             // Funziona in React 17+
26             console.log(event.type); // "click"
27         }, 1000);
28     };
29
30     return <button onClick={handleClick}>Click</button>;
31 }

```

## 6.3 onClick: Eventi Click

onClick è l'evento più comune in React.

### 6.3.1 onClick Base

Listing 6.4: onClick esempi

```
1 function ClickExamples() {
2   // 1. Inline arrow function
3   return (
4     <button onClick={() => console.log('Clicked!')}>
5       Click 1
6     </button>
7   );
8
9   // 2. Named function
10  const handleClick = () => {
11    console.log('Clicked!');
12  };
13
14  return <button onClick={handleClick}>Click 2</button>;
15
16  // 3. Con parametri
17  const handleClickWithParam = (id) => {
18    console.log('Clicked item:', id);
19  };
20
21  return (
22    <button onClick={() => handleClickWithParam(123)}>
23      Click 3
24    </button>
25  );
26
27  // 4. Con event object
28  const handleClickWithEvent = (e) => {
29    console.log('Clicked:', e.target);
30  };
31
32  return <button onClick={handleClickWithEvent}>Click 4</button>;
33
34  // 5. Con event e parametri
35  const handleClickBoth = (id, e) => {
36    console.log('ID:', id, 'Event:', e);
37  };
38
39  return (
40    <button onClick={(e) => handleClickBoth(123, e)}>
41      Click 5
42    </button>
43  );
44 }
```

### 6.3.2 onClick Patterns

Listing 6.5: Pattern comuni onClick

```

1 function ClickPatterns() {
2   const [count, setCount] = useState(0);
3   const [items, setItems] = useState(['A', 'B', 'C']);
4
5   // Pattern 1: Aggiornamento state
6   const increment = () => {
7     setCount(count + 1);
8   };
9
10  // Pattern 2: Toggle boolean
11  const [isOpen, setIsOpen] = useState(false);
12  const toggle = () => setIsOpen(!isOpen);
13
14  // Pattern 3: Handler con parametro da lista
15  const handleItemClick = (item) => {
16    console.log('Clicked:', item);
17  };
18
19  // Pattern 4: Prevent default
20  const handleLinkClick = (e) => {
21    e.preventDefault();
22    console.log('Link clicked but not followed');
23  };
24
25  // Pattern 5: Conditional handler
26  const handleConditionalClick = () => {
27    if (count < 10) {
28      setCount(count + 1);
29    } else {
30      alert('Limite raggiunto!');
31    }
32  };
33
34  return (
35    <div>
36      {/* Pattern 1 */}
37      <button onClick={increment}>Count: {count}</button>
38
39      {/* Pattern 2 */}
40      <button onClick={toggle}>
41        {isOpen ? 'Chiudi' : 'Apri'}
42      </button>
43
44      {/* Pattern 3 */}
45      {items.map(item => (
46        <button key={item} onClick={() => handleItemClick(item)}
47          {item}
48        </button>
49      ))}
50
51      {/* Pattern 4 */}
52      <a href="https://example.com" onClick={handleLinkClick}>
53        Link
54      </a>
55
56      {/* Pattern 5 */}
57      <button onClick={handleConditionalClick}>

```



```
58         Incrementa (max 10)
59     </button>
60 </div>
61 );
62 }
```

### 6.3.3 Gestione Click Multipli

Listing 6.6: Double click e timing

```
1 function MultiClickExample() {
2     const [clicks, setClicks] = useState(0);
3     const [lastClick, setLastClick] = useState(0);
4
5     // Single click
6     const handleClick = () => {
7         setClicks(prev => prev + 1);
8     };
9
10    // Double click
11    const handleDoubleClick = () => {
12        console.log('Double clicked!');
13        setClicks(0); // Reset
14    };
15
16    // Debounced click
17    const timeoutRef = useRef(null);
18
19    const handleDebouncedClick = () => {
20        clearTimeout(timeoutRef.current);
21
22        timeoutRef.current = setTimeout(() => {
23            console.log('Clicked (debounced)');
24        }, 300);
25    };
26
27    // Click con delay detection
28    const handleClickWithDelay = () => {
29        const now = Date.now();
30        const timeSinceLastClick = now - lastClick;
31
32        if (timeSinceLastClick < 300) {
33            console.log('Quick click!');
34        } else {
35            console.log('Slow click!');
36        }
37
38        setLastClick(now);
39    };
40
41    return (
42        <div>
43            <button onClick={handleClick} onDoubleClick={
44                handleDoubleClick}>
45                Clicks: {clicks} (double click to reset)
46            </button>
```

```

47         <button onClick={handleDebounceClick}>
48             Debounced Click
49         </button>
50
51         <button onClick={handleClickWithDelay}>
52             Click with timing
53         </button>
54     </div>
55     );
56 }

```

## 6.4 onChange: Eventi Input

onChange è usato per gestire cambiamenti nei form elements.

### 6.4.1 onChange con Input

Listing 6.7: onChange input types

```

1 function InputExamples() {
2     const [text, setText] = useState('');
3     const [number, setNumber] = useState(0);
4     const [email, setEmail] = useState('');
5     const [password, setPassword] = useState('');
6
7     return (
8         <div>
9             {/* Text input */}
10            <input
11                type="text"
12                value={text}
13                onChange={(e) => setText(e.target.value)}
14                placeholder="Nome"
15            />
16            <p>Testo: {text}</p>
17
18            {/* Number input */}
19            <input
20                type="number"
21                value={number}
22                onChange={(e) => setNumber(Number(e.target.value))}
23                placeholder="Età"
24            />
25            <p>Numero: {number}</p>
26
27            {/* Email input con validazione */}
28            <input
29                type="email"
30                value={email}
31                onChange={(e) => setEmail(e.target.value)}
32                placeholder="Email"
33            />
34            <p style={{ color: email.includes('@') ? 'green' : 'red' }}>
35                Email: {email}
36            </p>
37        </div>
38    );
39 }

```

```

38         {/* Password input */}
39         <input
40             type="password"
41             value={password}
42             onChange={(e) => setPassword(e.target.value)}
43             placeholder="Password"
44         />
45         <p>Password length: {password.length}</p>
46     </div>
47 );
48 }

```

### 6.4.2 onChange con Select

Listing 6.8: onChange select

```

1  function SelectExamples() {
2      const [country, setCountry] = useState('');
3      const [language, setLanguage] = useState('it');
4      const [categories, setCategories] = useState([]);
5
6      return (
7          <div>
8              {/* Select singolo */}
9              <select value={country} onChange={(e) => setCountry(e.target
10                  .value)}>
11                  <option value="">Seleziona paese...</option>
12                  <option value="IT">Italia</option>
13                  <option value="US">USA</option>
14                  <option value="UK">UK</option>
15              </select>
16              <p>Paese selezionato: {country}</p>
17
18              {/* Select con default */}
19              <select value={language} onChange={(e) => setLanguage(e.
20                  target.value)}>
21                  <option value="it">Italiano</option>
22                  <option value="en">English</option>
23                  <option value="es">Español</option>
24              </select>
25              <p>Lingua: {language}</p>
26
27              {/* Select multiplo */}
28              <select
29                  multiple
30                  value={categories}
31                  onChange={(e) => {
32                      const selected = Array.from(
33                          e.target.selectedOptions,
34                          option => option.value
35                      );
36                      setCategories(selected);
37                  }}
38              >
39                  <option value="tech">Tech</option>
40                  <option value="sport">Sport</option>
41                  <option value="music">Musica</option>

```

```

40         <option value="travel">Viaggi</option>
41     </select>
42     <p>Categorie: {categories.join(', ')}</p>
43 </div>
44 );
45 }

```

### 6.4.3 onChange con Checkbox e Radio

Listing 6.9: onChange checkbox/radio

```

1 function CheckboxRadioExamples() {
2     const [isChecked, setIsChecked] = useState(false);
3     const [gender, setGender] = useState('');
4     const [interests, setInterests] = useState([]);
5
6     // Checkbox singola
7     const handleCheckboxChange = (e) => {
8         setIsChecked(e.target.checked);
9     };
10
11    // Radio buttons
12    const handleRadioChange = (e) => {
13        setGender(e.target.value);
14    };
15
16    // Checkbox multiple
17    const handleInterestChange = (interest) => {
18        setInterests(prev =>
19            prev.includes(interest)
20                ? prev.filter(i => i !== interest)
21                : [...prev, interest]
22        );
23    };
24
25    return (
26        <div>
27            {/* Checkbox singola */}
28            <label>
29                <input
30                    type="checkbox"
31                    checked={isChecked}
32                    onChange={handleCheckboxChange}
33                />
34                Accetto i termini
35            </label>
36            <p>Accettato: {isChecked ? 'Si' : 'No'}</p>
37
38            {/* Radio buttons */}
39            <div>
40                <label>
41                    <input
42                        type="radio"
43                        name="gender"
44                        value="male"
45                        checked={gender === 'male'}
46                        onChange={handleRadioChange}

```

```

47         />
48         Maschio
49     </label>
50     <label>
51         <input
52             type="radio"
53             name="gender"
54             value="female"
55             checked={gender === 'female'}
56             onChange={handleRadioChange}
57         />
58         Femmina
59     </label>
60 </div>
61 <p>Genere: {gender}</p>
62
63 {/* Checkbox multiple */}
64 <div>
65     {['Sport', 'Musica', 'Lettura', 'Viaggi'].map(interest
66         => (
67             <label key={interest}>
68                 <input
69                     type="checkbox"
70                     checked={interests.includes(interest)}
71                     onChange={() => handleInterestChange(
72                         interest)}
73                 {interest}
74             </label>
75         ))}
76 </div>
77 <p>Interessi: {interests.join(', ')}</p>
78 </div>
79 );

```

#### 6.4.4 onChange con Textarea

Listing 6.10: onChange textarea

```

1 function TextareaExample() {
2     const [message, setMessage] = useState('');
3     const maxLength = 200;
4
5     const handleChange = (e) => {
6         const value = e.target.value;
7
8         // Limita lunghezza
9         if (value.length <= maxLength) {
10             setMessage(value);
11         }
12     };
13
14     const remainingChars = maxLength - message.length;
15
16     return (
17         <div>

```

```

18         <textarea
19             value={message}
20             onChange={handleChange}
21             placeholder="Scrivi un messaggio..."
22             rows={5}
23             cols={50}
24         />
25
26         <p>
27             Caratteri rimanenti: {remainingChars} / {maxLength}
28         </p>
29
30         <p>Righe: {message.split('\n').length}</p>
31
32         <p style={{ color: remainingChars < 20 ? 'red' : 'black' }}>
33             {remainingChars < 20 && 'Stai raggiungendo il limite!'}
34         </p>
35     </div>
36 );
37 }

```

## 6.5 Altri Eventi Comuni

### 6.5.1 onSubmit: Form Submission

Listing 6.11: onSubmit form

```

1 function FormSubmitExample() {
2     const [formData, setFormData] = useState({
3         username: '',
4         email: '',
5         password: ''
6     });
7
8     const handleChange = (e) => {
9         const { name, value } = e.target;
10        setFormData(prev => ({
11            ...prev,
12            [name]: value
13        }));
14    };
15
16    const handleSubmit = (e) => {
17        e.preventDefault(); //             IMPORTANTE: Previeni reload pagina
18
19        console.log('Form submitted:', formData);
20
21        // Validazione
22        if (!formData.username || !formData.email || !formData.password) {
23            alert('Compila tutti i campi!');
24            return;
25        }
26
27        // Invia dati
28        fetch('/api/register', {
29            method: 'POST',

```

```

30         headers: { 'Content-Type': 'application/json' },
31         body: JSON.stringify(formData)
32     })
33     .then(response => response.json())
34     .then(data => {
35         console.log('Success:', data);
36         // Reset form
37         setFormData({ username: '', email: '', password: '' });
38     });
39 };
40
41 return (
42     <form onSubmit={handleSubmit}>
43         <input
44             name="username"
45             value={formData.username}
46             onChange={handleChange}
47             placeholder="Username"
48         />
49
50         <input
51             name="email"
52             type="email"
53             value={formData.email}
54             onChange={handleChange}
55             placeholder="Email"
56         />
57
58         <input
59             name="password"
60             type="password"
61             value={formData.password}
62             onChange={handleChange}
63             placeholder="Password"
64         />
65
66         <button type="submit">Registrati</button>
67     </form>
68 );
69 }

```

### 6.5.2 onFocus e onBlur

Listing 6.12: onFocus e onBlur

```

1 function FocusBlurExample() {
2     const [isFocused, setIsFocused] = useState(false);
3     const [value, setValue] = useState('');
4     const [touched, setTouched] = useState(false);
5
6     const handleFocus = () => {
7         setIsFocused(true);
8     };
9
10    const handleBlur = () => {
11        setIsFocused(false);
12        setTouched(true);

```

```

13     };
14
15     // Validazione solo dopo blur
16     const error = touched && value.length < 3
17       ? 'Minimo 3 caratteri'
18       : '';
19
20     return (
21       <div>
22         <input
23           value={value}
24           onChange={(e) => setValue(e.target.value)}
25           onFocus={handleFocus}
26           onBlur={handleBlur}
27           placeholder="Nome"
28           style={{
29             borderColor: isFocused ? 'blue' : error ? 'red' : '
30             gray'
31           }}
32         />
33
34         <p>Focus: {isFocused ? 'Si' : 'No'}</p>
35
36         {error && <span style={{ color: 'red' }}>{error}</span>}
37       </div>
38     );
39   }

```

### 6.5.3 onKeyDown, onKeyUp, onKeyPress

Listing 6.13: Eventi keyboard

```

1 function KeyboardEvents() {
2   const [input, setInput] = useState('');
3   const [lastKey, setLastKey] = useState('');
4
5   const handleKeyDown = (e) => {
6     setLastKey(e.key);
7
8     // Enter per submit
9     if (e.key === 'Enter') {
10       console.log('Enter pressed!');
11     }
12
13     // Escape per cancellare
14     if (e.key === 'Escape') {
15       setInput('');
16     }
17
18     // Ctrl + S per salvare
19     if (e.ctrlKey && e.key === 's') {
20       e.preventDefault();
21       console.log('Saving...');
22     }
23
24     // Arrow keys
25     if (e.key === 'ArrowUp') {

```



```

26         console.log('Up arrow');
27     }
28 };
29
30 const handleKeyPress = (e) => {
31     // Solo caratteri alfanumerici
32     if (!/[a-zA-Z0-9]/.test(e.key)) {
33         e.preventDefault();
34     }
35 };
36
37 return (
38     <div>
39         <input
40             value={input}
41             onChange={(e) => setInput(e.target.value)}
42             onKeyDown={handleKeyDown}
43             onKeyPress={handleKeyPress}
44             placeholder="Prova i tasti..."
45         />
46
47         <p>Ultimo tasto: {lastKey}</p>
48         <p>Input: {input}</p>
49
50         <ul>
51             <li>Enter: Submit</li>
52             <li>Escape: Cancella</li>
53             <li>Ctrl+S: Salva</li>
54         </ul>
55     </div>
56 );
57 }

```

#### 6.5.4 onMouseEnter e onMouseLeave

Listing 6.14: Eventi mouse hover

```

1 function HoverExample() {
2     const [isHovering, setIsHovering] = useState(false);
3     const [hoverCount, setHoverCount] = useState(0);
4     const [position, setPosition] = useState({ x: 0, y: 0 });
5
6     const handleMouseEnter = () => {
7         setIsHovering(true);
8         setHoverCount(prev => prev + 1);
9     };
10
11     const handleMouseLeave = () => {
12         setIsHovering(false);
13     };
14
15     const handleMouseMove = (e) => {
16         // Posizione relativa all'elemento
17         const rect = e.currentTarget.getBoundingClientRect();
18         setPosition({
19             x: e.clientX - rect.left,
20             y: e.clientY - rect.top

```

```

21     });
22 };
23
24 return (
25     <div>
26         <div>
27             onMouseEnter={handleMouseEnter}
28             onMouseLeave={handleMouseLeave}
29             onMouseMove={handleMouseMove}
30             style={{
31                 width: '300px',
32                 height: '200px',
33                 backgroundColor: isHovering ? 'lightblue' : '
                 lightgray',
34                 display: 'flex',
35                 alignItems: 'center',
36                 justifyContent: 'center',
37                 cursor: 'pointer',
38                 transition: 'background-color 0.3s'
39             }}
40         >
41             <div>
42                 <p>Hovering: {isHovering ? 'Si' : 'No'}</p>
43                 <p>Hover count: {hoverCount}</p>
44                 <p>Position: ({position.x.toFixed(0)}, {position.y.
                     toFixed(0)})</p>
45             </div>
46         </div>
47     </div>
48 );
49 }

```

### 6.5.5 onScroll

Listing 6.15: Evento scroll

```

1 function ScrollExample() {
2     const [scrollPosition, setScrollPosition] = useState(0);
3     const [isTop, setIsTop] = useState(true);
4     const scrollRef = useRef(null);
5
6     const handleScroll = (e) => {
7         const scrollTop = e.target.scrollTop;
8         setScrollPosition(scrollTop);
9         setIsTop(scrollTop === 0);
10    };
11
12    // Window scroll
13    useEffect(() => {
14        const handleWindowScroll = () => {
15            setScrollPosition(window.scrollY);
16        };
17
18        window.addEventListener('scroll', handleWindowScroll);
19        return () => window.removeEventListener('scroll',
            handleWindowScroll);
20    }, []);

```

```

21
22     return (
23         <div>
24             {/* Scrollable container */}
25             <div
26                 ref={scrollRef}
27                 onScroll={handleScroll}
28                 style={{
29                     height: '300px',
30                     overflow: 'auto',
31                     border: '1px solid black'
32                 }}
33             >
34                 <div style={{ height: '1000px', padding: '20px' }}>
35                     <p>Scroll position: {scrollTop}px</p>
36                     <p>At top: {isTop ? 'Si' : 'No'}</p>
37                     <p>Contenuto scrollabile...</p>
38                 </div>
39             </div>
40         </div>
41     );
42 }

```

## 6.6 Event Delegation e Bubbling

### 6.6.1 Event Bubbling

Listing 6.16: Event bubbling

```

1 function BubblingExample() {
2     const handleParentClick = (e) => {
3         console.log('Parent clicked');
4         console.log('Target:', e.target.tagName);           // Elemento
5         console.log('CurrentTarget:', e.currentTarget.tagName); //
6         console.log('Elemento con handler');
7     };
8
9     const handleChildClick = (e) => {
10         console.log('Child clicked');
11
12         // Ferma la propagazione
13         // e.stopPropagation();
14     };
15
16     return (
17         <div
18             onClick={handleParentClick}
19             style={{ padding: '20px', backgroundColor: 'lightblue' }}
20         >
21             Parent
22             <button onClick={handleChildClick}>
23                 Child (clicca qui)
24             </button>
25         </div>
26     );
27     // Se clicchi il button, vedi:

```

```

27 // "Child clicked"
28 // "Parent clicked" (bubbling!)
29 }

```

### 6.6.2 Event Delegation

Listing 6.17: Event delegation pattern

```

1 function TodoList() {
2   const [todos, setTodos] = useState([
3     { id: 1, text: 'Todo 1' },
4     { id: 2, text: 'Todo 2' },
5     { id: 3, text: 'Todo 3' }
6   ]);
7
8   // Handler per ogni elemento (non scalabile)
9   const BadApproach = () => (
10     <ul>
11       {todos.map(todo => (
12         <li key={todo.id}>
13           {todo.text}
14           <button onClick={() => deleteTodo(todo.id)}>Delete</
15             button>
16         </li>
17       )}}
18     </ul>
19   );
20
21   // Event delegation (un solo handler)
22   const handleListClick = (e) => {
23     // Verifica se è un button delete
24     if (e.target.classList.contains('delete-btn')) {
25       const id = Number(e.target.dataset.id);
26       deleteTodo(id);
27     }
28   };
29
30   const deleteTodo = (id) => {
31     setTodos(todos.filter(todo => todo.id !== id));
32   };
33
34   return (
35     <ul onClick={handleListClick}>
36       {todos.map(todo => (
37         <li key={todo.id}>
38           {todo.text}
39           <button
40             className="delete-btn"
41             data-id={todo.id}
42             >
43             Delete
44           </button>
45         </li>
46       )}}
47     </ul>
48   );

```

## 6.7 Custom Events

### 6.7.1 Eventi Personalizzati

Listing 6.18: Custom events pattern

```
1 function CustomEventExample() {
2   const handleCustomEvent = (data) => {
3     console.log('Custom event:', data);
4   };
5
6   return (
7     <div>
8       <CustomButton onCustomClick={handleCustomEvent} />
9     </div>
10  );
11 }
12
13 function CustomButton({ onCustomClick }) {
14   const handleClick = (e) => {
15     // Raccogli dati aggiuntivi
16     const data = {
17       timestamp: Date.now(),
18       position: {
19         x: e.clientX,
20         y: e.clientY
21       },
22       button: e.button
23     };
24
25     // Chiama callback con dati custom
26     onCustomClick?.(data);
27   };
28
29   return <button onClick={handleClick}>Click me</button>;
30 }
```

## 6.8 Performance e Best Practices

### 6.8.1 1. Evita Inline Functions nei Render

Listing 6.19: Ottimizzazione event handlers

```
1 function TodoItem({ todo, onToggle, onDelete }) {
2   // MALE: Crea nuova funzione ad ogni render
3   return (
4     <li>
5       <input
6         type="checkbox"
7         onChange={() => onToggle(todo.id)}
8       />
9       <button onClick={() => onDelete(todo.id)}>Delete</button>
10    </li>
11  );
12 }
13
14 // MEGLIO: useCallback
```

```

15 function TodoList() {
16   const [todos, setTodos] = useState([]);
17
18   const handleToggle = useCallback((id) => {
19     setTodos(prev => prev.map(todo =>
20       todo.id === id ? { ...todo, completed: !todo.completed } :
21       todo
22     ));
23
24   const handleDelete = useCallback((id) => {
25     setTodos(prev => prev.filter(todo => todo.id !== id));
26   }, []);
27
28   return (
29     <ul>
30       {todos.map(todo => (
31         <TodoItem
32           key={todo.id}
33           todo={todo}
34           onToggle={handleToggle}
35           onDelete={handleDelete}
36         />
37       ))}
38     </ul>
39   );
40 }

```

## 6.8.2 2. Debouncing e Throttling

Listing 6.20: Debounce e throttle

```

1 // Debounce: Esegui dopo che l'utente smette di digitare
2 function SearchWithDebounce() {
3   const [query, setQuery] = useState('');
4   const [results, setResults] = useState([]);
5
6   useEffect(() => {
7     const timer = setTimeout(() => {
8       if (query) {
9         fetch('/api/search?q=${query}')
10          .then(res => res.json())
11          .then(setResults);
12       }
13     }, 500); // Aspetta 500ms dopo l'ultimo input
14
15     return () => clearTimeout(timer);
16   }, [query]);
17
18   return (
19     <div>
20       <input
21         value={query}
22         onChange={(e) => setQuery(e.target.value)}
23         placeholder="Cerca..."
24       />
25     </div>

```

```

26         {results.map(result => (
27             <li key={result.id}>{result.name}</li>
28         ))}
29     </ul>
30 </div>
31 );
32 }
33
34 // Throttle: Esegui al massimo una volta ogni X ms
35 function ScrollProgress() {
36     const [progress, setProgress] = useState(0);
37     const throttleRef = useRef(null);
38
39     useEffect(() => {
40         const handleScroll = () => {
41             if (throttleRef.current) return;
42
43             throttleRef.current = setTimeout(() => {
44                 const winScroll = document.documentElement.scrollTop;
45                 const height = document.documentElement.scrollHeight -
46                     document.documentElement.clientHeight;
47                 const scrolled = (winScroll / height) * 100;
48
49                 setProgress(scrolled);
50                 throttleRef.current = null;
51             }, 100); // Max una volta ogni 100ms
52         };
53
54         window.addEventListener('scroll', handleScroll);
55         return () => window.removeEventListener('scroll', handleScroll);
56     }, []);
57
58     return (
59         <div style={{
60             position: 'fixed',
61             top: 0,
62             left: 0,
63             width: `${progress}%`,
64             height: '4px',
65             backgroundColor: 'blue'
66         }} />
67     );
68 }

```

### 6.8.3 3. Passive Event Listeners

Listing 6.21: Passive listeners per performance

```

1 function PassiveListenerExample() {
2     useEffect(() => {
3         const handleTouchMove = (e) => {
4             // Non chiama preventDefault()
5             console.log('Touch move');
6         };
7
8         // Passive listener per scroll performance
9         document.addEventListener(

```

```
10         'touchmove',
11         handleTouchMove,
12         { passive: true }
13     );
14
15     return () => {
16         document.removeEventListener('touchmove', handleTouchMove);
17     };
18 }, []);
19
20 return <div>Swipe qui</div>;
21 }
```

## 6.9 Conclusione

In questo capitolo abbiamo esplorato:

- Synthetic Events in React
- onClick per gestione click
- onChange per input e form
- Altri eventi: onSubmit, onFocus, onBlur, keyboard, mouse
- Event bubbling e delegation
- Performance optimization: debounce, throttle, useCallback
- Best practices per event handling

Nel prossimo capitolo approfondiremo i form!



# Capitolo 7

## Form in React

### 7.1 Introduzione ai Form

I form sono una parte essenziale di quasi tutte le applicazioni web. React offre un approccio potente e flessibile per gestire form attraverso i "controlled components".

#### 7.1.1 Controlled vs Uncontrolled Components

Listing 7.1: Controlled vs Uncontrolled

```
1 //      Uncontrolled: Il DOM gestisce lo stato
2 function UncontrolledForm() {
3     const nameRef = useRef();
4
5     const handleSubmit = (e) => {
6         e.preventDefault();
7         console.log('Name:', nameRef.current.value);
8     };
9
10    return (
11        <form onSubmit={handleSubmit}>
12            <input ref={nameRef} defaultValue="Mario" />
13            <button type="submit">Submit</button>
14        </form>
15    );
16 }
17
18 //      Controlled: React gestisce lo stato
19 function ControlledForm() {
20     const [name, setName] = useState('Mario');
21
22     const handleSubmit = (e) => {
23         e.preventDefault();
24         console.log('Name:', name);
25     };
26
27    return (
28        <form onSubmit={handleSubmit}>
29            <input
30                value={name}
31                onChange={(e) => setName(e.target.value)}
32            />
33            <button type="submit">Submit</button>
34        </form>
```

```

35     );
36 }

```

### Vantaggi Controlled Components:

- Validazione in tempo reale
- Formattazione automatica dell'input
- Disabilitazione condizionale del submit
- Stato sincronizzato con React
- Più facile da testare

## 7.2 Form Base Controllati

### 7.2.1 Form con Input Multipli

Listing 7.2: Form completo controllato

```

1 function RegistrationForm() {
2   const [formData, setFormData] = useState({
3     username: '',
4     email: '',
5     password: '',
6     confirmPassword: '',
7     age: '',
8     country: '',
9     gender: '',
10    terms: false,
11    newsletter: false
12  });
13
14  // Handler generico per tutti gli input
15  const handleChange = (e) => {
16    const { name, value, type, checked } = e.target;
17
18    setFormData(prev => ({
19      ...prev,
20      [name]: type === 'checkbox' ? checked : value
21    }));
22  };
23
24  const handleSubmit = (e) => {
25    e.preventDefault();
26    console.log('Form data:', formData);
27  };
28
29  return (
30    <form onSubmit={handleSubmit}>
31      {/* Text Input */}
32      <div>
33        <label htmlFor="username">Username:</label>
34        <input
35          id="username"
36          type="text"
37          name="username"

```

```

38         value={formData.username}
39         onChange={handleChange}
40         required
41     />
42 </div>
43
44 {/* Email Input */}
45 <div>
46     <label htmlFor="email">Email:</label>
47     <input
48         id="email"
49         type="email"
50         name="email"
51         value={formData.email}
52         onChange={handleChange}
53         required
54     />
55 </div>
56
57 {/* Password Input */}
58 <div>
59     <label htmlFor="password">Password:</label>
60     <input
61         id="password"
62         type="password"
63         name="password"
64         value={formData.password}
65         onChange={handleChange}
66         required
67     />
68 </div>
69
70 {/* Confirm Password */}
71 <div>
72     <label htmlFor="confirmPassword">Conferma Password:</
73     label>
74     <input
75         id="confirmPassword"
76         type="password"
77         name="confirmPassword"
78         value={formData.confirmPassword}
79         onChange={handleChange}
80         required
81     />
82 </div>
83
84 {/* Number Input */}
85 <div>
86     <label htmlFor="age">Età:</label>
87     <input
88         id="age"
89         type="number"
90         name="age"
91         value={formData.age}
92         onChange={handleChange}
93         min="18"
94         max="120"
95     />

```

```

95     </div>
96
97     {/* Select */}
98     <div>
99         <label htmlFor="country">Paese:</label>
100         <select
101             id="country"
102             name="country"
103             value={formData.country}
104             onChange={handleChange}
105         >
106             <option value="">Seleziona...</option>
107             <option value="IT">Italia</option>
108             <option value="US">USA</option>
109             <option value="UK">UK</option>
110             <option value="FR">Francia</option>
111         </select>
112     </div>
113
114     {/* Radio Buttons */}
115     <div>
116         <label>Genere:</label>
117         <label>
118             <input
119                 type="radio"
120                 name="gender"
121                 value="male"
122                 checked={formData.gender === 'male'}
123                 onChange={handleChange}
124             />
125             Maschio
126         </label>
127         <label>
128             <input
129                 type="radio"
130                 name="gender"
131                 value="female"
132                 checked={formData.gender === 'female'}
133                 onChange={handleChange}
134             />
135             Femmina
136         </label>
137         <label>
138             <input
139                 type="radio"
140                 name="gender"
141                 value="other"
142                 checked={formData.gender === 'other'}
143                 onChange={handleChange}
144             />
145             Altro
146         </label>
147     </div>
148
149     {/* Checkboxes */}
150     <div>
151         <label>
152             <input

```

```

153         type="checkbox"
154         name="terms"
155         checked={formData.terms}
156         onChange={handleChange}
157     />
158     Accetto i termini e condizioni
159 </label>
160 </div>
161
162 <div>
163     <label>
164         <input
165             type="checkbox"
166             name="newsletter"
167             checked={formData.newsletter}
168             onChange={handleChange}
169         />
170         Iscriviti alla newsletter
171     </label>
172 </div>
173
174     {/ * Submit Button */}
175     <button type="submit">Registrati</button>
176
177     {/ * Debug */}
178     <pre>{JSON.stringify(formData, null, 2)}</pre>
179 </form>
180 );
181 }

```

## 7.3 Validazione Form

### 7.3.1 Validazione Base

Listing 7.3: Validazione manuale

```

1 function ValidatedForm() {
2     const [formData, setFormData] = useState({
3         username: '',
4         email: '',
5         password: '',
6         confirmPassword: ''
7     });
8
9     const [errors, setErrors] = useState({});
10    const [touched, setTouched] = useState({});
11
12    const handleChange = (e) => {
13        const { name, value } = e.target;
14        setFormData(prev => ({ ...prev, [name]: value }));
15
16        // Rimuovi errore quando l'utente inizia a digitare
17        if (errors[name]) {
18            setErrors(prev => ({ ...prev, [name]: '' }));
19        }
20    };
21 }

```

```

22  const handleBlur = (e) => {
23    const { name } = e.target;
24    setTouched(prev => ({ ...prev, [name]: true }));
25    validateField(name, formData[name]);
26  };
27
28  const validateField = (name, value) => {
29    let error = '';
30
31    switch (name) {
32      case 'username':
33        if (!value) {
34          error = 'Username è richiesto';
35        } else if (value.length < 3) {
36          error = 'Username deve essere almeno 3 caratteri';
37        } else if (!/^[a-zA-Z0-9_]+$/.test(value)) {
38          error = 'Username può contenere solo lettere, numeri
39              e underscore';
40        }
41        break;
42
43      case 'email':
44        if (!value) {
45          error = 'Email è richiesta';
46        } else if (!/^\S+@\S+\.\S+$/.test(value)) {
47          error = 'Email non valida';
48        }
49        break;
50
51      case 'password':
52        if (!value) {
53          error = 'Password è richiesta';
54        } else if (value.length < 8) {
55          error = 'Password deve essere almeno 8 caratteri';
56        } else if (!/(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/.test(value)
57            ) {
58          error = 'Password deve contenere maiuscola,
59              minuscola e numero';
60        }
61        break;
62
63      case 'confirmPassword':
64        if (!value) {
65          error = 'Conferma password è richiesta';
66        } else if (value !== formData.password) {
67          error = 'Le password non corrispondono';
68        }
69        break;
70    }
71
72    setErrors(prev => ({ ...prev, [name]: error }));
73    return error;
74  };
75
76  const validateForm = () => {
77    const newErrors = {};
78
79    Object.keys(formData).forEach(key => {

```

```

77         const error = validateField(key, formData[key]);
78         if (error) newErrors[key] = error;
79     });
80
81     setErrors(newErrors);
82     return Object.keys(newErrors).length === 0;
83 };
84
85 const handleSubmit = (e) => {
86     e.preventDefault();
87
88     // Mark all fields as touched
89     const allTouched = Object.keys(formData).reduce(
90         (acc, key) => ({ ...acc, [key]: true }),
91         {}
92     );
93     setTouched(allTouched);
94
95     if (validateForm()) {
96         console.log('Form valido:', formData);
97         // Submit...
98     } else {
99         console.log('Form non valido');
100     }
101 };
102
103 return (
104     <form onSubmit={handleSubmit}>
105         <div>
106             <label htmlFor="username">Username:</label>
107             <input
108                 id="username"
109                 type="text"
110                 name="username"
111                 value={formData.username}
112                 onChange={handleChange}
113                 onBlur={handleBlur}
114                 className={touched.username && errors.username ? '
115                     error' : ''}
116             />
117             {touched.username && errors.username && (
118                 <span className="error-message">{errors.username}</
119                 span>
120             )}
121         </div>
122
123         <div>
124             <label htmlFor="email">Email:</label>
125             <input
126                 id="email"
127                 type="email"
128                 name="email"
129                 value={formData.email}
130                 onChange={handleChange}
131                 onBlur={handleBlur}
132                 className={touched.email && errors.email ? 'error' :
133                     ''}
134             />

```

```

132         {touched.email && errors.email && (
133             <span className="error-message">{errors.email}</span>
134         )}
135     </div>
136
137     <div>
138         <label htmlFor="password">Password:</label>
139         <input
140             id="password"
141             type="password"
142             name="password"
143             value={formData.password}
144             onChange={handleChange}
145             onBlur={handleBlur}
146             className={touched.password && errors.password ? '
147                 error' : ''}
148             />
149         {touched.password && errors.password && (
150             <span className="error-message">{errors.password}</
151             span>
152         )}
153     </div>
154
155     <div>
156         <label htmlFor="confirmPassword">Conferma Password:</
157         label>
158         <input
159             id="confirmPassword"
160             type="password"
161             name="confirmPassword"
162             value={formData.confirmPassword}
163             onChange={handleChange}
164             onBlur={handleBlur}
165             className={touched.confirmPassword && errors.
166                 confirmPassword ? 'error' : ''}
167             />
168         {touched.confirmPassword && errors.confirmPassword && (
169             <span className="error-message">{errors.
170             confirmPassword}</span>
171         )}
172     </div>
173
174     <button type="submit">Registrati</button>
175 </form>
176 );
177 }

```

### 7.3.2 Custom Hook per Validazione

Listing 7.4: useForm custom hook

```

1 function useForm(initialValues, validate) {
2     const [values, setValues] = useState(initialValues);
3     const [errors, setErrors] = useState({});
4     const [touched, setTouched] = useState({});
5     const [isSubmitting, setIsSubmitting] = useState(false);

```



```
6
7  const handleChange = (e) => {
8    const { name, value, type, checked } = e.target;
9    const newValue = type === 'checkbox' ? checked : value;
10
11    setValues(prev => ({ ...prev, [name]: newValue }));
12
13    // Rimuovi errore
14    if (errors[name]) {
15      setErrors(prev => ({ ...prev, [name]: '' }));
16    }
17  };
18
19  const handleBlur = (e) => {
20    const { name } = e.target;
21    setTouched(prev => ({ ...prev, [name]: true }));
22  };
23
24  const handleSubmit = async (onSubmit) => {
25    // Mark all as touched
26    const allTouched = Object.keys(values).reduce(
27      (acc, key) => ({ ...acc, [key]: true }),
28      {}
29    );
30    setTouched(allTouched);
31
32    // Validate
33    const validationErrors = validate(values);
34    setErrors(validationErrors);
35
36    if (Object.keys(validationErrors).length === 0) {
37      setIsSubmitting(true);
38      try {
39        await onSubmit(values);
40      } finally {
41        setIsSubmitting(false);
42      }
43    }
44  };
45
46  const resetForm = () => {
47    setValues(initialValues);
48    setErrors({});
49    setTouched({});
50    setIsSubmitting(false);
51  };
52
53  return {
54    values,
55    errors,
56    touched,
57    isSubmitting,
58    handleChange,
59    handleBlur,
60    handleSubmit,
61    resetForm
62  };
63 }
```

```

64
65 // Uso
66 function LoginForm() {
67     const validate = (values) => {
68         const errors = {};
69
70         if (!values.email) {
71             errors.email = 'Email richiesta';
72         } else if (!/\S+@\S+\.\S+/.test(values.email)) {
73             errors.email = 'Email non valida';
74         }
75
76         if (!values.password) {
77             errors.password = 'Password richiesta';
78         } else if (values.password.length < 6) {
79             errors.password = 'Password troppo corta';
80         }
81
82         return errors;
83     };
84
85     const form = useForm(
86         { email: '', password: '' },
87         validate
88     );
89
90     const onSubmit = async (values) => {
91         console.log('Submitting:', values);
92         await new Promise(resolve => setTimeout(resolve, 1000));
93         console.log('Submitted!');
94         form.resetForm();
95     };
96
97     return (
98         <form onSubmit={(e) => {
99             e.preventDefault();
100             form.handleSubmit(onSubmit);
101         }}>
102             <div>
103                 <input
104                     name="email"
105                     type="email"
106                     value={form.values.email}
107                     onChange={form.handleChange}
108                     onBlur={form.handleBlur}
109                     placeholder="Email"
110                 />
111                 {form.touched.email && form.errors.email && (
112                     <span>{form.errors.email}</span>
113                 )}
114             </div>
115
116             <div>
117                 <input
118                     name="password"
119                     type="password"
120                     value={form.values.password}
121                     onChange={form.handleChange}

```

```

122         onBlur={form.handleBlur}
123         placeholder="Password"
124       />
125       {form.touched.password && form.errors.password && (
126         <span>{form.errors.password}</span>
127       )}
128     </div>
129
130     <button type="submit" disabled={form.isSubmitting}>
131       {form.isSubmitting ? 'Accedendo...' : 'Accedi'}
132     </button>
133   </form>
134 );
135 }

```

## 7.4 React Hook Form

React Hook Form è una libreria popolare per la gestione dei form con performance ottimali.

### 7.4.1 Setup React Hook Form

Listing 7.5: Installazione

```

1 npm install react-hook-form

```

### 7.4.2 Form Base con React Hook Form

Listing 7.6: React Hook Form esempio base

```

1 import { useForm } from 'react-hook-form';
2
3 function BasicForm() {
4   const {
5     register,
6     handleSubmit,
7     formState: { errors, isSubmitting },
8     reset
9   } = useForm({
10     defaultValues: {
11       username: '',
12       email: '',
13       password: ''
14     }
15   });
16
17   const onSubmit = async (data) => {
18     console.log('Form data:', data);
19     await new Promise(resolve => setTimeout(resolve, 1000));
20     console.log('Submitted!');
21     reset();
22   };
23
24   return (
25     <form onSubmit={handleSubmit(onSubmit)}>
26       {/* Username */}

```

```

27 <div>
28   <label htmlFor="username">Username:</label>
29   <input
30     id="username"
31     {...register('username', {
32       required: 'Username è richiesto',
33       minLength: {
34         value: 3,
35         message: 'Minimo 3 caratteri'
36       },
37       pattern: {
38         value: /^[a-zA-Z0-9_]+$/,
39         message: 'Solo lettere, numeri e underscore'
40       }
41     })}
42   />
43   {errors.username && (
44     <span className="error">{errors.username.message}</span>
45   )}
46 </div>
47
48 {/* Email */}
49 <div>
50   <label htmlFor="email">Email:</label>
51   <input
52     id="email"
53     type="email"
54     {...register('email', {
55       required: 'Email è richiesta',
56       pattern: {
57         value: /\S+@\S+\.\S+/,
58         message: 'Email non valida'
59       }
60     })}
61   />
62   {errors.email && (
63     <span className="error">{errors.email.message}</span>
64   )}
65 </div>
66
67 {/* Password */}
68 <div>
69   <label htmlFor="password">Password:</label>
70   <input
71     id="password"
72     type="password"
73     {...register('password', {
74       required: 'Password è richiesta',
75       minLength: {
76         value: 8,
77         message: 'Minimo 8 caratteri'
78       },
79       validate: {
80         hasUpperCase: (value) =>
81           /[A-Z]/.test(value) || 'Deve contenere
           maiuscola',

```

```

82         hasLowerCase: (value) =>
83             /[a-z]/.test(value) || 'Deve contenere
84                 minuscola',
85         hasNumber: (value) =>
86             /\d/.test(value) || 'Deve contenere un
87                 numero'
88     }
89     }
90     <div>
91         {errors.password && (
92             <span className="error">{errors.password.message}</
93             span>
94         )}
95     </div>
96
97     <button type="submit" disabled={isSubmitting}>
98         {isSubmitting ? 'Invio...' : 'Invia'}
99     </button>
100 </form>
101 );
102 }

```

### 7.4.3 Validazione Avanzata React Hook Form

Listing 7.7: Validazione custom

```

1  import { useForm } from 'react-hook-form';
2
3  function AdvancedForm() {
4      const { register, handleSubmit, watch, formState: { errors } } =
5          useForm();
6
7      // Watch per validazione dipendente
8      const password = watch('password');
9
10     const onSubmit = (data) => {
11         console.log(data);
12     };
13
14     return (
15         <form onSubmit={handleSubmit(onSubmit)}>
16             { /* Password */ }
17             <input
18                 type="password"
19                 {...register('password', {
20                     required: 'Password richiesta',
21                     minLength: { value: 8, message: 'Minimo 8 caratteri' }
22                 })}
23             />
24             {errors.password && <span>{errors.password.message}</span>}
25
26             { /* Confirm Password - dipende da password */ }
27             <input
28                 type="password"
29                 {...register('confirmPassword', {
30                     required: 'Conferma password richiesta',

```

```

30         validate: (value) =>
31             value === password || 'Le password non
               corrispondono'
32     }}}
33     />
34     {errors.confirmPassword && (
35         <span>{errors.confirmPassword.message}</span>
36     )}
37
38     {/* Age - validazione numerica */}
39     <input
40         type="number"
41         {...register('age', {
42             required: 'Età richiesta',
43             min: { value: 18, message: 'Devi avere almeno 18
               anni' },
44             max: { value: 120, message: 'Età non valida' },
45             valueAsNumber: true
46         })}
47     />
48     {errors.age && <span>{errors.age.message}</span>}
49
50     {/* Terms - checkbox */}
51     <label>
52         <input
53             type="checkbox"
54             {...register('terms', {
55                 required: 'Devi accettare i termini'
56             })}
57         />
58         Accetto i termini
59     </label>
60     {errors.terms && <span>{errors.terms.message}</span>}
61
62     <button type="submit">Invia</button>
63 </form>
64 );
65 }

```

#### 7.4.4 Form Arrays con React Hook Form

Listing 7.8: Dynamic form fields

```

1  import { useForm, useFieldArray } from 'react-hook-form';
2
3  function DynamicForm() {
4      const { register, control, handleSubmit } = useForm({
5          defaultValues: {
6              users: [{ name: '', email: '' }]
7          }
8      });
9
10     const { fields, append, remove } = useFieldArray({
11         control,
12         name: 'users'
13     });
14

```

```

15     const onSubmit = (data) => {
16         console.log('Users:', data.users);
17     };
18
19     return (
20         <form onSubmit={handleSubmit(onSubmit)}>
21             {fields.map((field, index) => (
22                 <div key={field.id}>
23                     <input
24                         {...register('users.${index}.name', {
25                             required: 'Nome richiesto'
26                         })}
27                     placeholder="Nome"
28                 />
29
30                     <input
31                         {...register('users.${index}.email', {
32                             required: 'Email richiesta',
33                             pattern: {
34                                 value: /\S+@\S+\.\S+/,
35                                 message: 'Email non valida'
36                             }
37                         })}
38                     placeholder="Email"
39                 />
40
41                 <button type="button" onClick={() => remove(index)}>
42                     Rimuovi
43                 </button>
44             </div>
45         )]}
46
47         <button
48             type="button"
49             onClick={() => append({ name: '', email: '' })}
50         >
51             Aggiungi Utente
52         </button>
53
54         <button type="submit">Invia</button>
55     </form>
56 );
57 }

```

## 7.5 Formik

Formik è un'altra libreria popolare per la gestione dei form in React.

### 7.5.1 Setup Formik

Listing 7.9: Installazione Formik

```

1 npm install formik yup

```

### 7.5.2 Form Base con Formik

Listing 7.10: Formik esempio base

```

1 import { Formik, Form, Field, ErrorMessage } from 'formik';
2 import * as Yup from 'yup';
3
4 // Schema validazione con Yup
5 const validationSchema = Yup.object({
6   username: Yup.string()
7     .min(3, 'Minimo 3 caratteri')
8     .required('Username richiesto'),
9   email: Yup.string()
10    .email('Email non valida')
11    .required('Email richiesta'),
12   password: Yup.string()
13    .min(8, 'Minimo 8 caratteri')
14    .matches(/[A-Z]/, 'Deve contenere maiuscola')
15    .matches(/[a-z]/, 'Deve contenere minuscola')
16    .matches(/\d/, 'Deve contenere numero')
17    .required('Password richiesta'),
18   confirmPassword: Yup.string()
19    .oneOf([Yup.ref('password')], 'Le password non corrispondono')
20    .required('Conferma password richiesta'),
21   age: Yup.number()
22    .min(18, 'Devi avere almeno 18 anni')
23    .max(120, 'Età non valida')
24    .required('Età richiesta'),
25   terms: Yup.boolean()
26    .oneOf([true], 'Devi accettare i termini')
27 });
28
29 function FormikExample() {
30   const initialValues = {
31     username: '',
32     email: '',
33     password: '',
34     confirmPassword: '',
35     age: '',
36     terms: false
37   };
38
39   const handleSubmit = async (values, { setSubmitting, resetForm }) => {
40     console.log('Form values:', values);
41     await new Promise(resolve => setTimeout(resolve, 1000));
42     console.log('Submitted!');
43     resetForm();
44     setSubmitting(false);
45   };
46
47   return (
48     <Formik
49       initialValues={initialValues}
50       validationSchema={validationSchema}
51       onSubmit={handleSubmit}
52     >
53     {({ isSubmitting, errors, touched }) => (
54       <Form>
55         {/* Username */}
56         <div>

```



```

57     <label htmlFor="username">Username:</label>
58     <Field
59         id="username"
60         name="username"
61         type="text"
62     />
63     <ErrorMessage name="username" component="span"
64         className="error" />
65 </div>
66 {/* Email */}
67 <div>
68     <label htmlFor="email">Email:</label>
69     <Field
70         id="email"
71         name="email"
72         type="email"
73     />
74     <ErrorMessage name="email" component="span"
75         className="error" />
76 </div>
77 {/* Password */}
78 <div>
79     <label htmlFor="password">Password:</label>
80     <Field
81         id="password"
82         name="password"
83         type="password"
84     />
85     <ErrorMessage name="password" component="span"
86         className="error" />
87 </div>
88 {/* Confirm Password */}
89 <div>
90     <label htmlFor="confirmPassword">Conferma
91         Password:</label>
92     <Field
93         id="confirmPassword"
94         name="confirmPassword"
95         type="password"
96     />
97     <ErrorMessage name="confirmPassword" component="span"
98         className="error" />
99 </div>
100 {/* Age */}
101 <div>
102     <label htmlFor="age">Età:</label>
103     <Field
104         id="age"
105         name="age"
106         type="number"
107     />
108     <ErrorMessage name="age" component="span"
109         className="error" />
110 </div>

```

```

109      {/* Terms Checkbox */}
110      <div>
111        <label>
112          <Field
113            type="checkbox"
114            name="terms"
115          />
116          Accetto i termini e condizioni
117        </label>
118        <ErrorMessage name="terms" component="span"
119          className="error" />
120      </div>
121
122      <button type="submit" disabled={isSubmitting}>
123        {isSubmitting ? 'Invio...' : 'Invia'}
124      </button>
125    </Form>
126  )}
127 </Formik>
128 );
129 }

```

### 7.5.3 Formik con Custom Components

Listing 7.11: Custom input components con Formik

```

1  import { Formik, Form, Field, ErrorMessage } from 'formik';
2
3  // Custom Input Component
4  function TextInput({ label, ...props }) {
5    return (
6      <div className="form-field">
7        <label htmlFor={props.id || props.name}>{label}</label>
8        <Field {...props} />
9        <ErrorMessage name={props.name} component="div" className="
10          error" />
11      </div>
12    );
13  }
14
15  // Custom Select Component
16  function Select({ label, options, ...props }) {
17    return (
18      <div className="form-field">
19        <label htmlFor={props.id || props.name}>{label}</label>
20        <Field as="select" {...props}>
21          <option value="">Seleziona...</option>
22          {options.map(option => (
23            <option key={option.value} value={option.value}>
24              {option.label}
25            </option>
26          ))}
27        </Field>
28        <ErrorMessage name={props.name} component="div" className="
29          error" />
30      </div>
31    );
32  }

```

```

29     );
30 }
31
32 // Custom Checkbox Component
33 function Checkbox({ children, ...props }) {
34     return (
35         <div className="form-field">
36             <label className="checkbox-label">
37                 <Field type="checkbox" {...props} />
38                 {children}
39             </label>
40             <ErrorMessage name={props.name} component="div" className="
41                 error" />
42         </div>
43     );
44 }
45
46 // Uso
47 function CustomComponentsForm() {
48     return (
49         <Formik
50             initialValues={{
51                 name: '',
52                 email: '',
53                 country: '',
54                 terms: false
55             }}
56             onSubmit={(values) => console.log(values)}
57         >
58             <Form>
59                 <TextInput
60                     label="Nome"
61                     name="name"
62                     type="text"
63                 />
64
65                 <TextInput
66                     label="Email"
67                     name="email"
68                     type="email"
69                 />
70
71                 <Select
72                     label="Paese"
73                     name="country"
74                     options={[
75                         { value: 'IT', label: 'Italia' },
76                         { value: 'US', label: 'USA' },
77                         { value: 'UK', label: 'UK' }
78                     ]}
79                 />
80
81                 <Checkbox name="terms">
82                     Accetto i termini e condizioni
83                 </Checkbox>
84
85                 <button type="submit">Invia</button>
86             </Form>

```

```

86     </Formik>
87   );
88 }

```

## 7.6 Form Patterns Avanzati

### 7.6.1 Multi-Step Form

Listing 7.12: Form multi-step

```

1  function MultiStepForm() {
2    const [step, setStep] = useState(1);
3    const [formData, setFormData] = useState({
4      // Step 1
5      personalInfo: {
6        firstName: '',
7        lastName: '',
8        email: ''
9      },
10     // Step 2
11     address: {
12       street: '',
13       city: '',
14       country: ''
15     },
16     // Step 3
17     account: {
18       username: '',
19       password: ''
20     }
21   });
22
23   const nextStep = () => setStep(prev => prev + 1);
24   const prevStep = () => setStep(prev => prev - 1);
25
26   const updateFormData = (section, data) => {
27     setFormData(prev => ({
28       ...prev,
29       [section]: { ...prev[section], ...data }
30     }));
31   };
32
33   const handleSubmit = () => {
34     console.log('Final form data:', formData);
35   };
36
37   return (
38     <div className="multi-step-form">
39       {/* Progress Indicator */}
40       <div className="progress">
41         <div className={`step ${step >= 1 ? 'active' : ''}`}>1</div>
42         <div className={`step ${step >= 2 ? 'active' : ''}`}>2</div>
43         <div className={`step ${step >= 3 ? 'active' : ''}`}>3</div>
44       </div>

```

```

45
46     {/* Step 1: Personal Info */}
47     {step === 1 && (
48         <Step1
49             data={formData.personalInfo}
50             onNext={({data}) => {
51                 updateFormData('personalInfo', data);
52                 nextStep();
53             }}
54         />
55     )}
56
57     {/* Step 2: Address */}
58     {step === 2 && (
59         <Step2
60             data={formData.address}
61             onNext={({data}) => {
62                 updateFormData('address', data);
63                 nextStep();
64             }}
65             onPrev={prevStep}
66         />
67     )}
68
69     {/* Step 3: Account */}
70     {step === 3 && (
71         <Step3
72             data={formData.account}
73             onSubmit={({data}) => {
74                 updateFormData('account', data);
75                 handleSubmit();
76             }}
77             onPrev={prevStep}
78         />
79     )}
80 </div>
81 );
82 }
83
84 // Step 1 Component
85 function Step1({ data, onNext }) {
86     const [formData, setFormData] = useState(data);
87
88     const handleSubmit = (e) => {
89         e.preventDefault();
90         onNext(formData);
91     };
92
93     return (
94         <form onSubmit={handleSubmit}>
95             <h2>Informazioni Personali</h2>
96             <input
97                 value={formData.firstName}
98                 onChange={({e}) => setFormData({ ...formData, firstName: e
99                     .target.value })}
100                 placeholder="Nome"
101                 required

```

```

102     <input
103         value={formData.lastName}
104         onChange={(e) => setFormData({ ...formData, lastName: e.
105             target.value })}
106         placeholder="Cognome"
107         required
108     />
109     <input
110         type="email"
111         value={formData.email}
112         onChange={(e) => setFormData({ ...formData, email: e.
113             target.value })}
114         placeholder="Email"
115         required
116     />
117     <button type="submit">Avanti</button>
118 </form>
119 );
120 }
121 // Step 2 e Step 3 simili...

```

## 7.6.2 Form con File Upload

Listing 7.13: File upload form

```

1 function FileUploadForm() {
2     const [file, setFile] = useState(null);
3     const [preview, setPreview] = useState(null);
4     const [uploading, setUploading] = useState(false);
5     const [progress, setProgress] = useState(0);
6
7     const handleFileChange = (e) => {
8         const selectedFile = e.target.files[0];
9
10        if (selectedFile) {
11            setFile(selectedFile);
12
13            // Preview per immagini
14            if (selectedFile.type.startsWith('image/')) {
15                const reader = new FileReader();
16                reader.onloadend = () => {
17                    setPreview(reader.result);
18                };
19                reader.readAsDataURL(selectedFile);
20            }
21        }
22    };
23
24    const handleSubmit = async (e) => {
25        e.preventDefault();
26
27        if (!file) {
28            alert('Seleziona un file');
29            return;
30        }
31    };

```

```

32     setUploading(true);
33
34     const formData = new FormData();
35     formData.append('file', file);
36     formData.append('description', 'File upload');
37
38     try {
39         const xhr = new XMLHttpRequest();
40
41         // Progress tracking
42         xhr.upload.addEventListener('progress', (e) => {
43             if (e.lengthComputable) {
44                 const percentComplete = (e.loaded / e.total) * 100;
45                 setProgress(percentComplete);
46             }
47         });
48
49         xhr.addEventListener('load', () => {
50             if (xhr.status === 200) {
51                 console.log('Upload completato!');
52                 setFile(null);
53                 setPreview(null);
54                 setProgress(0);
55             }
56         });
57
58         xhr.open('POST', '/api/upload');
59         xhr.send(formData);
60     } catch (error) {
61         console.error('Errore upload:', error);
62     } finally {
63         setUploading(false);
64     }
65 };
66
67 return (
68     <form onSubmit={handleSubmit}>
69         <div>
70             <input
71                 type="file"
72                 onChange={handleFileChange}
73                 accept="image/*"
74             />
75         </div>
76
77         {preview && (
78             <div>
79                 <h3>Anteprima:</h3>
80                 <img src={preview} alt="Preview" style={{ maxWidth:
81                     '300px' }} />
82             </div>
83         )}
84
85         {file && (
86             <div>
87                 <p>File: {file.name}</p>
88                 <p>Dimensione: {(file.size / 1024).toFixed(2)} KB</p>
89             </div>
90         )}
91     </form>
92 );

```

```

88         </div>
89     )}
90
91     {uploading && (
92         <div>
93             <progress value={progress} max="100" />
94             <p>{progress.toFixed(0)}%</p>
95         </div>
96     )}
97
98     <button type="submit" disabled={!file || uploading}>
99         {uploading ? 'Caricamento...' : 'Carica'}
100     </button>
101 </form>
102 );
103 }

```

### 7.6.3 Form con Async Validation

Listing 7.14: Validazione asincrona

```

1 function AsyncValidationForm() {
2     const [username, setUsername] = useState('');
3     const [checking, setChecking] = useState(false);
4     const [isAvailable, setIsAvailable] = useState(null);
5     const debounceRef = useRef(null);
6
7     // Check username availability
8     const checkUsername = async (value) => {
9         if (value.length < 3) {
10             setIsAvailable(null);
11             return;
12         }
13
14         setChecking(true);
15
16         try {
17             // Simula API call
18             await new Promise(resolve => setTimeout(resolve, 1000));
19             const response = await fetch(`/api/check-username?username=${
20                 value
21             }`);
22             const data = await response.json();
23
24             setIsAvailable(data.available);
25         } catch (error) {
26             console.error('Error checking username:', error);
27         } finally {
28             setChecking(false);
29         }
30     };
31
32     const handleUsernameChange = (e) => {
33         const value = e.target.value;
34         setUsername(value);
35         setIsAvailable(null);
36
37         // Debounce check

```



```

36      clearTimeout(debounceRef.current);
37      debounceRef.current = setTimeout(() => {
38          checkUsername(value);
39      }, 500);
40  };
41
42  const handleSubmit = (e) => {
43      e.preventDefault();
44
45      if (isAvailable) {
46          console.log('Submitting username:', username);
47      }
48  };
49
50  return (
51      <form onSubmit={handleSubmit}>
52          <div>
53              <label>Username:</label>
54              <input
55                  value={username}
56                  onChange={handleUsernameChange}
57                  placeholder="Scegli username"
58              />
59
60              {checking && <span>Controllo disponibilità...</span>}
61
62              {!checking && isAvailable === true && (
63                  <span style={{ color: 'green' }}>      Disponibile</
64                      span>
65                  )}
66
67              {!checking && isAvailable === false && (
68                  <span style={{ color: 'red' }}>      Non disponibile</
69                      span>
70                  )}
71              </div>
72
73              <button type="submit" disabled={!isAvailable}>
74                  Registra
75              </button>
76          </form>
77      );
78  }

```

## 7.7 Best Practices

### 7.7.1 1. Accessibilità

Listing 7.15: Form accessibili

```

1  function AccessibleForm() {
2      return (
3          <form>
4              {/*      Label associata a input */}
5              <label htmlFor="email">Email:</label>
6              <input id="email" type="email" name="email" />
7          </form>
8      );
9  }

```

```

8      { /*      Aria attributes */ }
9      <input
10         id="username"
11         aria-label="Username"
12         aria-required="true"
13         aria-invalid={errors.username ? 'true' : 'false'}
14         aria-describedby="username-error"
15     />
16     <span id="username-error" role="alert">
17         {errors.username}
18     </span>
19
20     { /*      Fieldset per raggruppamento */ }
21     <fieldset>
22         <legend>Informazioni di contatto</legend>
23         <input type="tel" name="phone" />
24         <input type="email" name="email" />
25     </fieldset>
26 </form>
27 );
28 }

```

### 7.7.2 2. Performance

Listing 7.16: Ottimizzazione performance

```

1  //      Debounce validation
2  const useDebounce = (value, delay) => {
3      const [debouncedValue, setDebouncedValue] = useState(value);
4
5      useEffect(() => {
6          const timer = setTimeout(() => setDebouncedValue(value), delay);
7          return () => clearTimeout(timer);
8      }, [value, delay]);
9
10     return debouncedValue;
11 };
12
13 //      Memoizza callbacks
14 const handleSubmit = useCallback(async (data) => {
15     await submitForm(data);
16 }, []);
17
18 //      Controlled components solo quando necessario
19 // Se non hai bisogno di validazione real-time, usa uncontrolled

```

### 7.7.3 3. Security

Listing 7.17: Sicurezza form

```

1  function SecureForm() {
2      const handleSubmit = async (data) => {
3          //      Sanitize input prima di inviare
4          const sanitized = {
5              username: DOMPurify.sanitize(data.username),
6              email: data.email.toLowerCase().trim()

```

```
7      };
8
9      //      HTTPS sempre
10     await fetch('https://api.example.com/submit', {
11       method: 'POST',
12       headers: {
13         'Content-Type': 'application/json',
14         //      CSRF token
15         'X-CSRF-Token': getCsrftoken()
16       },
17       body: JSON.stringify(sanitized)
18     });
19   };
20
21   return <form onSubmit={handleSubmit}>...</form>;
22 }
```

## 7.8 Conclusioni

In questo capitolo abbiamo esplorato:

- Controlled vs Uncontrolled components
- Form base con validazione manuale
- React Hook Form per form performanti
- Formik per form complessi
- Pattern avanzati: multi-step, file upload, async validation
- Best practices: accessibilità, performance, security

Ora hai tutti gli strumenti per gestire form complessi in React!



## Capitolo 8

# Ciclo di Vita dei Componenti

### 8.1 Introduzione al Lifecycle

Il ciclo di vita di un componente React rappresenta l'insieme delle fasi che un componente attraversa dalla sua creazione alla sua distruzione. Comprendere il lifecycle è fondamentale per gestire correttamente effetti collaterali, sottoscrizioni, timer e operazioni asincrone.

#### 8.1.1 Fasi del Lifecycle

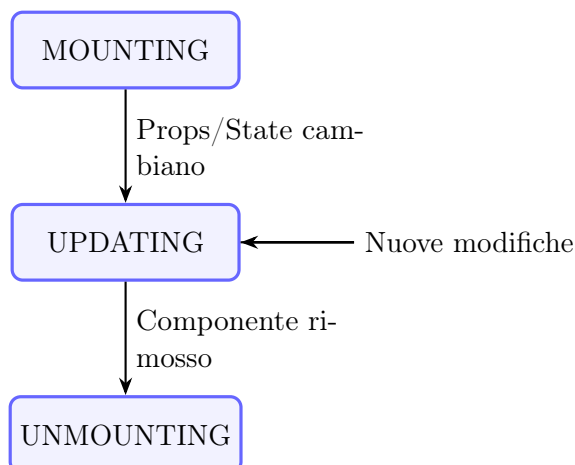
Ogni componente React passa attraverso tre fasi principali:

1. **Mounting** (Montaggio): Il componente viene creato e inserito nel DOM
2. **Updating** (Aggiornamento): Il componente viene ri-renderizzato a causa di cambiamenti in props o state
3. **Unmounting** (Smontaggio): Il componente viene rimosso dal DOM

#### Nota Importante

Con i Function Components e gli Hooks, non parliamo più di "lifecycle methods" ma di "effetti" gestiti tramite `useEffect`. Il paradigma è diverso: invece di pensare a "quando" fare qualcosa, pensiamo a "cosa" sincronizzare.

#### 8.1.2 Diagramma del Lifecycle



## 8.2 useEffect: Il Hook del Lifecycle

`useEffect` è il principale strumento per gestire il lifecycle nei Function Components. Permette di eseguire effetti collaterali in risposta a cambiamenti nel componente.

### 8.2.1 Sintassi Base

Listing 8.1: Sintassi base di `useEffect`

```
1 import { useEffect } from 'react';
2
3 function MyComponent() {
4   useEffect(() => {
5     // Codice da eseguire (effetto)
6
7     return () => {
8       // Codice di cleanup (opzionale)
9     };
10  }, [dependencies]); // Array di dipendenze
11 }
```

### 8.2.2 Componenti di `useEffect`

1. **Effect Function:** La funzione che contiene il codice dell'effetto
2. **Cleanup Function:** La funzione opzionale di pulizia (cleanup)
3. **Dependency Array:** L'array che determina quando l'effetto viene ri-eseguito

## 8.3 Dependency Array: Controllo dell'Esecuzione

L'array di dipendenze è cruciale per controllare quando un effetto viene eseguito.

### 8.3.1 Tre Pattern Principali

Listing 8.2: Pattern del dependency array

```
1 // 1. Nessun array: esegue ad ogni render
2 useEffect(() => {
3   console.log('Esegue ad ogni render');
4 });
5
6 // 2. Array vuoto: esegue solo al mount
7 useEffect(() => {
8   console.log('Esegue solo una volta al mount');
9 }, []);
10
11 // 3. Con dipendenze: esegue quando cambiano le dipendenze
12 useEffect(() => {
13   console.log('Esegue quando userId cambia');
14 }, [userId]);
```

### 8.3.2 Esempio Pratico: Document Title

Listing 8.3: Aggiornamento dinamico del titolo della pagina

```
1 import { useState, useEffect } from 'react';
2
3 function DocumentTitleUpdater() {
4   const [count, setCount] = useState(0);
5   const [name, setName] = useState('Utente');
6
7   // Effetto che dipende da count
8   useEffect(() => {
9     document.title = `Hai cliccato ${count} volte`;
10  }, [count]); // Si ri-esegue solo quando count cambia
11
12  // Effetto che dipende da name
13  useEffect(() => {
14    console.log(`L'utente ora si chiama: ${name}`);
15  }, [name]); // Si ri-esegue solo quando name cambia
16
17  return (
18    <div>
19      <h2>Ciao, {name}!</h2>
20      <input
21        type="text"
22        value={name}
23        onChange={(e) => setName(e.target.value)}
24        placeholder="Il tuo nome"
25      />
26
27      <p>Hai cliccato {count} volte</p>
28      <button onClick={() => setCount(count + 1)}>
29        Clicca qui
30      </button>
31    </div>
32  );
33 }
```

## 8.4 Mounting: Inizializzazione del Componente

La fase di mounting è quando il componente viene creato e inserito nel DOM per la prima volta.

### 8.4.1 Operazioni Tipiche al Mount

- Fetch di dati iniziali da API
- Sottoscrizione a eventi o servizi esterni
- Inizializzazione di librerie terze parti
- Setup di timer o intervalli
- Connessione a WebSocket

### 8.4.2 Esempio: Fetch Dati al Mount

Listing 8.4: Caricamento dati all'inizializzazione

```
1 import { useState, useEffect } from 'react';
2
3 function UserProfile({ userId }) {
4   const [user, setUser] = useState(null);
5   const [loading, setLoading] = useState(true);
6   const [error, setError] = useState(null);
7
8   useEffect(() => {
9     // Questa funzione viene eseguita solo al mount
10    // (o quando userId cambia)
11
12    setLoading(true);
13    setError(null);
14
15    fetch('https://api.example.com/users/${userId}')
16      .then(response => {
17        if (!response.ok) {
18          throw new Error('Errore nel caricamento');
19        }
20        return response.json();
21      })
22      .then(data => {
23        setUser(data);
24        setLoading(false);
25      })
26      .catch(err => {
27        setError(err.message);
28        setLoading(false);
29      });
30    }, [userId]); // Ri-esegue quando userId cambia
31
32    if (loading) return <p>Caricamento...</p>;
33    if (error) return <p>Errore: {error}</p>;
34    if (!user) return null;
35
36    return (
37      <div className="user-profile">
38        <h2>{user.name}</h2>
39        <p>Email: {user.email}</p>
40        <p>Username: {user.username}</p>
41      </div>
42    );
43 }
```

### 8.4.3 Pattern Avanzato: Async/Await nel useEffect

Listing 8.5: Uso di async/await in useEffect

```
1 import { useState, useEffect } from 'react';
2
3 function UserList() {
4   const [users, setUsers] = useState([]);
5   const [loading, setLoading] = useState(true);
6
7   useEffect(() => {
8     // Non possiamo rendere useEffect async direttamente
```



```
9      // Quindi creiamo una funzione async interna
10
11      const fetchUsers = async () => {
12        try {
13          setLoading(true);
14          const response = await fetch('https://api.example.com/users');
15
16          if (!response.ok) {
17            throw new Error('HTTP error! status: ${response.status}');
18          }
19
20          const data = await response.json();
21          setUsers(data);
22        } catch (error) {
23          console.error('Errore nel fetch:', error);
24        } finally {
25          setLoading(false);
26        }
27      };
28
29      fetchUsers();
30    }, []); // Array vuoto = esegue solo al mount
31
32    if (loading) {
33      return <div>Caricamento utenti...</div>;
34    }
35
36    return (
37      <ul>
38        {users.map(user => (
39          <li key={user.id}>{user.name}</li>
40        ))}
41      </ul>
42    );
43  }
```

#### Perché non possiamo usare async direttamente?

`useEffect` si aspetta che la funzione ritorni `undefined` o una funzione di `cleanup`. Una funzione `async` ritorna sempre una `Promise`, quindi non possiamo usarla direttamente. La soluzione è creare una funzione `async` interna e chiamarla.

## 8.5 Unmounting: Pulizia del Componente

La fase di `unmounting` è quando il componente viene rimosso dal DOM. È fondamentale pulire le risorse per evitare `memory leak`.

### 8.5.1 Cleanup Function

La funzione di `cleanup` viene eseguita:

- Prima che l'effetto venga ri-eseguito (se le dipendenze cambiano)
- Quando il componente viene smontato dal DOM

### 8.5.2 Esempio: Timer con Cleanup

Listing 8.6: Gestione di un timer con cleanup

```
1 import { useState, useEffect } from 'react';
2
3 function Timer() {
4   const [seconds, setSeconds] = useState(0);
5   const [isRunning, setIsRunning] = useState(false);
6
7   useEffect(() => {
8     let intervalId = null;
9
10    if (isRunning) {
11      // Setup: avvia il timer
12      intervalId = setInterval(() => {
13        setSeconds(prev => prev + 1);
14      }, 1000);
15
16      console.log('Timer avviato');
17    }
18
19    // Cleanup: pulisce il timer
20    return () => {
21      if (intervalId) {
22        clearInterval(intervalId);
23        console.log('Timer pulito');
24      }
25    };
26  }, [isRunning]); // Ri-esegue quando isRunning cambia
27
28  const toggle = () => setIsRunning(!isRunning);
29  const reset = () => {
30    setIsRunning(false);
31    setSeconds(0);
32  };
33
34  return (
35    <div className="timer">
36      <h2>Timer: {seconds}s</h2>
37      <button onClick={toggle}>
38        {isRunning ? 'Pausa' : 'Avvia'}
39      </button>
40      <button onClick={reset}>Reset</button>
41    </div>
42  );
43 }
```

### 8.5.3 Esempio: Event Listener con Cleanup

Listing 8.7: Gestione di event listener del browser

```
1 import { useState, useEffect } from 'react';
2
3 function WindowSize() {
4   const [windowSize, setWindowSize] = useState({
5     width: window.innerWidth,
6     height: window.innerHeight
```

```

7   });
8
9   useEffect(() => {
10    // Handler dell'evento
11    const handleResize = () => {
12      setWindowSize({
13        width: window.innerWidth,
14        height: window.innerHeight
15      });
16    };
17
18    // Setup: aggiungi event listener
19    window.addEventListener('resize', handleResize);
20    console.log('Event listener aggiunto');
21
22    // Cleanup: rimuovi event listener
23    return () => {
24      window.removeEventListener('resize', handleResize);
25      console.log('Event listener rimosso');
26    };
27  }, []); // Array vuoto = setup al mount, cleanup all'unmount
28
29  return (
30    <div>
31      <h2>Dimensioni Finestra</h2>
32      <p>Larghezza: {windowSize.width}px</p>
33      <p>Altezza: {windowSize.height}px</p>
34    </div>
35  );
36 }

```

#### 8.5.4 Esempio: WebSocket con Cleanup

Listing 8.8: Connessione WebSocket con pulizia corretta

```

1  import { useState, useEffect } from 'react';
2
3  function ChatRoom({ roomId }) {
4    const [messages, setMessages] = useState([]);
5    const [connectionStatus, setConnectionStatus] = useState('disconnected');
6
7    useEffect(() => {
8      // Setup: crea connessione WebSocket
9      const ws = new WebSocket('wss://chat.example.com/room/${roomId}');
10
11      ws.onopen = () => {
12        console.log('Connesso alla chat room:', roomId);
13        setConnectionStatus('connected');
14      };
15
16      ws.onmessage = (event) => {
17        const message = JSON.parse(event.data);
18        setMessages(prev => [...prev, message]);
19      };
20
21      ws.onerror = (error) => {

```

```

22     console.error('WebSocket error:', error);
23     setConnectionStatus('error');
24 };
25
26 ws.onclose = () => {
27     console.log('Disconnesso dalla chat room');
28     setConnectionStatus('disconnected');
29 };
30
31 // Cleanup: chiudi connessione quando il componente si smonta
32 // o quando roomId cambia
33 return () => {
34     console.log('Chiusura connessione WebSocket...');
35     ws.close();
36 };
37 }, [roomId]); // Ri-connetti quando cambia la room
38
39 return (
40     <div className="chat-room">
41         <div className="status">
42             Status: <span className={connectionStatus}>{connectionStatus}</span>
43         </div>
44         <div className="messages">
45             {messages.map((msg, index) => (
46                 <div key={index} className="message">
47                     <strong>{msg.user}</strong> {msg.text}
48                 </div>
49             ))}
50         </div>
51     </div>
52 );
53 }

```

## 8.6 Updating: Aggiornamenti del Componente

La fase di updating si verifica ogni volta che props o state cambiano, causando un re-render del componente.

### 8.6.1 Controllo degli Aggiornamenti

Listing 8.9: Esempio di effetto che si ri-esegue agli aggiornamenti

```

1  import { useState, useEffect } from 'react';
2
3  function SearchResults({ searchTerm }) {
4      const [results, setResults] = useState([]);
5      const [loading, setLoading] = useState(false);
6
7      useEffect(() => {
8          // Questo effetto si ri-esegue ogni volta che searchTerm cambia
9          if (!searchTerm) {
10             setResults([]);
11             return;
12         }
13     });

```

```

14     setLoading(true);
15
16     // Simula chiamata API
17     const timeoutId = setTimeout(() => {
18         const mockResults = [
19             'Risultato 1 per "${searchTerm}"',
20             'Risultato 2 per "${searchTerm}"',
21             'Risultato 3 per "${searchTerm}"'
22         ];
23         setResults(mockResults);
24         setLoading(false);
25     }, 500);
26
27     // Cleanup: annulla la ricerca precedente se searchTerm cambia
28     return () => {
29         clearTimeout(timeoutId);
30     };
31 }, [searchTerm]);
32
33 if (loading) return <div>Ricerca in corso...</div>;
34
35 return (
36     <ul>
37         {results.map((result, index) => (
38             <li key={index}>{result}</li>
39         ))}
40     </ul>
41 );
42 }
43
44 // Componente padre con input
45 function SearchApp() {
46     const [searchTerm, setSearchTerm] = useState('');
47
48     return (
49         <div>
50             <input
51                 type="text"
52                 value={searchTerm}
53                 onChange={(e) => setSearchTerm(e.target.value)}
54                 placeholder="Cerca..."
55             />
56             <SearchResults searchTerm={searchTerm} />
57         </div>
58     );
59 }

```

### 8.6.2 Debouncing con useEffect

Il debouncing è una tecnica per ritardare l'esecuzione di un'operazione fino a quando l'utente smette di effettuare azioni.

Listing 8.10: Implementazione di debounce con useEffect

```

1 import { useState, useEffect } from 'react';
2
3 function DebouncedSearch() {
4     const [searchTerm, setSearchTerm] = useState('');

```

```

5   const [debouncedTerm, setDebouncedTerm] = useState('');
6
7   // Effetto per il debouncing
8   useEffect(() => {
9     // Imposta un timer per aggiornare debouncedTerm dopo 500ms
10    const timerId = setTimeout(() => {
11      setDebouncedTerm(searchTerm);
12    }, 500);
13
14    // Cleanup: cancella il timer se searchTerm cambia
15    // prima che i 500ms siano trascorsi
16    return () => {
17      clearTimeout(timerId);
18    };
19  }, [searchTerm]);
20
21  // Effetto per la ricerca (si attiva solo quando debouncedTerm cambia)
22  useEffect(() => {
23    if (debouncedTerm) {
24      console.log('Eseguo ricerca per:', debouncedTerm);
25      // Qui faremmo la chiamata API
26    }
27  }, [debouncedTerm]);
28
29  return (
30    <div>
31      <input
32        type="text"
33        value={searchTerm}
34        onChange={(e) => setSearchTerm(e.target.value)}
35        placeholder="Cerca (con debounce)..."
36      />
37      <p>Termine di ricerca: {searchTerm}</p>
38      <p>Ricerca attiva per: {debouncedTerm}</p>
39    </div>
40  );
41 }

```

## 8.7 Pattern Avanzati del Lifecycle

### 8.7.1 Cancellazione di Fetch con AbortController

Per evitare race condition e aggiornamenti su componenti smontati, è importante cancellare le richieste HTTP.

Listing 8.11: Fetch con cancellazione tramite AbortController

```

1   import { useState, useEffect } from 'react';
2
3   function UserData({ userId }) {
4     const [user, setUser] = useState(null);
5     const [loading, setLoading] = useState(true);
6     const [error, setError] = useState(null);
7
8     useEffect(() => {
9       // Crea un AbortController per questa richiesta
10      const controller = new AbortController();
11      const signal = controller.signal;

```

```
12
13   const fetchUser = async () => {
14     try {
15       setLoading(true);
16       setError(null);
17
18       const response = await fetch(
19         'https://api.example.com/users/${userId}',
20         { signal } // Passa il signal alla fetch
21       );
22
23       if (!response.ok) {
24         throw new Error('Errore nel caricamento');
25       }
26
27       const data = await response.json();
28
29       // Controlla se il componente è ancora montato
30       if (!signal.aborted) {
31         setUser(data);
32       }
33     } catch (err) {
34       // Ignora errori di abort
35       if (err.name !== 'AbortError') {
36         setError(err.message);
37       }
38     } finally {
39       if (!signal.aborted) {
40         setLoading(false);
41       }
42     }
43   };
44
45   fetchUser();
46
47   // Cleanup: annulla la richiesta se il componente si smonta
48   // o se userId cambia
49   return () => {
50     controller.abort();
51   };
52 }, [userId]);
53
54 if (loading) return <div>Caricamento...</div>;
55 if (error) return <div>Errore: {error}</div>;
56 if (!user) return null;
57
58 return (
59   <div>
60     <h2>{user.name}</h2>
61     <p>{user.email}</p>
62   </div>
63 );
64 }
```

### 8.7.2 Custom Hook per il Lifecycle

Possiamo creare custom hooks per riutilizzare logica di lifecycle.

Listing 8.12: Custom hook useMount e useUnmount

```
1 import { useEffect } from 'react';
2
3 // Hook che esegue una funzione solo al mount
4 function useMount(callback) {
5   useEffect(() => {
6     callback();
7   }, []); // Array vuoto = esegue solo al mount
8 }
9
10 // Hook che esegue una funzione solo all'unmount
11 function useUnmount(callback) {
12   useEffect(() => {
13     return callback; // Ritorna la funzione di cleanup
14   }, []); // Array vuoto = cleanup solo all'unmount
15 }
16
17 // Hook che esegue codice al mount e all'unmount
18 function useMountUnmount(onMount, onUnmount) {
19   useEffect(() => {
20     onMount();
21     return onUnmount;
22   }, []);
23 }
24
25 // Esempio di utilizzo
26 function ExampleComponent() {
27   useMount(() => {
28     console.log('Componente montato!');
29   });
30
31   useUnmount(() => {
32     console.log('Componente smontato!');
33   });
34
35   return <div>Esempio</div>;
36 }
```

### 8.7.3 Hook useUpdateEffect

Un hook che esegue l'effetto solo agli aggiornamenti, non al mount iniziale.

Listing 8.13: Custom hook useUpdateEffect

```
1 import { useEffect, useRef, useState } from 'react';
2
3 function useUpdateEffect(effect, dependencies) {
4   const isFirstRender = useRef(true);
5
6   useEffect(() => {
7     if (isFirstRender.current) {
8       isFirstRender.current = false;
9       return;
10    }
11
12    return effect();
13  }, dependencies);
14 }
```



```
15
16 // Esempio di utilizzo
17 function Counter() {
18   const [count, setCount] = useState(0);
19
20   // Questo NON si esegue al primo render
21   useEffect(() => {
22     console.log('Count è stato aggiornato a:', count);
23   }, [count]);
24
25   return (
26     <div>
27       <p>Count: {count}</p>
28       <button onClick={() => setCount(count + 1)}>Incrementa</button>
29     </div>
30   );
31 }
```

#### 8.7.4 Hook useInterval

Un hook personalizzato per gestire intervalli in modo sicuro.

Listing 8.14: Custom hook useInterval

```
1 import { useEffect, useRef, useState } from 'react';
2
3 function useInterval(callback, delay) {
4   const savedCallback = useRef();
5
6   // Salva la callback più recente
7   useEffect(() => {
8     savedCallback.current = callback;
9   }, [callback]);
10
11   // Setup dell'intervallo
12   useEffect(() => {
13     if (delay === null) return;
14
15     const tick = () => {
16       savedCallback.current();
17     };
18
19     const id = setInterval(tick, delay);
20
21     // Cleanup
22     return () => clearInterval(id);
23   }, [delay]);
24 }
25
26 // Esempio di utilizzo
27 function Clock() {
28   const [time, setTime] = useState(new Date());
29
30   useInterval(() => {
31     setTime(new Date());
32   }, 1000); // Aggiorna ogni secondo
33
34   return (
```

```
35     <div>
36       <h2>{time.toLocaleTimeString()}</h2>
37     </div>
38   );
39 }
```

## 8.8 Errori Comuni e Best Practices

### 8.8.1 Errore: Dipendenze Mancanti

Listing 8.15: Esempio di dipendenze mancanti

```
1 // SBAGLIATO: count non è nelle dipendenze
2 function BadExample() {
3   const [count, setCount] = useState(0);
4
5   useEffect(() => {
6     const interval = setInterval(() => {
7       console.log(count); // Usa count ma non è nelle dipendenze!
8     }, 1000);
9
10    return () => clearInterval(interval);
11  }, []); // Manca count
12
13  return <button onClick={() => setCount(count + 1)}>Click</button>;
14 }
15
16 // CORRETTO: Usa la forma funzionale di setState
17 function GoodExample() {
18   const [count, setCount] = useState(0);
19
20   useEffect(() => {
21     const interval = setInterval(() => {
22       setCount(prev => {
23         console.log(prev); // Accedi al valore tramite callback
24         return prev;
25       });
26     }, 1000);
27
28     return () => clearInterval(interval);
29   }, []); // Corretto: non dipende da count
30
31   return <button onClick={() => setCount(count + 1)}>Click</button>;
32 }
```

### 8.8.2 Errore: Memory Leak

Listing 8.16: Come evitare memory leak

```
1 // SBAGLIATO: Nessun cleanup
2 function BadComponent() {
3   const [data, setData] = useState(null);
4
5   useEffect(() => {
6     fetch('https://api.example.com/data')
7       .then(res => res.json())
```

```
8      .then(data => setData(data)); // Potrebbe eseguire su componente
      smontato
9  }, []);
10
11  return <div>{data?.value}</div>;
12 }
13
14 // CORRETTO: Con flag di cleanup
15 function GoodComponent() {
16   const [data, setData] = useState(null);
17
18   useEffect(() => {
19     let isMounted = true;
20
21     fetch('https://api.example.com/data')
22       .then(res => res.json())
23       .then(data => {
24         if (isMounted) { // Controlla se il componente è ancora montato
25           setData(data);
26         }
27       });
28
29     return () => {
30       isMounted = false; // Cleanup
31     };
32   }, []);
33
34   return <div>{data?.value}</div>;
35 }
```

### 8.8.3 Best Practice: Separazione degli Effetti

Listing 8.17: Separare effetti con dipendenze diverse

```
1 // SBAGLIATO: Un unico effetto con troppe responsabilità
2 function BadComponent({ userId, theme }) {
3   useEffect(() => {
4     // Fetch user data
5     fetchUser(userId);
6
7     // Update theme
8     document.body.className = theme;
9
10    // Setup analytics
11    analytics.track('page_view');
12  }, [userId, theme]); // Entrambe le operazioni si ri-eseguono
13 }
14
15 // CORRETTO: Effetti separati
16 function GoodComponent({ userId, theme }) {
17   // Effetto per il fetch dell'utente
18   useEffect(() => {
19     fetchUser(userId);
20   }, [userId]);
21
22   // Effetto per il tema
23   useEffect(() => {
```

```

24     document.body.className = theme;
25 }, [theme]);
26
27 // Effetto per analytics (solo al mount)
28 useEffect(() => {
29     analytics.track('page_view');
30 }, []);
31 }

```

## 8.9 Casi d'Uso Pratici Completi

### 8.9.1 Sistema di Notifiche con Timeout

Listing 8.18: Sistema completo di notifiche

```

1  import { useState, useEffect } from 'react';
2
3  function Notification({ message, duration = 3000, onClose }) {
4      useEffect(() => {
5          // Imposta timer per chiudere automaticamente
6          const timer = setTimeout(() => {
7              onClose();
8          }, duration);
9
10         // Cleanup: cancella timer se il componente si smonta prima
11         return () => {
12             clearTimeout(timer);
13         };
14     }, [duration, onClose]);
15
16     return (
17         <div className="notification">
18             <p>{message}</p>
19             <button onClick={onClose}>X</button>
20         </div>
21     );
22 }
23
24 function NotificationSystem() {
25     const [notifications, setNotifications] = useState([]);
26
27     const addNotification = (message) => {
28         const id = Date.now();
29         setNotifications(prev => [...prev, { id, message }]);
30     };
31
32     const removeNotification = (id) => {
33         setNotifications(prev => prev.filter(n => n.id !== id));
34     };
35
36     return (
37         <div>
38             <button onClick={() => addNotification('Nuova notifica!')}>
39                 Mostra Notifica
40             </button>
41
42             <div className="notifications-container">

```

```

43     {notifications.map(notification => (
44         <Notification
45             key={notification.id}
46             message={notification.message}
47             onClose={() => removeNotification(notification.id)}
48         />
49     ))}
50 </div>
51 </div>
52 );
53 }

```

### 8.9.2 Auto-Save con Debounce

Listing 8.19: Auto-save di un form con debounce

```

1  import { useState, useEffect, useRef } from 'react';
2
3  function AutoSaveForm() {
4      const [formData, setFormData] = useState({
5          title: '',
6          content: ''
7      });
8      const [saveStatus, setSaveStatus] = useState('saved'); // saved,
        saving, error
9      const [lastSaved, setLastSaved] = useState(null);
10
11     // Debounce e auto-save
12     useEffect(() => {
13         // Non salvare se i campi sono vuoti
14         if (!formData.title && !formData.content) {
15             return;
16         }
17
18         setSaveStatus('saving');
19
20         // Debounce di 2 secondi
21         const saveTimer = setTimeout(async () => {
22             try {
23                 // Simula chiamata API
24                 await new Promise(resolve => setTimeout(resolve, 500));
25
26                 console.log('Dati salvati:', formData);
27
28                 setSaveStatus('saved');
29                 setLastSaved(new Date());
30             } catch (error) {
31                 setSaveStatus('error');
32                 console.error('Errore nel salvataggio:', error);
33             }
34         }, 2000);
35
36         // Cleanup: cancella il timer se formData cambia prima di 2 secondi
37         return () => {
38             clearTimeout(saveTimer);
39         };
40     }, [formData]);

```

```
41
42  const handleChange = (field, value) => {
43    setFormData(prev => ({ ...prev, [field]: value }));
44  };
45
46  return (
47    <div className="auto-save-form">
48      <div className="status-bar">
49        {saveStatus === 'saving' && <span>Salvataggio in corso...</span>
50        }
51        {saveStatus === 'saved' && (
52          <span>
53            Salvato {lastSaved && 'alle ${lastSaved.toLocaleTimeString()}
54            '}
55          </span>
56        )}
57        {saveStatus === 'error' && <span className="error">Errore
58          salvataggio</span>}
59      </div>
60
61      <input
62        type="text"
63        value={formData.title}
64        onChange={(e) => handleChange('title', e.target.value)}
65        placeholder="Titolo"
66      />
67
68      <textarea
69        value={formData.content}
70        onChange={(e) => handleChange('content', e.target.value)}
71        placeholder="Contenuto"
72        rows={10}
73      />
74    </div>
75  );
76 }
```

## 8.10 Riepilogo Best Practices

### Checklist Lifecycle Best Practices

- Sempre includere tutte le dipendenze necessarie nell'array di dipendenze
- Separare effetti con dipendenze diverse in `useEffect` distinti
- Implementare sempre `cleanup` per timer, event listener, subscription
- Usare `AbortController` per cancellare fetch in corso
- Controllare se il componente è ancora montato prima di `setState` dopo operazioni `async`
- Non usare `async` direttamente in `useEffect`, creare funzione `async` interna
- Usare `useRef` per valori che non devono causare re-render
- Evitare di avere troppe responsabilità in un singolo effetto

- Preferire custom hooks per logica di lifecycle riutilizzabile
- Testare sempre il comportamento all'unmount

## 8.11 Conclusioni

La gestione del lifecycle è uno degli aspetti più importanti di React. Con `useEffect` e i pattern corretti, possiamo:

- Gestire effetti collaterali in modo dichiarativo
- Sincronizzare componenti con sistemi esterni
- Evitare memory leak e race condition
- Creare applicazioni performanti e affidabili

La chiave è pensare in termini di "sincronizzazione" piuttosto che di "eventi del lifecycle". Ogni effetto dovrebbe rispondere alla domanda: "Con cosa devo sincronizzare questo componente?"





## Capitolo 9

# Routing con React Router

### 9.1 Introduzione al Routing in React

Il routing è essenziale per creare applicazioni React multi-pagina (SPA - Single Page Application). React Router è la libreria standard per la gestione del routing in React, permettendo di navigare tra diverse "pagine" senza ricaricare il browser.

#### 9.1.1 Cos'è React Router

React Router è una libreria che sincronizza l'UI con l'URL del browser, permettendo di:

- Creare applicazioni multi-pagina
- Gestire la navigazione dichiarativa
- Implementare route nidificate
- Proteggere route con autenticazione
- Implementare lazy loading per code splitting
- Gestire parametri URL e query string

#### 9.1.2 Installazione

Listing 9.1: Installazione di React Router

```
1 # Installazione con npm
2 npm install react-router-dom
3
4 # Installazione con yarn
5 yarn add react-router-dom
6
7 # Installazione con pnpm
8 pnpm add react-router-dom
```

#### React Router v6

Questo capitolo si concentra su React Router v6, che introduce cambiamenti significativi rispetto alla v5. Le API sono più semplici e l'approccio è più dichiarativo.

## 9.2 Setup Iniziale

### 9.2.1 Struttura Base dell'Applicazione

Listing 9.2: Setup iniziale con BrowserRouter

```
1 // main.jsx o index.jsx
2 import React from 'react';
3 import ReactDOM from 'react-dom/client';
4 import { BrowserRouter } from 'react-router-dom';
5 import App from './App';
6 import './index.css';
7
8 ReactDOM.createRoot(document.getElementById('root')).render(
9   <React.StrictMode>
10     <BrowserRouter>
11       <App />
12     </BrowserRouter>
13   </React.StrictMode>
14 );
```

### 9.2.2 Componente App con Routes

Listing 9.3: App.jsx con routes di base

```
1 import { Routes, Route, Link } from 'react-router-dom';
2 import Home from './pages/Home';
3 import About from './pages/About';
4 import Contact from './pages/Contact';
5 import NotFound from './pages/NotFound';
6
7 function App() {
8   return (
9     <div className="app">
10       {/* Navigation */}
11       <nav>
12         <ul>
13           <li><Link to="/">Home</Link></li>
14           <li><Link to="/about">About</Link></li>
15           <li><Link to="/contact">Contact</Link></li>
16         </ul>
17       </nav>
18
19       {/* Routes */}
20       <Routes>
21         <Route path="/" element={<Home />} />
22         <Route path="/about" element={<About />} />
23         <Route path="/contact" element={<Contact />} />
24         <Route path="*" element={<NotFound />} />
25       </Routes>
26     </div>
27   );
28 }
29
30 export default App;
```

### 9.2.3 Componenti Pagina di Base

Listing 9.4: Esempi di componenti pagina

```
1 // pages/Home.jsx
2 function Home() {
3   return (
4     <div>
5       <h1>Home Page</h1>
6       <p>Benvenuto nella home page!</p>
7     </div>
8   );
9 }
10
11 export default Home;
12
13 // pages/About.jsx
14 function About() {
15   return (
16     <div>
17       <h1>About Page</h1>
18       <p>Informazioni sull'applicazione</p>
19     </div>
20   );
21 }
22
23 export default About;
24
25 // pages/NotFound.jsx
26 import { Link } from 'react-router-dom';
27
28 function NotFound() {
29   return (
30     <div>
31       <h1>404 - Pagina Non Trovata</h1>
32       <p>La pagina che stai cercando non esiste.</p>
33       <Link to="/">Torna alla Home</Link>
34     </div>
35   );
36 }
37
38 export default NotFound;
```

## 9.3 Navigazione

### 9.3.1 Link Component

Il componente `Link` è usato per creare link che non ricaricano la pagina.

Listing 9.5: Utilizzo del componente `Link`

```
1 import { Link } from 'react-router-dom';
2
3 function Navigation() {
4   return (
5     <nav>
6       {/* Link semplice */}
7       <Link to="/">Home</Link>
```

```

8
9      {/* Link con classi CSS */}
10     <Link to="/about" className="nav-link">About</Link>
11
12     {/* Link con inline style */}
13     <Link to="/contact" style={{ color: 'blue' }}>Contact</Link>
14
15     {/* Link con state (passa dati alla route) */}
16     <Link to="/profile" state={{ from: 'navigation' }}>
17       Profile
18     </Link>
19   </nav>
20 );
21 }

```

### 9.3.2 NavLink Component

NavLink è una versione speciale di Link che sa quando è attivo.

Listing 9.6: NavLink per navigazione attiva

```

1  import { NavLink } from 'react-router-dom';
2
3  function Navigation() {
4    return (
5      <nav>
6        {/* NavLink con classe attiva automatica */}
7        <NavLink
8          to="/"
9          className={({ isActive }) => isActive ? 'active' : ''}
10       >
11          Home
12        </NavLink>
13
14        {/* NavLink con stile attivo */}
15        <NavLink
16          to="/about"
17          style={({ isActive }) => ({
18            color: isActive ? 'red' : 'blue',
19            fontWeight: isActive ? 'bold' : 'normal'
20          })}
21       >
22          About
23        </NavLink>
24
25        {/* NavLink con end (esatto match del path) */}
26        <NavLink
27          to="/"
28          end
29          className={({ isActive }) => isActive ? 'active' : ''}
30       >
31          Home (exact)
32        </NavLink>
33      </nav>
34    );
35  }
36
37  // CSS

```

```
38 // .active {
39 //   color: red;
40 //   font-weight: bold;
41 //   border-bottom: 2px solid red;
42 // }
```

### 9.3.3 Navigazione Programmatica

Usare `useNavigate` per navigare da codice (es. dopo submit form).

Listing 9.7: `useNavigate` per navigazione programmatica

```
1 import { useNavigate } from 'react-router-dom';
2
3 function LoginForm() {
4   const navigate = useNavigate();
5
6   const handleSubmit = async (e) => {
7     e.preventDefault();
8
9     // Simula login
10    const success = await login(username, password);
11
12    if (success) {
13      // Naviga alla dashboard dopo login
14      navigate('/dashboard');
15
16      // Naviga con replace (non aggiunge alla history)
17      // navigate('/dashboard', { replace: true });
18
19      // Naviga passando state
20      // navigate('/dashboard', { state: { user: userData } });
21    }
22  };
23
24  // Torna indietro
25  const goBack = () => {
26    navigate(-1); // Come browser back button
27  };
28
29  // Vai avanti
30  const goForward = () => {
31    navigate(1);
32  };
33
34  return (
35    <form onSubmit={handleSubmit}>
36      {/* form fields */}
37      <button type="submit">Login</button>
38      <button type="button" onClick={goBack}>Indietro</button>
39    </form>
40  );
41 }
```

## 9.4 Parametri URL

### 9.4.1 Route Parameters

I parametri URL permettono di creare route dinamiche.

Listing 9.8: Route con parametri

```
1 import { Routes, Route } from 'react-router-dom';
2 import UserProfile from '../pages/UserProfile';
3 import BlogPost from '../pages/BlogPost';
4
5 function App() {
6   return (
7     <Routes>
8       {/* Parametro singolo */}
9       <Route path="/users/:userId" element={<UserProfile />} />
10
11       {/* Parametri multipli */}
12       <Route path="/blog/:category/:postId" element={<BlogPost />} />
13
14       {/* Parametro opzionale (con ?) */}
15       <Route path="/products/:id?" element={<Products />} />
16     </Routes>
17   );
18 }
```

### 9.4.2 useParams Hook

Accedere ai parametri URL con useParams.

Listing 9.9: Accesso ai parametri con useParams

```
1 import { useParams } from 'react-router-dom';
2 import { useState, useEffect } from 'react';
3
4 function UserProfile() {
5   const { userId } = useParams(); // Estrai parametro dall'URL
6   const [user, setUser] = useState(null);
7   const [loading, setLoading] = useState(true);
8
9   useEffect(() => {
10     const fetchUser = async () => {
11       setLoading(true);
12       try {
13         const response = await fetch(`/api/users/${userId}`);
14         const data = await response.json();
15         setUser(data);
16       } catch (error) {
17         console.error('Errore:', error);
18       } finally {
19         setLoading(false);
20       }
21     };
22
23     fetchUser();
24   }, [userId]); // Ri-esegui quando userId cambia
25
26   if (loading) return <div>Caricamento...</div>;
```

```

27   if (!user) return <div>Utente non trovato</div>;
28
29   return (
30     <div>
31       <h1>{user.name}</h1>
32       <p>Email: {user.email}</p>
33       <p>User ID: {userId}</p>
34     </div>
35   );
36 }

```

### 9.4.3 Esempio: Blog con Parametri Multipli

Listing 9.10: Blog post con categoria e ID

```

1  import { useParams, Link } from 'react-router-dom';
2
3  function BlogPost() {
4    const { category, postId } = useParams();
5    const [post, setPost] = useState(null);
6
7    useEffect(() => {
8      // Fetch post based on category and postId
9      fetch(`/api/blog/${category}/${postId}`)
10        .then(res => res.json())
11        .then(setPost);
12    }, [category, postId]);
13
14    if (!post) return <div>Loading...</div>;
15
16    return (
17      <article>
18        <div className="breadcrumb">
19          <Link to="/blog">Blog</Link> /
20          <Link to={`/${category}`}>{category}</Link> /
21          {post.title}
22        </div>
23
24        <h1>{post.title}</h1>
25        <p className="category">Categoria: {category}</p>
26        <div className="content">{post.content}</div>
27      </article>
28    );
29  }

```

## 9.5 Query Parameters

### 9.5.1 useSearchParams Hook

Gestire query string (?page=1&sort=asc) con `useSearchParams`.

Listing 9.11: Gestione query parameters

```

1  import { useSearchParams } from 'react-router-dom';
2
3  function ProductList() {
4    const [searchParams, setSearchParams] = useSearchParams();

```

```

5
6 // Leggi query params
7 const page = searchParams.get('page') || '1';
8 const sort = searchParams.get('sort') || 'name';
9 const category = searchParams.get('category');
10
11 // Aggiorna query params
12 const updatePage = (newPage) => {
13   setSearchParams({ page: newPage, sort, category });
14 };
15
16 const updateSort = (newSort) => {
17   setSearchParams({ page, sort: newSort, category });
18 };
19
20 // Rimuovi un query param
21 const clearCategory = () => {
22   const params = new URLSearchParams(searchParams);
23   params.delete('category');
24   setSearchParams(params);
25 };
26
27 return (
28   <div>
29     <h1>Prodotti</h1>
30
31     {/* Filtri */}
32     <div>
33       <button onClick={() => updateSort('name')}>Ordina per Nome</
34         button>
35       <button onClick={() => updateSort('price')}>Ordina per Prezzo</
36         button>
37     </div>
38
39     {/* Paginazione */}
40     <div>
41       <button onClick={() => updatePage(parseInt(page) - 1)}>
42         Precedente
43       </button>
44       <span>Pagina {page}</span>
45       <button onClick={() => updatePage(parseInt(page) + 1)}>
46         Successivo
47       </button>
48     </div>
49
50     {/* Mostra filtri attivi */}
51     <div>
52       Ordinamento: {sort}
53       {category && (
54         <span>
55           , Categoria: {category}
56           <button onClick={clearCategory}>X</button>
57         </span>
58       )}
59     </div>
60   </div>
61 );
62 }

```



### 9.5.2 Esempio Completo: Ricerca e Filtri

Listing 9.12: Sistema di ricerca con filtri URL

```

1 import { useSearchParams, useNavigate } from 'react-router-dom';
2 import { useState, useEffect } from 'react';
3
4 function ProductSearch() {
5   const [searchParams, setSearchParams] = useSearchParams();
6   const [products, setProducts] = useState([]);
7   const [loading, setLoading] = useState(false);
8
9   // Estrai parametri dalla URL
10  const query = searchParams.get('q') || '';
11  const category = searchParams.get('category') || 'all';
12  const minPrice = searchParams.get('minPrice') || '';
13  const maxPrice = searchParams.get('maxPrice') || '';
14
15  // Form state locale
16  const [searchInput, setSearchInput] = useState(query);
17
18  // Fetch prodotti quando cambiano i parametri
19  useEffect(() => {
20    const fetchProducts = async () => {
21      setLoading(true);
22      const params = new URLSearchParams(searchParams);
23      const response = await fetch(`/api/products?${params}`);
24      const data = await response.json();
25      setProducts(data);
26      setLoading(false);
27    };
28
29    fetchProducts();
30  }, [searchParams]);
31
32  // Handler per il form di ricerca
33  const handleSearch = (e) => {
34    e.preventDefault();
35    setSearchParams(prev => {
36      const params = new URLSearchParams(prev);
37      if (searchInput) {
38        params.set('q', searchInput);
39      } else {
40        params.delete('q');
41      }
42      return params;
43    });
44  };
45
46  // Handler per cambio categoria
47  const handleCategoryChange = (newCategory) => {
48    setSearchParams(prev => {
49      const params = new URLSearchParams(prev);
50      params.set('category', newCategory);
51      return params;
52    });
53  };
54
55  // Handler per filtro prezzo

```

```

56   const handlePriceFilter = (min, max) => {
57     setSearchParams(prev => {
58       const params = new URLSearchParams(prev);
59       if (min) params.set('minPrice', min);
60       else params.delete('minPrice');
61       if (max) params.set('maxPrice', max);
62       else params.delete('maxPrice');
63       return params;
64     });
65   };
66
67   // Reset tutti i filtri
68   const resetFilters = () => {
69     setSearchParams({});
70     setSearchInput('');
71   };
72
73   return (
74     <div className="product-search">
75       {/* Search Form */}
76       <form onSubmit={handleSearch}>
77         <input
78           type="text"
79           value={searchInput}
80           onChange={(e) => setSearchInput(e.target.value)}
81           placeholder="Cerca prodotti..."
82         />
83         <button type="submit">Cerca</button>
84       </form>
85
86       {/* Filters */}
87       <div className="filters">
88         <select
89           value={category}
90           onChange={(e) => handleCategoryChange(e.target.value)}
91         >
92           <option value="all">Tutte le categorie</option>
93           <option value="electronics">Elettronica</option>
94           <option value="clothing">Abbigliamento</option>
95           <option value="books">Libri</option>
96         </select>
97
98         <input
99           type="number"
100          placeholder="Prezzo min"
101          value={minPrice}
102          onChange={(e) => handlePriceFilter(e.target.value, maxPrice)}
103        />
104
105        <input
106          type="number"
107          placeholder="Prezzo max"
108          value={maxPrice}
109          onChange={(e) => handlePriceFilter(minPrice, e.target.value)}
110        />
111
112        <button onClick={resetFilters}>Reset Filtri</button>
113      </div>

```

```

114      {/* Active Filters Display */}
115      <div className="active-filters">
116        {query && <span>Ricerca: "{query}"</span>}
117        {category !== 'all' && <span>Categoria: {category}</span>}
118        {(minPrice || maxPrice) && (
119          <span>
120            Prezzo: {minPrice || '0'} - {maxPrice || '0'}
121          </span>
122        )}
123      </div>
124
125      {/* Results */}
126      {loading ? (
127        <div>Caricamento...</div>
128      ) : (
129        <div className="products">
130          {products.map(product => (
131            <div key={product.id} className="product">
132              <h3>{product.name}</h3>
133              <p>{product.price}</p>
134            </div>
135          ))}
136        </div>
137      )}
138    </div>
139  );
140 }
141 }

```

## 9.6 Route Nidificate (Nested Routes)

Le route nidificate permettono di creare layout con sottopagine.

### 9.6.1 Layout con Outlet

Listing 9.13: Layout componente con Outlet

```

1  import { Outlet, Link } from 'react-router-dom';
2
3  // Layout principale con sidebar
4  function DashboardLayout() {
5    return (
6      <div className="dashboard">
7        <aside className="sidebar">
8          <h2>Dashboard</h2>
9          <nav>
10             <Link to="/dashboard">Overview</Link>
11             <Link to="/dashboard/stats">Statistiche</Link>
12             <Link to="/dashboard/settings">Impostazioni</Link>
13             <Link to="/dashboard/users">Utenti</Link>
14           </nav>
15         </aside>
16
17         <main className="content">
18           {/* Qui vengono renderizzate le route figlie */}
19           <Outlet />

```

```

20     </main>
21   </div>
22 );
23 }
24
25 export default DashboardLayout;

```

### 9.6.2 Configurazione Route Nidificate

Listing 9.14: Setup di route nidificate

```

1  import { Routes, Route } from 'react-router-dom';
2  import DashboardLayout from '../layouts/DashboardLayout';
3  import DashboardHome from '../pages/dashboard/Home';
4  import DashboardStats from '../pages/dashboard/Stats';
5  import DashboardSettings from '../pages/dashboard/Settings';
6  import DashboardUsers from '../pages/dashboard/Users';
7
8  function App() {
9    return (
10     <Routes>
11       {/* Route principale con nested routes */}
12       <Route path="/dashboard" element={<DashboardLayout />}>
13         {/* Index route (default) */}
14         <Route index element={<DashboardHome />} />
15
16         {/* Nested routes */}
17         <Route path="stats" element={<DashboardStats />} />
18         <Route path="settings" element={<DashboardSettings />} />
19         <Route path="users" element={<DashboardUsers />} />
20       </Route>
21     </Routes>
22   );
23 }

```

### 9.6.3 Route Nidificate Multilivello

Listing 9.15: Nesting profondo con più livelli

```

1  import { Routes, Route } from 'react-router-dom';
2
3  function App() {
4    return (
5     <Routes>
6       <Route path="/" element={<RootLayout />}>
7         <Route index element={<Home />} />
8
9         {/* Blog con nested routes */}
10        <Route path="blog" element={<BlogLayout />}>
11          <Route index element={<BlogHome />} />
12          <Route path=":category" element={<CategoryLayout />}>
13            <Route index element={<CategoryHome />} />
14            <Route path=":postId" element={<BlogPost />} />
15          </Route>
16        </Route>
17      </Routes>

```

```

18     { /* Admin con nested routes */ }
19     <Route path="admin" element={<AdminLayout />}>
20       <Route index element={<AdminDashboard />} />
21       <Route path="users" element={<UsersLayout />}>
22         <Route index element={<UsersList />} />
23         <Route path=":userId" element={<UserDetail />} />
24         <Route path=":userId/edit" element={<UserEdit />} />
25       </Route>
26     </Route>
27   </Route>
28 </Routes>
29 );
30 }

```

## 9.7 Protected Routes (Route Protette)

Le route protette richiedono autenticazione per l'accesso.

### 9.7.1 Componente ProtectedRoute

Listing 9.16: Implementazione di route protetta

```

1 import { Navigate, useLocation } from 'react-router-dom';
2 import { useAuth } from '../hooks/useAuth'; // Custom hook per auth
3
4 function ProtectedRoute({ children }) {
5   const { isAuthenticated, loading } = useAuth();
6   const location = useLocation();
7
8   if (loading) {
9     return <div>Caricamento...</div>;
10  }
11
12  if (!isAuthenticated) {
13    // Redirect al login, salvando la location corrente
14    return <Navigate to="/login" state={{ from: location }} replace />;
15  }
16
17  return children;
18 }
19
20 export default ProtectedRoute;

```

### 9.7.2 Utilizzo delle Protected Routes

Listing 9.17: Setup di route protette

```

1 import { Routes, Route } from 'react-router-dom';
2 import ProtectedRoute from '../components/ProtectedRoute';
3
4 function App() {
5   return (
6     <Routes>
7       { /* Route pubbliche */ }
8       <Route path="/" element={<Home />} />

```

```

 9      <Route path="/login" element={<Login />} />
10      <Route path="/register" element={<Register />} />
11
12      { /* Route protette */ }
13      <Route
14          path="/dashboard"
15          element={
16              <ProtectedRoute>
17                  <Dashboard />
18              </ProtectedRoute>
19          }
20      />
21
22      <Route
23          path="/profile"
24          element={
25              <ProtectedRoute>
26                  <Profile />
27              </ProtectedRoute>
28          }
29      />
30
31      { /* Nested protected routes */ }
32      <Route
33          path="/admin"
34          element={
35              <ProtectedRoute>
36                  <AdminLayout />
37              </ProtectedRoute>
38          }
39      >
40          <Route index element={<AdminDashboard />} />
41          <Route path="/users" element={<AdminUsers />} />
42      </Route>
43  </Routes>
44  );
45  }

```

### 9.7.3 Hook useAuth Personalizzato

Listing 9.18: Custom hook per autenticazione

```

1  import { createContext, useContext, useState, useEffect } from 'react';
2
3  const AuthContext = createContext(null);
4
5  export function AuthProvider({ children }) {
6      const [user, setUser] = useState(null);
7      const [loading, setLoading] = useState(true);
8
9      useEffect(() => {
10         // Controlla se c'è un utente loggato (es. da localStorage)
11         const checkAuth = async () => {
12             try {
13                 const token = localStorage.getItem('token');
14                 if (token) {
15                     const response = await fetch('/api/auth/me', {

```

```

16         headers: { Authorization: 'Bearer ${token}' }
17     });
18     if (response.ok) {
19         const userData = await response.json();
20         setUser(userData);
21     }
22 }
23 } catch (error) {
24     console.error('Auth check failed:', error);
25 } finally {
26     setLoading(false);
27 }
28 };
29
30 checkAuth();
31 }, []);
32
33 const login = async (email, password) => {
34     const response = await fetch('/api/auth/login', {
35         method: 'POST',
36         headers: { 'Content-Type': 'application/json' },
37         body: JSON.stringify({ email, password })
38     });
39
40     if (response.ok) {
41         const { user, token } = await response.json();
42         localStorage.setItem('token', token);
43         setUser(user);
44         return true;
45     }
46     return false;
47 };
48
49 const logout = () => {
50     localStorage.removeItem('token');
51     setUser(null);
52 };
53
54 return (
55     <AuthContext.Provider
56         value={{
57             user,
58             isAuthenticated: !!user,
59             loading,
60             login,
61             logout
62         }}
63     >
64     {children}
65     </AuthContext.Provider>
66 );
67 }
68
69 export function useAuth() {
70     const context = useContext(AuthContext);
71     if (!context) {
72         throw new Error('useAuth deve essere usato dentro AuthProvider');
73     }

```

```
74   return context;
75 }
```

### 9.7.4 Route con Ruoli (Role-Based)

Listing 9.19: Route protette per ruolo

```
1  import { Navigate } from 'react-router-dom';
2  import { useAuth } from '../hooks/useAuth';
3
4  function RoleProtectedRoute({ children, allowedRoles }) {
5    const { user, isAuthenticated, loading } = useAuth();
6
7    if (loading) {
8      return <div>Caricamento...</div>;
9    }
10
11    if (!isAuthenticated) {
12      return <Navigate to="/login" replace />;
13    }
14
15    if (allowedRoles && !allowedRoles.includes(user.role)) {
16      return <Navigate to="/unauthorized" replace />;
17    }
18
19    return children;
20  }
21
22  // Utilizzo
23  function App() {
24    return (
25      <Routes>
26        { /* Solo admin */ }
27        <Route
28          path="/admin"
29          element={
30            <RoleProtectedRoute allowedRoles={['admin']>
31              <AdminPanel />
32            </RoleProtectedRoute>
33          }
34        />
35
36        { /* Admin e moderatori */ }
37        <Route
38          path="/moderation"
39          element={
40            <RoleProtectedRoute allowedRoles={['admin', 'moderator']>
41              <ModerationPanel />
42            </RoleProtectedRoute>
43          }
44        />
45
46        { /* Tutti gli utenti autenticati */ }
47        <Route
48          path="/dashboard"
49          element={
50            <RoleProtectedRoute>
```



```

51         <Dashboard />
52     </RoleProtectedRoute>
53 }
54 />
55 </Routes>
56 );
57 }

```

## 9.8 Lazy Loading e Code Splitting

Il lazy loading carica i componenti solo quando necessario, riducendo il bundle iniziale.

### 9.8.1 Setup Base con React.lazy

Listing 9.20: Lazy loading di base

```

1  import { lazy, Suspense } from 'react';
2  import { Routes, Route } from 'react-router-dom';
3
4  // Lazy import
5  const Home = lazy(() => import('./pages/Home'));
6  const About = lazy(() => import('./pages/About'));
7  const Dashboard = lazy(() => import('./pages/Dashboard'));
8  const Profile = lazy(() => import('./pages/Profile'));
9
10 function App() {
11     return (
12         <Suspense fallback={<div>Caricamento...</div>}>
13             <Routes>
14                 <Route path="/" element={<Home />} />
15                 <Route path="/about" element={<About />} />
16                 <Route path="/dashboard" element={<Dashboard />} />
17                 <Route path="/profile" element={<Profile />} />
18             </Routes>
19         </Suspense>
20     );
21 }

```

### 9.8.2 Loading Component Personalizzato

Listing 9.21: Componente loading personalizzato

```

1  function LoadingSpinner() {
2      return (
3          <div className="loading-container">
4              <div className="spinner"></div>
5              <p>Caricamento in corso...</p>
6          </div>
7      );
8  }
9
10 // CSS per spinner
11 // .spinner {
12 //     border: 4px solid #f3f3f3;
13 //     border-top: 4px solid #3498db;

```

```

14 //   border-radius: 50%;
15 //   width: 40px;
16 //   height: 40px;
17 //   animation: spin 1s linear infinite;
18 // }
19 //
20 // @keyframes spin {
21 //   0% { transform: rotate(0deg); }
22 //   100% { transform: rotate(360deg); }
23 // }
24
25 function App() {
26   return (
27     <Suspense fallback={<LoadingSpinner />}>
28       <Routes>
29         {/* routes */}
30       </Routes>
31     </Suspense>
32   );
33 }

```

### 9.8.3 Lazy Loading con Error Boundary

Listing 9.22: Gestione errori nel lazy loading

```

1 import { Component } from 'react';
2
3 class ErrorBoundary extends Component {
4   constructor(props) {
5     super(props);
6     this.state = { hasError: false, error: null };
7   }
8
9   static getDerivedStateFromError(error) {
10    return { hasError: true, error };
11  }
12
13  componentDidCatch(error, errorInfo) {
14    console.error('Error loading component:', error, errorInfo);
15  }
16
17  render() {
18    if (this.state.hasError) {
19      return (
20        <div className="error-container">
21          <h2>Errore nel caricamento</h2>
22          <p>{this.state.error?.message}</p>
23          <button onClick={() => window.location.reload()}>
24            Ricarica Pagina
25          </button>
26        </div>
27      );
28    }
29
30    return this.props.children;
31  }
32 }

```

```

33
34 // Utilizzo
35 function App() {
36   return (
37     <ErrorBoundary>
38       <Suspense fallback={<LoadingSpinner />}>
39         <Routes>
40           <Route path="/" element={<LazyHome />} />
41           <Route path="/dashboard" element={<LazyDashboard />} />
42         </Routes>
43       </Suspense>
44     </ErrorBoundary>
45   );
46 }

```

### 9.8.4 Preloading delle Route

Listing 9.23: Precaricamento route al hover

```

1  import { lazy } from 'react';
2  import { Link } from 'react-router-dom';
3
4  // Lazy imports
5  const Dashboard = lazy(() => import('./pages/Dashboard'));
6
7  // Funzione per preload
8  const preloadDashboard = () => {
9    import('./pages/Dashboard');
10 };
11
12 function Navigation() {
13   return (
14     <nav>
15       <Link to="/">Home</Link>
16
17       {/* Preload al mouseenter */}
18       <Link
19         to="/dashboard"
20         onMouseEnter={preloadDashboard}
21         onFocus={preloadDashboard}
22       >
23         Dashboard
24       </Link>
25     </nav>
26   );
27 }

```

## 9.9 useLocation Hook

useLocation fornisce accesso alla location corrente.

Listing 9.24: Utilizzo di useLocation

```

1  import { useLocation, useNavigate } from 'react-router-dom';
2  import { useEffect } from 'react';
3
4  function Analytics() {

```

```

5   const location = useLocation();
6
7   useEffect(() => {
8     // Track page view
9     console.log('Page view:', location.pathname);
10    // analytics.track('page_view', { path: location.pathname });
11  }, [location]);
12
13  return null;
14 }
15
16 function LoginPage() {
17   const location = useLocation();
18   const navigate = useNavigate();
19
20   const handleLogin = async (credentials) => {
21     const success = await login(credentials);
22
23     if (success) {
24       // Redirect alla pagina da cui l'utente veniva
25       const from = location.state?.from?.pathname || '/dashboard';
26       navigate(from, { replace: true });
27     }
28   };
29
30   return (
31     <div>
32       <h1>Login</h1>
33       {location.state?.from && (
34         <p>Devi effettuare il login per accedere a questa pagina</p>
35       )}
36       { /* form */ }
37     </div>
38   );
39 }

```

## 9.10 Applicazione Completa

### 9.10.1 Struttura del Progetto

Listing 9.25: Struttura directory consigliata

```

1  src/
2      components/
3          common/
4              LoadingSpinner.jsx
5              ErrorBoundary.jsx
6              Navigation.jsx
7          auth/
8              ProtectedRoute.jsx
9              RoleProtectedRoute.jsx
10         layout/
11             RootLayout.jsx
12             DashboardLayout.jsx
13             AdminLayout.jsx
14         pages/
15             Home.jsx

```

```

16         About.jsx
17         Login.jsx
18         Register.jsx
19         NotFound.jsx
20         dashboard/
21             Dashboard.jsx
22             Stats.jsx
23             Settings.jsx
24         admin/
25             AdminPanel.jsx
26             Users.jsx
27     hooks/
28         useAuth.js
29     context/
30         AuthContext.jsx
31 App.jsx
32 main.jsx

```

### 9.10.2 App.jsx Completa

Listing 9.26: Configurazione completa dell'app

```

1  import { lazy, Suspense } from 'react';
2  import { Routes, Route } from 'react-router-dom';
3  import { AuthProvider } from '../context/AuthContext';
4  import ErrorBoundary from '../components/common/ErrorBoundary';
5  import LoadingSpinner from '../components/common/LoadingSpinner';
6  import ProtectedRoute from '../components/auth/ProtectedRoute';
7  import RoleProtectedRoute from '../components/auth/RoleProtectedRoute';
8  import RootLayout from '../components/layout/RootLayout';
9  import DashboardLayout from '../components/layout/DashboardLayout';
10 import AdminLayout from '../components/layout/AdminLayout';
11
12 // Lazy loaded pages
13 const Home = lazy(() => import('../pages/Home'));
14 const About = lazy(() => import('../pages/About'));
15 const Login = lazy(() => import('../pages/Login'));
16 const Register = lazy(() => import('../pages/Register'));
17 const Dashboard = lazy(() => import('../pages/dashboard/Dashboard'));
18 const Stats = lazy(() => import('../pages/dashboard/Stats'));
19 const Settings = lazy(() => import('../pages/dashboard/Settings'));
20 const AdminPanel = lazy(() => import('../pages/admin/AdminPanel'));
21 const AdminUsers = lazy(() => import('../pages/admin/Users'));
22 const NotFound = lazy(() => import('../pages/NotFound'));
23
24 function App() {
25     return (
26         <AuthProvider>
27         <ErrorBoundary>
28             <Suspense fallback={<LoadingSpinner />}>
29                 <Routes>
30                     {/* Root layout */}
31                     <Route path="/" element={<RootLayout />}>
32                         {/* Public routes */}
33                         <Route index element={<Home />} />
34                         <Route path="about" element={<About />} />
35                         <Route path="login" element={<Login />} />

```

```
36     <Route path="register" element={<Register />} />
37
38     {/* Protected dashboard routes */}
39     <Route
40       path="dashboard"
41       element={
42         <ProtectedRoute>
43           <DashboardLayout />
44         </ProtectedRoute>
45       }
46     >
47       <Route index element={<Dashboard />} />
48       <Route path="stats" element={<Stats />} />
49       <Route path="settings" element={<Settings />} />
50     </Route>
51
52     {/* Admin routes (role-based) */}
53     <Route
54       path="admin"
55       element={
56         <RoleProtectedRoute allowedRoles={['admin']}>
57           <AdminLayout />
58         </RoleProtectedRoute>
59       }
60     >
61       <Route index element={<AdminPanel />} />
62       <Route path="users" element={<AdminUsers />} />
63     </Route>
64
65     {/* 404 */}
66     <Route path="*" element={<NotFound />} />
67   </Route>
68 </Routes>
69 </Suspense>
70 </ErrorBoundary>
71 </AuthProvider>
72 );
73 }
74
75 export default App;
```

## 9.11 Best Practices

### Best Practices per Routing

- Usa sempre **BrowserRouter** per SPA moderne
- Implementa lazy loading per route pesanti
- Proteggi route sensibili con autenticazione
- Usa **NavLink** per navigazione con stato attivo
- Gestisci sempre la route 404
- Usa **Suspense** con fallback significativi

- Implementa Error Boundaries per lazy loaded routes
- Traccia navigazione con analytics
- Usa nested routes per layout condivisi
- Preferisci useNavigate a window.location

## 9.12 Conclusioni

React Router v6 offre un sistema di routing potente e flessibile per applicazioni React. Con le sue API dichiarative, supporto per lazy loading, route protette e navigazione programmatica, permette di creare SPA complesse e performanti con ottima user experience.





# Capitolo 10

## State Management Avanzato

### 10.1 Introduzione allo State Management

Man mano che le applicazioni React crescono, gestire lo stato diventa più complesso. Passare props attraverso molti livelli (prop drilling) diventa ingombrante e difficile da mantenere. Le soluzioni di state management risolvono questo problema.

#### 10.1.1 Problemi dello State Locale

Listing 10.1: Problema: Prop Drilling

```
1 // Componente Root
2 function App() {
3   const [user, setUser] = useState(null);
4
5   return (
6     <Layout user={user} setUser={setUser}>
7       <Dashboard user={user} setUser={setUser} />
8     </Layout>
9   );
10 }
11
12 // Layout deve passare props anche se non le usa
13 function Layout({ user, setUser, children }) {
14   return (
15     <div>
16       <Header user={user} setUser={setUser} />
17       <main>{children}</main>
18     </div>
19   );
20 }
21
22 // Header finalmente usa le props
23 function Header({ user, setUser }) {
24   return (
25     <header>
26       {user ? (
27         <UserMenu user={user} onLogout={() => setUser(null)} />
28       ) : (
29         <LoginButton />
30       )}
31     </header>
32   );
33 }
```

```
34 // Problema: Layout passa props che non usa (prop drilling)
35
```

### 10.1.2 Quando Usare State Management Globale

#### Quando NON serve state management globale

- Stato usato solo da un componente
- Stato condiviso tra pochi componenti vicini
- Form state locale
- UI state temporaneo (modals, tooltips)

#### Quando serve state management globale

- Autenticazione utente (user, token, permissions)
- Temi e preferenze UI
- Carrello e-commerce
- Notifiche e toast messages
- Dati condivisi da molti componenti
- Cache di dati API

## 10.2 Context API

Context API è la soluzione built-in di React per condividere stato senza prop drilling.

### 10.2.1 Creazione di un Context

Listing 10.2: Context base per tema

```
1 import { createContext, useContext, useState } from 'react';
2
3 // 1. Crea il Context
4 const ThemeContext = createContext(null);
5
6 // 2. Provider Component
7 export function ThemeProvider({ children }) {
8   const [theme, setTheme] = useState('light');
9
10  const toggleTheme = () => {
11    setTheme(prev => prev === 'light' ? 'dark' : 'light');
12  };
13
14  const value = {
15    theme,
16    toggleTheme
17  };
18
19  return (
```

```
20     <ThemeContext.Provider value={value}>
21       {children}
22     </ThemeContext.Provider>
23   );
24 }
25
26 // 3. Custom Hook per usare il context
27 export function useTheme() {
28   const context = useContext(ThemeContext);
29
30   if (context === null) {
31     throw new Error('useTheme deve essere usato dentro ThemeProvider');
32   }
33
34   return context;
35 }
```

### 10.2.2 Utilizzo del Context

Listing 10.3: Usare il context nell'app

```
1 // main.jsx
2 import { ThemeProvider } from './context/ThemeContext';
3
4 ReactDOM.createRoot(document.getElementById('root')).render(
5   <ThemeProvider>
6     <App />
7   </ThemeProvider>
8 );
9
10 // Componente che usa il tema
11 function Header() {
12   const { theme, toggleTheme } = useTheme();
13
14   return (
15     <header className={theme}>
16       <h1>My App</h1>
17       <button onClick={toggleTheme}>
18         Toggle Theme (Current: {theme})
19       </button>
20     </header>
21   );
22 }
23
24 // Altro componente che usa il tema
25 function Card({ children }) {
26   const { theme } = useTheme();
27
28   return (
29     <div className={`card card-${theme}`}>
30       {children}
31     </div>
32   );
33 }
```

### 10.2.3 Context per Autenticazione

Listing 10.4: Auth Context completo

```

1 import { createContext, useContext, useState, useEffect } from 'react';
2
3 const AuthContext = createContext(null);
4
5 export function AuthProvider({ children }) {
6   const [user, setUser] = useState(null);
7   const [loading, setLoading] = useState(true);
8
9   // Controlla auth al mount
10  useEffect(() => {
11    const checkAuth = async () => {
12      const token = localStorage.getItem('authToken');
13
14      if (token) {
15        try {
16          const response = await fetch('/api/auth/me', {
17            headers: { Authorization: 'Bearer ${token}' }
18          });
19
20          if (response.ok) {
21            const userData = await response.json();
22            setUser(userData);
23          } else {
24            localStorage.removeItem('authToken');
25          }
26        } catch (error) {
27          console.error('Auth check failed:', error);
28        }
29      }
30
31      setLoading(false);
32    };
33
34    checkAuth();
35  }, []);
36
37  const login = async (email, password) => {
38    try {
39      const response = await fetch('/api/auth/login', {
40        method: 'POST',
41        headers: { 'Content-Type': 'application/json' },
42        body: JSON.stringify({ email, password })
43      });
44
45      if (!response.ok) {
46        throw new Error('Login failed');
47      }
48
49      const { user, token } = await response.json();
50      localStorage.setItem('authToken', token);
51      setUser(user);
52      return { success: true };
53    } catch (error) {
54      return { success: false, error: error.message };
55    }
56  };
57

```

```

58   const logout = () => {
59     localStorage.removeItem('authToken');
60     setUser(null);
61   };
62
63   const register = async (userData) => {
64     try {
65       const response = await fetch('/api/auth/register', {
66         method: 'POST',
67         headers: { 'Content-Type': 'application/json' },
68         body: JSON.stringify(userData)
69       });
70
71       if (!response.ok) {
72         throw new Error('Registration failed');
73       }
74
75       const { user, token } = await response.json();
76       localStorage.setItem('authToken', token);
77       setUser(user);
78       return { success: true };
79     } catch (error) {
80       return { success: false, error: error.message };
81     }
82   };
83
84   const value = {
85     user,
86     loading,
87     isAuthenticated: !!user,
88     login,
89     logout,
90     register
91   };
92
93   return (
94     <AuthContext.Provider value={value}>
95       {children}
96     </AuthContext.Provider>
97   );
98 }
99
100 export function useAuth() {
101   const context = useContext(AuthContext);
102
103   if (context === null) {
104     throw new Error('useAuth deve essere usato dentro AuthProvider');
105   }
106
107   return context;
108 }

```

### 10.2.4 Context con Reducer

Per logica complessa, combina Context con useReducer.

Listing 10.5: Context con useReducer per carrello

```
1 import { createContext, useContext, useReducer } from 'react';
2
3 const CartContext = createContext(null);
4
5 // Reducer
6 function cartReducer(state, action) {
7   switch (action.type) {
8     case 'ADD_ITEM':
9       const existingIndex = state.items.findIndex(
10         item => item.id === action.payload.id
11       );
12
13       if (existingIndex >= 0) {
14         const newItems = [...state.items];
15         newItems[existingIndex].quantity += 1;
16         return { ...state, items: newItems };
17       }
18
19       return {
20         ...state,
21         items: [...state.items, { ...action.payload, quantity: 1 }]
22       };
23
24     case 'REMOVE_ITEM':
25       return {
26         ...state,
27         items: state.items.filter(item => item.id !== action.payload)
28       };
29
30     case 'UPDATE_QUANTITY':
31       return {
32         ...state,
33         items: state.items.map(item =>
34           item.id === action.payload.id
35             ? { ...item, quantity: action.payload.quantity }
36             : item
37         )
38       };
39
40     case 'CLEAR_CART':
41       return { ...state, items: [] };
42
43     default:
44       return state;
45   }
46 }
47
48 // Provider
49 export function CartProvider({ children }) {
50   const [state, dispatch] = useReducer(cartReducer, { items: [] });
51
52   // Selectors
53   const totalItems = state.items.reduce(
54     (sum, item) => sum + item.quantity,
55     0
56   );
57
58   const totalPrice = state.items.reduce(
```

```

59     (sum, item) => sum + (item.price * item.quantity),
60     0
61   );
62
63   // Actions
64   const addItem = (item) => {
65     dispatch({ type: 'ADD_ITEM', payload: item });
66   };
67
68   const removeItem = (itemId) => {
69     dispatch({ type: 'REMOVE_ITEM', payload: itemId });
70   };
71
72   const updateQuantity = (itemId, quantity) => {
73     if (quantity <= 0) {
74       removeItem(itemId);
75     } else {
76       dispatch({
77         type: 'UPDATE_QUANTITY',
78         payload: { id: itemId, quantity }
79       });
80     }
81   };
82
83   const clearCart = () => {
84     dispatch({ type: 'CLEAR_CART' });
85   };
86
87   const value = {
88     items: state.items,
89     totalItems,
90     totalPrice,
91     addItem,
92     removeItem,
93     updateQuantity,
94     clearCart
95   };
96
97   return (
98     <CartContext.Provider value={value}>
99       {children}
100     </CartContext.Provider>
101   );
102 }
103
104 export function useCart() {
105   const context = useContext(CartContext);
106
107   if (context === null) {
108     throw new Error('useCart deve essere usato dentro CartProvider');
109   }
110
111   return context;
112 }

```

### 10.2.5 Utilizzo del Cart Context

Listing 10.6: Componenti che usano il carrello

```
1 // Componente Product
2 function ProductCard({ product }) {
3   const { addItem } = useCart();
4
5   return (
6     <div className="product-card">
7       <h3>{product.name}</h3>
8       <p>{product.price} </p>
9       <button onClick={() => addItem(product)}>
10         Aggiungi al Carrello
11       </button>
12     </div>
13   );
14 }
15
16 // Componente CartIcon
17 function CartIcon() {
18   const { totalItems, totalPrice } = useCart();
19
20   return (
21     <div className="cart-icon">
22       <ShoppingCartIcon />
23       <span className="badge">{totalItems}</span>
24       <span className="price">{totalPrice.toFixed(2)} </span>
25     </div>
26   );
27 }
28
29 // Componente Cart
30 function Cart() {
31   const { items, updateQuantity, removeItem, clearCart } = useCart();
32
33   return (
34     <div className="cart">
35       <h2>Carrello</h2>
36
37       {items.length === 0 ? (
38         <p>Il carrello è vuoto</p>
39       ) : (
40         <>
41           {items.map(item => (
42             <div key={item.id} className="cart-item">
43               <h4>{item.name}</h4>
44               <p>{item.price} </p>
45
46               <div className="quantity-controls">
47                 <button onClick={() => updateQuantity(item.id, item.
48                   quantity - 1)}>
49                   -
50                 </button>
51                 <span>{item.quantity}</span>
52                 <button onClick={() => updateQuantity(item.id, item.
53                   quantity + 1)}>
54                   +
55                 </button>
56               </div>
57             </div>
58           ))}
59         </>
60       )}
61     </div>
62   );
63 }
```



```

56         <button onClick={() => removeItem(item.id)}>
57             Rimuovi
58         </button>
59     </div>
60     )}}
61
62     <button onClick={clearCart}>Svuota Carrello</button>
63 </>
64     )}
65 </div>
66 );
67 }

```

### 10.2.6 Performance: Splitting Context

Per evitare re-render inutili, separa context per tipo di dato.

Listing 10.7: Separazione context per performance

```

1  // SBAGLIATO: Un unico context con troppi dati
2  const AppContext = createContext(null);
3
4  function AppProvider({ children }) {
5      const [user, setUser] = useState(null);
6      const [theme, setTheme] = useState('light');
7      const [notifications, setNotifications] = useState([]);
8      const [settings, setSettings] = useState({});
9
10     // Ogni cambiamento ri-renderizza tutti i consumer!
11     return (
12         <AppContext.Provider value={{
13             user, setUser,
14             theme, setTheme,
15             notifications, setNotifications,
16             settings, setSettings
17         }}>
18             {children}
19         </AppContext.Provider>
20     );
21 }
22
23 // CORRETTO: Context separati
24 // UserContext.jsx
25 const UserContext = createContext(null);
26 export function UserProvider({ children }) {
27     const [user, setUser] = useState(null);
28     return (
29         <UserContext.Provider value={{ user, setUser }}>
30             {children}
31         </UserContext.Provider>
32     );
33 }
34
35 // ThemeContext.jsx
36 const ThemeContext = createContext(null);
37 export function ThemeProvider({ children }) {
38     const [theme, setTheme] = useState('light');
39     return (

```

```
40     <ThemeContext.Provider value={{ theme, setTheme }}>
41       {children}
42     </ThemeContext.Provider>
43   );
44 }
45
46 // App.jsx - Composizione providers
47 function App() {
48   return (
49     <UserProvider>
50       <ThemeProvider>
51         <NotificationProvider>
52           <SettingsProvider>
53             <AppContent />
54           </SettingsProvider>
55         </NotificationProvider>
56       </ThemeProvider>
57     </UserProvider>
58   );
59 }
```

## 10.3 Redux Toolkit

Redux Toolkit è la soluzione ufficiale per Redux, semplificando molto il boilerplate.

### 10.3.1 Installazione

Listing 10.8: Installazione Redux Toolkit

```
1 npm install @reduxjs/toolkit react-redux
2
3 # o con yarn
4 yarn add @reduxjs/toolkit react-redux
```

### 10.3.2 Setup Store

Listing 10.9: Configurazione base dello store

```
1 // store/store.js
2 import { configureStore } from '@reduxjs/toolkit';
3 import counterReducer from '../slices/counterSlice';
4 import authReducer from '../slices/authSlice';
5 import cartReducer from '../slices/cartSlice';
6
7 export const store = configureStore({
8   reducer: {
9     counter: counterReducer,
10    auth: authReducer,
11    cart: cartReducer
12  }
13 });
```

### 10.3.3 Provider Setup

Listing 10.10: Wrap app con Provider

```
1 // main.jsx
2 import { Provider } from 'react-redux';
3 import { store } from './store/store';
4
5 ReactDOM.createRoot(document.getElementById('root')).render(
6   <Provider store={store}>
7     <App />
8   </Provider>
9 );
```

### 10.3.4 Creazione di uno Slice

Listing 10.11: Counter slice esempio

```
1 // store/slices/counterSlice.js
2 import { createSlice } from '@reduxjs/toolkit';
3
4 const counterSlice = createSlice({
5   name: 'counter',
6
7   initialState: {
8     value: 0
9   },
10
11   reducers: {
12     increment: (state) => {
13       state.value += 1; // Immer permette mutazioni dirette
14     },
15
16     decrement: (state) => {
17       state.value -= 1;
18     },
19
20     incrementByAmount: (state, action) => {
21       state.value += action.payload;
22     },
23
24     reset: (state) => {
25       state.value = 0;
26     }
27   }
28 });
29
30 // Export actions
31 export const { increment, decrement, incrementByAmount, reset } =
32   counterSlice.actions;
33
34 // Export reducer
35 export default counterSlice.reducer;
36
37 // Selectors
38 export const selectCount = (state) => state.counter.value;
```

### 10.3.5 Utilizzo in Componenti

Listing 10.12: Usare Redux in componenti

```
1 import { useSelector, useDispatch } from 'react-redux';
2 import {
3   increment,
4   decrement,
5   incrementByAmount,
6   reset,
7   selectCount
8 } from './store/slices/counterSlice';
9
10 function Counter() {
11   const count = useSelector(selectCount);
12   const dispatch = useDispatch();
13
14   return (
15     <div>
16       <h2>Count: {count}</h2>
17
18       <button onClick={() => dispatch(increment())}>+</button>
19       <button onClick={() => dispatch(decrement())}>-</button>
20       <button onClick={() => dispatch(incrementByAmount(5))}>+5</button>
21       <button onClick={() => dispatch(reset())}>Reset</button>
22     </div>
23   );
24 }
```

### 10.3.6 Slice Complesso: Autenticazione

Listing 10.13: Auth slice con async thunks

```
1 // store/slices/authSlice.js
2 import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
3
4 // Async thunk per login
5 export const loginUser = createAsyncThunk(
6   'auth/login',
7   async ({ email, password }, { rejectWithValue }) => {
8     try {
9       const response = await fetch('/api/auth/login', {
10         method: 'POST',
11         headers: { 'Content-Type': 'application/json' },
12         body: JSON.stringify({ email, password })
13       });
14
15       if (!response.ok) {
16         throw new Error('Login failed');
17       }
18
19       const data = await response.json();
20       localStorage.setItem('token', data.token);
21       return data;
22     } catch (error) {
23       return rejectWithValue(error.message);
24     }
25 }
```

```

26 );
27
28 // Async thunk per fetch user
29 export const fetchUser = createAsyncThunk(
30   'auth/fetchUser',
31   async (_, { rejectWithValue }) => {
32     try {
33       const token = localStorage.getItem('token');
34
35       if (!token) {
36         throw new Error('No token');
37       }
38
39       const response = await fetch('/api/auth/me', {
40         headers: { Authorization: `Bearer ${token}` }
41       });
42
43       if (!response.ok) {
44         throw new Error('Failed to fetch user');
45       }
46
47       return await response.json();
48     } catch (error) {
49       return rejectWithValue(error.message);
50     }
51   }
52 );
53
54 const authSlice = createSlice({
55   name: 'auth',
56
57   initialState: {
58     user: null,
59     token: localStorage.getItem('token'),
60     loading: false,
61     error: null
62   },
63
64   reducers: {
65     logout: (state) => {
66       state.user = null;
67       state.token = null;
68       state.error = null;
69       localStorage.removeItem('token');
70     },
71
72     clearError: (state) => {
73       state.error = null;
74     }
75   },
76
77   extraReducers: (builder) => {
78     builder
79       // Login
80       .addCase(loginUser.pending, (state) => {
81         state.loading = true;
82         state.error = null;
83       })

```

```

84     .addCase(loginUser.fulfilled, (state, action) => {
85         state.loading = false;
86         state.user = action.payload.user;
87         state.token = action.payload.token;
88         state.error = null;
89     })
90     .addCase(loginUser.rejected, (state, action) => {
91         state.loading = false;
92         state.error = action.payload;
93     })
94
95     // Fetch User
96     .addCase(fetchUser.pending, (state) => {
97         state.loading = true;
98     })
99     .addCase(fetchUser.fulfilled, (state, action) => {
100         state.loading = false;
101         state.user = action.payload;
102     })
103     .addCase(fetchUser.rejected, (state, action) => {
104         state.loading = false;
105         state.token = null;
106         state.error = action.payload;
107         localStorage.removeItem('token');
108     });
109 }
110 });
111
112 export const { logout, clearError } = authSlice.actions;
113 export default authSlice.reducer;
114
115 // Selectors
116 export const selectUser = (state) => state.auth.user;
117 export const selectIsAuthenticated = (state) => !!state.auth.user;
118 export const selectAuthLoading = (state) => state.auth.loading;
119 export const selectAuthError = (state) => state.auth.error;

```

### 10.3.7 Utilizzo Auth Slice

Listing 10.14: Login component con Redux

```

1  import { useState } from 'react';
2  import { useDispatch, useSelector } from 'react-redux';
3  import { useNavigate } from 'react-router-dom';
4  import {
5      loginUser,
6      selectAuthLoading,
7      selectAuthError,
8      clearError
9  } from './store/slices/authSlice';
10
11  function LoginPage() {
12      const [email, setEmail] = useState('');
13      const [password, setPassword] = useState('');
14
15      const dispatch = useDispatch();
16      const navigate = useNavigate();

```

```

17  const loading = useSelector(selectAuthLoading);
18  const error = useSelector(selectAuthError);
19
20  const handleSubmit = async (e) => {
21    e.preventDefault();
22
23    const result = await dispatch(loginUser({ email, password }));
24
25    if (loginUser.fulfilled.match(result)) {
26      navigate('/dashboard');
27    }
28  };
29
30  useEffect(() => {
31    // Pulisci errori quando il componente si smonta
32    return () => {
33      dispatch(clearError());
34    };
35  }, [dispatch]);
36
37  return (
38    <div>
39      <h1>Login</h1>
40
41      {error && (
42        <div className="error">
43          {error}
44          <button onClick={() => dispatch(clearError())}>X</button>
45        </div>
46      )}
47
48      <form onSubmit={handleSubmit}>
49        <input
50          type="email"
51          value={email}
52          onChange={(e) => setEmail(e.target.value)}
53          placeholder="Email"
54          required
55        />
56
57        <input
58          type="password"
59          value={password}
60          onChange={(e) => setPassword(e.target.value)}
61          placeholder="Password"
62          required
63        />
64
65        <button type="submit" disabled={loading}>
66          {loading ? 'Loading...' : 'Login'}
67        </button>
68      </form>
69    </div>
70  );
71 }

```

### 10.3.8 Cart Slice Completo

Listing 10.15: Carrello e-commerce con Redux

```
1 // store/slices/cartSlice.js
2 import { createSlice } from '@reduxjs/toolkit';
3
4 const cartSlice = createSlice({
5   name: 'cart',
6
7   initialState: {
8     items: [],
9     isOpen: false
10  },
11
12  reducers: {
13    addItem: (state, action) => {
14      const existingItem = state.items.find(
15        item => item.id === action.payload.id
16      );
17
18      if (existingItem) {
19        existingItem.quantity += 1;
20      } else {
21        state.items.push({ ...action.payload, quantity: 1 });
22      }
23    },
24
25    removeItem: (state, action) => {
26      state.items = state.items.filter(
27        item => item.id !== action.payload
28      );
29    },
30
31    updateQuantity: (state, action) => {
32      const { id, quantity } = action.payload;
33      const item = state.items.find(item => item.id === id);
34
35      if (item) {
36        if (quantity <= 0) {
37          state.items = state.items.filter(item => item.id !== id);
38        } else {
39          item.quantity = quantity;
40        }
41      }
42    },
43
44    clearCart: (state) => {
45      state.items = [];
46    },
47
48    toggleCart: (state) => {
49      state.isOpen = !state.isOpen;
50    },
51
52    closeCart: (state) => {
53      state.isOpen = false;
54    }
55  }
56 }
```



```

56  });
57
58  export const {
59    addItem,
60    removeItem,
61    updateQuantity,
62    clearCart,
63    toggleCart,
64    closeCart
65  } = cartSlice.actions;
66
67  export default cartSlice.reducer;
68
69  // Selectors con memoization
70  export const selectCartItems = (state) => state.cart.items;
71  export const selectIsCartOpen = (state) => state.cart.isOpen;
72
73  export const selectTotalItems = (state) =>
74    state.cart.items.reduce((sum, item) => sum + item.quantity, 0);
75
76  export const selectTotalPrice = (state) =>
77    state.cart.items.reduce(
78      (sum, item) => sum + (item.price * item.quantity),
79      0
80    );

```

### 10.3.9 RTK Query per API

RTK Query semplifica il fetching e caching di dati API.

Listing 10.16: API slice con RTK Query

```

1  // store/api/apiSlice.js
2  import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'
3
4  export const apiSlice = createApi({
5    reducerPath: 'api',
6
7    baseQuery: fetchBaseQuery({
8      baseUrl: '/api',
9      prepareHeaders: (headers, { getState }) => {
10        const token = getState().auth.token;
11        if (token) {
12          headers.set('authorization', `Bearer ${token}`);
13        }
14        return headers;
15      }
16    }),
17
18    tagTypes: ['User', 'Product', 'Order'],
19
20    endpoints: (builder) => ({
21      // Get all products
22      getProducts: builder.query({
23        query: () => '/products',
24        providesTags: ['Product']
25      }),

```

```

26
27 // Get single product
28 getProduct: builder.query({
29   query: (id) => `/products/${id}`,
30   providesTags: (result, error, id) => [{ type: 'Product', id }]
31 }),
32
33 // Create product
34 createProduct: builder.mutation({
35   query: (newProduct) => ({
36     url: '/products',
37     method: 'POST',
38     body: newProduct
39   }),
40   invalidatesTags: ['Product']
41 }),
42
43 // Update product
44 updateProduct: builder.mutation({
45   query: ({ id, ...updates }) => ({
46     url: `/products/${id}`,
47     method: 'PUT',
48     body: updates
49   }),
50   invalidatesTags: (result, error, { id }) => [{ type: 'Product', id
51     }]
52   }),
53
54 // Delete product
55 deleteProduct: builder.mutation({
56   query: (id) => ({
57     url: `/products/${id}`,
58     method: 'DELETE'
59   }),
60   invalidatesTags: ['Product']
61   }),
62
63 // Get user orders
64 getUserOrders: builder.query({
65   query: () => '/orders',
66   providesTags: ['Order']
67   })
68   });
69
70 export const {
71   useGetProductsQuery,
72   useGetProductQuery,
73   useCreateProductMutation,
74   useUpdateProductMutation,
75   useDeleteProductMutation,
76   useGetUserOrdersQuery
77 } = apiSlice;

```

### 10.3.10 Store con RTK Query

Listing 10.17: Store configuration con RTK Query

```
1 // store/store.js
2 import { configureStore } from '@reduxjs/toolkit';
3 import { apiSlice } from '../api/apiSlice';
4 import authReducer from '../slices/authSlice';
5 import cartReducer from '../slices/cartSlice';
6
7 export const store = configureStore({
8   reducer: {
9     [apiSlice.reducerPath]: apiSlice.reducer,
10    auth: authReducer,
11    cart: cartReducer
12  },
13
14  middleware: (getDefaultMiddleware) =>
15    getDefaultMiddleware().concat(apiSlice.middleware)
16 });
```

### 10.3.11 Utilizzo RTK Query

Listing 10.18: Componente con RTK Query

```
1 import {
2   useGetProductsQuery,
3   useCreateProductMutation,
4   useDeleteProductMutation
5 } from '../store/api/apiSlice';
6
7 function ProductList() {
8   const {
9     data: products,
10    isLoading,
11    isError,
12    error,
13    refetch
14  } = useGetProductsQuery();
15
16  const [createProduct, { isLoading: isCreating }] =
17    useCreateProductMutation();
18
19  const [deleteProduct, { isLoading: isDeleting }] =
20    useDeleteProductMutation();
21
22  const handleCreate = async () => {
23    try {
24      await createProduct({
25        name: 'New Product',
26        price: 99.99
27      }).unwrap();
28      // Successo
29    } catch (error) {
30      console.error('Failed to create:', error);
31    }
32  };
33
34  const handleDelete = async (id) => {
35    try {
```

```

36     await deleteProduct(id).unwrap();
37   } catch (error) {
38     console.error('Failed to delete:', error);
39   }
40 };
41
42 if (isLoading) return <div>Loading...</div>;
43 if (isError) return <div>Error: {error.message}</div>;
44
45 return (
46   <div>
47     <h1>Products</h1>
48
49     <button onClick={handleCreate} disabled={isCreating}>
50       {isCreating ? 'Creating...' : 'Add Product'}
51     </button>
52
53     <button onClick={refetch}>Refresh</button>
54
55     <ul>
56       {products.map(product => (
57         <li key={product.id}>
58           {product.name} - {product.price}
59           <button
60             onClick={() => handleDelete(product.id)}
61             disabled={isDeleting}
62           >
63             Delete
64           </button>
65         </li>
66       ))}
67     </ul>
68   </div>
69 );
70 }

```

## 10.4 Zustand

Zustand è una libreria di state management leggera e semplice, alternativa a Redux con meno boilerplate.

### 10.4.1 Installazione

Listing 10.19: Installazione Zustand

```

1 npm install zustand
2
3 # o con yarn
4 yarn add zustand

```

### 10.4.2 Store Base

Listing 10.20: Creazione di uno store Zustand

```

1 // store/useCounterStore.js

```

```

2 import { create } from 'zustand';
3
4 const useCounterStore = create((set) => ({
5   // State
6   count: 0,
7
8   // Actions
9   increment: () => set((state) => ({ count: state.count + 1 })),
10  decrement: () => set((state) => ({ count: state.count - 1 })),
11  incrementByAmount: (amount) =>
12    set((state) => ({ count: state.count + amount })),
13  reset: () => set({ count: 0 })
14 }));
15
16 export default useCounterStore;

```

### 10.4.3 Utilizzo in Componenti

Listing 10.21: Usare Zustand in componenti

```

1 import useCounterStore from './store/useCounterStore';
2
3 function Counter() {
4   // Seleziona solo ciò che serve
5   const count = useCounterStore((state) => state.count);
6   const increment = useCounterStore((state) => state.increment);
7   const decrement = useCounterStore((state) => state.decrement);
8   const reset = useCounterStore((state) => state.reset);
9
10  return (
11    <div>
12      <h2>Count: {count}</h2>
13      <button onClick={increment}>+</button>
14      <button onClick={decrement}>-</button>
15      <button onClick={reset}>Reset</button>
16    </div>
17  );
18 }

```

### 10.4.4 Auth Store con Zustand

Listing 10.22: Store autenticazione completo

```

1 // store/useAuthStore.js
2 import { create } from 'zustand';
3 import { persist } from 'zustand/middleware';
4
5 const useAuthStore = create(
6   persist(
7     (set, get) => ({
8       // State
9       user: null,
10      token: null,
11      loading: false,
12      error: null,
13

```

```

14 // Actions
15 login: async (email, password) => {
16   set({ loading: true, error: null });
17
18   try {
19     const response = await fetch('/api/auth/login', {
20       method: 'POST',
21       headers: { 'Content-Type': 'application/json' },
22       body: JSON.stringify({ email, password })
23     });
24
25     if (!response.ok) {
26       throw new Error('Login failed');
27     }
28
29     const data = await response.json();
30
31     set({
32       user: data.user,
33       token: data.token,
34       loading: false,
35       error: null
36     });
37
38     return { success: true };
39   } catch (error) {
40     set({
41       loading: false,
42       error: error.message
43     });
44     return { success: false, error: error.message };
45   }
46 },
47
48 logout: () => {
49   set({
50     user: null,
51     token: null,
52     error: null
53   });
54 },
55
56 clearError: () => set({ error: null })
57 },
58 {
59   name: 'auth-storage',
60   partialize: (state) => ({
61     token: state.token
62   })
63 }
64 )
65 );
66
67 export default useAuthStore;

```

### 10.4.5 Middleware in Zustand

Listing 10.23: Uso di middleware: persist e devtools

```
1 import { create } from 'zustand';
2 import { persist, devtools } from 'zustand/middleware';
3
4 const useStore = create(
5   devtools(
6     persist(
7       (set, get) => ({
8         bears: 0,
9         increasePopulation: () =>
10           set((state) => ({ bears: state.bears + 1 })),
11       }),
12       {
13         name: 'animals-storage',
14       }
15     ),
16     { name: 'AnimalStore' }
17   )
18 );
```

## 10.5 Confronto: Context vs Redux vs Zustand

### Context API

#### Vantaggi:

- Built-in, nessuna dipendenza esterna
- Semplice da configurare
- Zero boilerplate

#### Svantaggi:

- Performance issues con molti consumer
- Nessun DevTools integrato
- Difficile testare

**Usa quando:** State semplice, piccole applicazioni

### Redux Toolkit

#### Vantaggi:

- Ecosystem maturo
- Ottimi DevTools
- Time-travel debugging
- RTK Query per data fetching

#### Svantaggi:

- Curva di apprendimento ripida
- Più boilerplate

**Usa quando:** Applicazioni grandi e complesse, team grandi

### Zustand

#### Vantaggi:

- Estremamente semplice
- Minimal boilerplate
- Ottima performance
- No Provider needed
- Bundle size piccolo

#### Svantaggi:

- Ecosystem più piccolo
- Community più piccola

**Usa quando:** Semplicità senza sacrificare features, app medie

## 10.6 Best Practices

### Checklist State Management

- Usa state locale quando possibile
- Normalizza i dati per evitare duplicazioni
- Separa UI state da server state
- Usa selectors memoizzati per performance
- Evita prop drilling con Context o state management
- Testa la logica di business separatamente
- Usa TypeScript per type safety
- Monitora performance con DevTools
- Implementa error handling robusto
- Documenta struttura dello state

## 10.7 Conclusioni

La scelta della soluzione di state management dipende dalle esigenze dell'applicazione:

- **Context API:** Per state semplice e piccole app
- **Redux Toolkit:** Per app enterprise con requisiti complessi
- **Zustand:** Per il miglior balance tra semplicità e features



Non esiste una soluzione universale: valuta complessità dell'app, dimensione del team, e requisiti di performance per scegliere la soluzione giusta.



# Capitolo 11

## Integrazione con API

### 11.1 Introduzione

L'integrazione con API è fondamentale nelle moderne applicazioni React. Questo capitolo copre le tecniche per effettuare richieste HTTP, gestire loading states, error handling e caching dei dati.

#### 11.1.1 Metodi Principali

- **Fetch API:** API nativa del browser
- **Axios:** Libreria HTTP con features avanzate
- **React Query/TanStack Query:** Libreria per data fetching e caching

### 11.2 Fetch API

Fetch è l'API nativa per richieste HTTP nei browser moderni.

#### 11.2.1 GET Request Base

Listing 11.1: Fetch GET base con useState e useEffect

```
1 import { useState, useEffect } from 'react';
2
3 function UserList() {
4   const [users, setUsers] = useState([]);
5   const [loading, setLoading] = useState(true);
6   const [error, setError] = useState(null);
7
8   useEffect(() => {
9     fetch('https://jsonplaceholder.typicode.com/users')
10      .then(response => {
11        if (!response.ok) {
12          throw new Error('Network response was not ok');
13        }
14        return response.json();
15      })
16      .then(data => {
17        setUsers(data);
18        setLoading(false);
19      })
20      .catch(error => {
21        setError(error.message);
```

```

22         setLoading(false);
23     });
24 }, []);
25
26 if (loading) return <div>Caricamento...</div>;
27 if (error) return <div>Errore: {error}</div>;
28
29 return (
30     <ul>
31         {users.map(user => (
32             <li key={user.id}>{user.name}</li>
33         ))}
34     </ul>
35 );
36 }

```

### 11.2.2 Async/Await con Fetch

Listing 11.2: Fetch con async/await

```

1  import { useState, useEffect } from 'react';
2
3  function Posts() {
4      const [posts, setPosts] = useState([]);
5      const [loading, setLoading] = useState(true);
6      const [error, setError] = useState(null);
7
8      useEffect(() => {
9          const fetchPosts = async () => {
10              try {
11                  setLoading(true);
12
13                  const response = await fetch(
14                      'https://jsonplaceholder.typicode.com/posts'
15                  );
16
17                  if (!response.ok) {
18                      throw new Error('HTTP error! status: ${response.status}');
19                  }
20
21                  const data = await response.json();
22                  setPosts(data);
23              } catch (error) {
24                  setError(error.message);
25              } finally {
26                  setLoading(false);
27              }
28          };
29
30          fetchPosts();
31      }, []);
32
33      if (loading) return <div>Caricamento...</div>;
34      if (error) return <div>Errore: {error}</div>;
35
36      return (
37          <div>

```

```

38     {posts.map(post => (
39       <article key={post.id}>
40         <h3>{post.title}</h3>
41         <p>{post.body}</p>
42       </article>
43     )}}
44   </div>
45 );
46 }

```

### 11.2.3 POST Request

Listing 11.3: Fetch POST per creare risorsa

```

1  import { useState } from 'react';
2
3  function CreatePost() {
4    const [title, setTitle] = useState('');
5    const [body, setBody] = useState('');
6    const [loading, setLoading] = useState(false);
7    const [error, setError] = useState(null);
8    const [success, setSuccess] = useState(false);
9
10   const handleSubmit = async (e) => {
11     e.preventDefault();
12
13     try {
14       setLoading(true);
15       setError(null);
16       setSuccess(false);
17
18       const response = await fetch(
19         'https://jsonplaceholder.typicode.com/posts',
20         {
21           method: 'POST',
22           headers: {
23             'Content-Type': 'application/json',
24           },
25           body: JSON.stringify({
26             title,
27             body,
28             userId: 1
29           })
30         }
31       );
32
33       if (!response.ok) {
34         throw new Error('Failed to create post');
35       }
36
37       const data = await response.json();
38       console.log('Created post:', data);
39
40       setSuccess(true);
41       setTitle('');
42       setBody('');
43     } catch (error) {

```

```

44     setError(error.message);
45   } finally {
46     setLoading(false);
47   }
48 };
49
50 return (
51   <form onSubmit={handleSubmit}>
52     <h2>Crea Post</h2>
53
54     {error && <div className="error">{error}</div>}
55     {success && <div className="success">Post creato con successo!</div>}
56
57     <input
58       type="text"
59       value={title}
60       onChange={(e) => setTitle(e.target.value)}
61       placeholder="Titolo"
62       required
63     />
64
65     <textarea
66       value={body}
67       onChange={(e) => setBody(e.target.value)}
68       placeholder="Contenuto"
69       required
70     />
71
72     <button type="submit" disabled={loading}>
73       {loading ? 'Creazione...' : 'Crea Post'}
74     </button>
75   </form>
76 );
77 }

```

### 11.2.4 PUT e DELETE Requests

Listing 11.4: Update e Delete con Fetch

```

1  import { useState } from 'react';
2
3  function PostManager({ postId }) {
4    const [loading, setLoading] = useState(false);
5
6    // UPDATE
7    const updatePost = async (updates) => {
8      try {
9        setLoading(true);
10
11        const response = await fetch(
12          'https://jsonplaceholder.typicode.com/posts/${postId}',
13          {
14            method: 'PUT',
15            headers: {
16              'Content-Type': 'application/json',
17            },

```

```

18         body: JSON.stringify(updates)
19     }
20 );
21
22     if (!response.ok) {
23         throw new Error('Failed to update post');
24     }
25
26     const data = await response.json();
27     console.log('Updated post:', data);
28 } catch (error) {
29     console.error('Update error:', error);
30 } finally {
31     setLoading(false);
32 }
33 };
34
35 // PATCH (partial update)
36 const patchPost = async (updates) => {
37     try {
38         setLoading(true);
39
40         const response = await fetch(
41             'https://jsonplaceholder.typicode.com/posts/${postId}',
42             {
43                 method: 'PATCH',
44                 headers: {
45                     'Content-Type': 'application/json',
46                 },
47                 body: JSON.stringify(updates)
48             }
49         );
50
51         const data = await response.json();
52         console.log('Patched post:', data);
53     } catch (error) {
54         console.error('Patch error:', error);
55     } finally {
56         setLoading(false);
57     }
58 };
59
60 // DELETE
61 const deletePost = async () => {
62     if (!confirm('Sei sicuro di voler eliminare questo post?')) {
63         return;
64     }
65
66     try {
67         setLoading(true);
68
69         const response = await fetch(
70             'https://jsonplaceholder.typicode.com/posts/${postId}',
71             {
72                 method: 'DELETE',
73             }
74         );
75

```

```

76     if (!response.ok) {
77         throw new Error('Failed to delete post');
78     }
79
80     console.log('Post deleted');
81 } catch (error) {
82     console.error('Delete error:', error);
83 } finally {
84     setLoading(false);
85 }
86 };
87
88 return (
89     <div>
90         <button onClick={() => updatePost({ title: 'Nuovo titolo' })}>
91             Update
92         </button>
93         <button onClick={() => patchPost({ title: 'Titolo modificato' })}>
94             Patch
95         </button>
96         <button onClick={deletePost} disabled={loading}>
97             Delete
98         </button>
99     </div>
100 );
101 }

```

### 11.2.5 Gestione Autenticazione con Headers

Listing 11.5: Richieste autenticate con token

```

1  import { useState, useEffect } from 'react';
2
3  function ProtectedData() {
4      const [data, setData] = useState(null);
5      const [loading, setLoading] = useState(true);
6      const [error, setError] = useState(null);
7
8      useEffect(() => {
9          const fetchProtectedData = async () => {
10             try {
11                 // Recupera token dal localStorage
12                 const token = localStorage.getItem('authToken');
13
14                 if (!token) {
15                     throw new Error('No authentication token found');
16                 }
17
18                 const response = await fetch('https://api.example.com/protected',
19                     {
20                         headers: {
21                             'Authorization': 'Bearer ${token}',
22                             'Content-Type': 'application/json'
23                         }
24                     });
25
26                 if (response.status === 401) {

```



```

26         // Token scaduto o invalido
27         throw new Error('Unauthorized - Please login again');
28     }
29
30     if (!response.ok) {
31         throw new Error('HTTP error! status: ${response.status}');
32     }
33
34     const data = await response.json();
35     setData(data);
36   } catch (error) {
37     setError(error.message);
38   } finally {
39     setLoading(false);
40   }
41 };
42
43 fetchProtectedData();
44 }, []);
45
46 if (loading) return <div>Caricamento...</div>;
47 if (error) return <div>Errore: {error}</div>;
48
49 return <div>{JSON.stringify(data, null, 2)}</div>;
50 }

```

### 11.2.6 AbortController per Cancellazione

Listing 11.6: Cancellazione richieste con AbortController

```

1  import { useState, useEffect } from 'react';
2
3  function SearchResults({ searchTerm }) {
4    const [results, setResults] = useState([]);
5    const [loading, setLoading] = useState(false);
6    const [error, setError] = useState(null);
7
8    useEffect(() => {
9      if (!searchTerm) {
10        setResults([]);
11        return;
12      }
13
14      // Crea AbortController
15      const controller = new AbortController();
16      const signal = controller.signal;
17
18      const fetchResults = async () => {
19        try {
20          setLoading(true);
21          setError(null);
22
23          const response = await fetch(
24            'https://api.example.com/search?q=${searchTerm}',
25            { signal } // Passa signal alla fetch
26          );
27

```

```

28     if (!response.ok) {
29       throw new Error('Search failed');
30     }
31
32     const data = await response.json();
33
34     // Controlla se non abortito
35     if (!signal.aborted) {
36       setResults(data);
37     }
38   } catch (error) {
39     // Ignora errori di abort
40     if (error.name !== 'AbortError') {
41       setError(error.message);
42     }
43   } finally {
44     if (!signal.aborted) {
45       setLoading(false);
46     }
47   }
48 };
49
50 fetchResults();
51
52 // Cleanup: abort fetch quando searchTerm cambia
53 return () => {
54   controller.abort();
55 };
56 }, [searchTerm]);
57
58 if (loading) return <div>Ricerca in corso...</div>;
59 if (error) return <div>Errore: {error}</div>;
60
61 return (
62   <ul>
63     {results.map(result => (
64       <li key={result.id}>{result.name}</li>
65     ))}
66   </ul>
67 );
68 }

```

### 11.3 Custom Hook useFetch

Creare un custom hook riutilizzabile per fetch.

Listing 11.7: Custom hook useFetch

```

1 // hooks/useFetch.js
2 import { useState, useEffect } from 'react';
3
4 function useFetch(url, options = {}) {
5   const [data, setData] = useState(null);
6   const [loading, setLoading] = useState(true);
7   const [error, setError] = useState(null);
8
9   useEffect(() => {
10     const controller = new AbortController();

```

```

11
12     const fetchData = async () => {
13         try {
14             setLoading(true);
15             setError(null);
16
17             const response = await fetch(url, {
18                 ...options,
19                 signal: controller.signal
20             });
21
22             if (!response.ok) {
23                 throw new Error('HTTP error! status: ${response.status}');
24             }
25
26             const json = await response.json();
27             setData(json);
28         } catch (error) {
29             if (error.name !== 'AbortError') {
30                 setError(error.message);
31             }
32         } finally {
33             if (!controller.signal.aborted) {
34                 setLoading(false);
35             }
36         }
37     };
38
39     fetchData();
40
41     return () => {
42         controller.abort();
43     };
44 }, [url]); // Dipende solo da url
45
46 return { data, loading, error };
47 }
48
49 export default useFetch;
50
51 // Utilizzo
52 function Users() {
53     const { data: users, loading, error } = useFetch(
54         'https://jsonplaceholder.typicode.com/users'
55     );
56
57     if (loading) return <div>Loading...</div>;
58     if (error) return <div>Error: {error}</div>;
59
60     return (
61         <ul>
62             {users.map(user => (
63                 <li key={user.id}>{user.name}</li>
64             ))}
65         </ul>
66     );
67 }

```

## 11.4 Axios

Axios è una libreria HTTP più feature-rich di Fetch.

### 11.4.1 Installazione

Listing 11.8: Installazione Axios

```
1 npm install axios
2
3 # o con yarn
4 yarn add axios
```

### 11.4.2 GET Request con Axios

Listing 11.9: Axios GET request

```
1 import { useState, useEffect } from 'react';
2 import axios from 'axios';
3
4 function Users() {
5   const [users, setUsers] = useState([]);
6   const [loading, setLoading] = useState(true);
7   const [error, setError] = useState(null);
8
9   useEffect(() => {
10     const fetchUsers = async () => {
11       try {
12         setLoading(true);
13
14         // Axios gestisce automaticamente JSON
15         const response = await axios.get(
16           'https://jsonplaceholder.typicode.com/users'
17         );
18
19         // I dati sono direttamente in response.data
20         setUsers(response.data);
21       } catch (error) {
22         // Axios fornisce error.response per errori HTTP
23         if (error.response) {
24           setError(`Error: ${error.response.status}`);
25         } else if (error.request) {
26           setError('No response from server');
27         } else {
28           setError(error.message);
29         }
30       } finally {
31         setLoading(false);
32       }
33     };
34
35     fetchUsers();
36   }, []);
37
38   if (loading) return <div>Loading...</div>;
39   if (error) return <div>{error}</div>;
40 }
```

```
41     return (  
42         <ul>  
43             {users.map(user => (  
44                 <li key={user.id}>{user.name}</li>  
45             ))}  
46         </ul>  
47     );  
48 }
```

### 11.4.3 POST, PUT, DELETE con Axios

Listing 11.10: CRUD operations con Axios

```
1  import axios from 'axios';  
2  
3  const API_URL = 'https://jsonplaceholder.typicode.com/posts';  
4  
5  // CREATE  
6  async function createPost(postData) {  
7      try {  
8          const response = await axios.post(API_URL, postData);  
9          return { success: true, data: response.data };  
10     } catch (error) {  
11         return { success: false, error: error.message };  
12     }  
13 }  
14  
15 // READ  
16 async function getPost(id) {  
17     try {  
18         const response = await axios.get(`${API_URL}/${id}`);  
19         return { success: true, data: response.data };  
20     } catch (error) {  
21         return { success: false, error: error.message };  
22     }  
23 }  
24  
25 // UPDATE (full)  
26 async function updatePost(id, postData) {  
27     try {  
28         const response = await axios.put(`${API_URL}/${id}`, postData);  
29         return { success: true, data: response.data };  
30     } catch (error) {  
31         return { success: false, error: error.message };  
32     }  
33 }  
34  
35 // UPDATE (partial)  
36 async function patchPost(id, updates) {  
37     try {  
38         const response = await axios.patch(`${API_URL}/${id}`, updates);  
39         return { success: true, data: response.data };  
40     } catch (error) {  
41         return { success: false, error: error.message };  
42     }  
43 }  
44 }
```

```

45 // DELETE
46 async function deletePost(id) {
47   try {
48     await axios.delete(`${API_URL}/${id}`);
49     return { success: true };
50   } catch (error) {
51     return { success: false, error: error.message };
52   }
53 }
54
55 // Utilizzo in componente
56 function PostEditor({ postId }) {
57   const handleUpdate = async () => {
58     const result = await updatePost(postId, {
59       title: 'Titolo aggiornato',
60       body: 'Corpo aggiornato'
61     });
62
63     if (result.success) {
64       console.log('Post aggiornato:', result.data);
65     } else {
66       console.error('Errore:', result.error);
67     }
68   };
69
70   const handleDelete = async () => {
71     const result = await deletePost(postId);
72
73     if (result.success) {
74       console.log('Post eliminato');
75     } else {
76       console.error('Errore:', result.error);
77     }
78   };
79
80   return (
81     <div>
82       <button onClick={handleUpdate}>Aggiorna</button>
83       <button onClick={handleDelete}>Elimina</button>
84     </div>
85   );
86 }

```

#### 11.4.4 Axios Instance e Interceptors

Listing 11.11: Configurazione Axios instance

```

1 // api/axios.js
2 import axios from 'axios';
3
4 // Crea instance personalizzata
5 const axiosInstance = axios.create({
6   baseURL: 'https://api.example.com',
7   timeout: 10000,
8   headers: {
9     'Content-Type': 'application/json'
10  }

```

```
11 });
12
13 // Request interceptor
14 axiosInstance.interceptors.request.use(
15   (config) => {
16     // Aggiungi token ad ogni richiesta
17     const token = localStorage.getItem('authToken');
18
19     if (token) {
20       config.headers.Authorization = 'Bearer ${token}';
21     }
22
23     console.log('Request:', config.method.toUpperCase(), config.url);
24     return config;
25   },
26   (error) => {
27     return Promise.reject(error);
28   }
29 );
30
31 // Response interceptor
32 axiosInstance.interceptors.response.use(
33   (response) => {
34     console.log('Response:', response.status, response.config.url);
35     return response;
36   },
37   (error) => {
38     // Gestione errori globale
39     if (error.response) {
40       switch (error.response.status) {
41         case 401:
42           // Unauthorized - redirect al login
43           localStorage.removeItem('authToken');
44           window.location.href = '/login';
45           break;
46
47         case 403:
48           // Forbidden
49           console.error('Access forbidden');
50           break;
51
52         case 404:
53           console.error('Resource not found');
54           break;
55
56         case 500:
57           console.error('Server error');
58           break;
59
60         default:
61           console.error('Request error:', error.response.status);
62       }
63     } else if (error.request) {
64       console.error('No response from server');
65     } else {
66       console.error('Error:', error.message);
67     }
68   }
```

```
69     return Promise.reject(error);
70   }
71 );
72
73 export default axiosInstance;
74
75 // Utilizzo
76 import api from './api/axios';
77
78 function MyComponent() {
79   const fetchData = async () => {
80     try {
81       // L'interceptor aggiunge automaticamente il token
82       const response = await api.get('/protected-route');
83       console.log(response.data);
84     } catch (error) {
85       // L'error interceptor ha già loggato l'errore
86       console.error('Failed to fetch data');
87     }
88   };
89
90   return <button onClick={fetchData}>Fetch Data</button>;
91 }
```

### 11.4.5 Cancellazione con Axios

Listing 11.12: Cancel token in Axios

```
1 import { useState, useEffect } from 'react';
2 import axios from 'axios';
3
4 function SearchComponent({ query }) {
5   const [results, setResults] = useState([]);
6   const [loading, setLoading] = useState(false);
7
8   useEffect(() => {
9     // Crea cancel token
10    const source = axios.CancelToken.source();
11
12    const fetchResults = async () => {
13      try {
14        setLoading(true);
15
16        const response = await axios.get(
17          'https://api.example.com/search?q=${query}',
18          {
19            cancelToken: source.token
20          }
21        );
22
23        setResults(response.data);
24      } catch (error) {
25        if (axios.isCancel(error)) {
26          console.log('Request canceled:', error.message);
27        } else {
28          console.error('Error:', error);
29        }
30      }
31    };
32  });
33 }
```



```

30     } finally {
31       setLoading(false);
32     }
33   };
34
35   if (query) {
36     fetchResults();
37   }
38
39   // Cleanup: cancella richiesta
40   return () => {
41     source.cancel('Operation canceled by user');
42   };
43 }, [query]);
44
45 return (
46   <div>
47     {loading && <div>Searching...</div>}
48     {results.map(result => (
49       <div key={result.id}>{result.name}</div>
50     ))}
51   </div>
52 );
53 }

```

## 11.5 React Query (TanStack Query)

React Query è la soluzione moderna per data fetching, caching e sincronizzazione.

### 11.5.1 Installazione

Listing 11.13: Installazione React Query

```

1 npm install @tanstack/react-query
2
3 # o con yarn
4 yarn add @tanstack/react-query

```

### 11.5.2 Setup Base

Listing 11.14: QueryClient setup

```

1 // main.jsx
2 import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
3 ;
4
5 import { ReactQueryDevtools } from '@tanstack/react-query-devtools';
6
7 // Crea query client
8 const queryClient = new QueryClient({
9   defaultOptions: {
10     queries: {
11       staleTime: 1000 * 60 * 5, // 5 minuti
12       cacheTime: 1000 * 60 * 10, // 10 minuti
13       refetchOnWindowFocus: false,
14       retry: 1
15     }
16   }
17 });

```

```
13     }
14   }
15 });
16
17 ReactDOM.createRoot(document.getElementById('root')).render(
18   <QueryClientProvider client={queryClient}>
19     <App />
20     <ReactQueryDevtools initialIsOpen={false} />
21   </QueryClientProvider>
22 );
```

### 11.5.3 useQuery Hook

Listing 11.15: Fetch data con useQuery

```
1 import { useQuery } from '@tanstack/react-query';
2 import axios from 'axios';
3
4 // Funzione di fetch
5 const fetchUsers = async () => {
6   const { data } = await axios.get(
7     'https://jsonplaceholder.typicode.com/users'
8   );
9   return data;
10 };
11
12 function Users() {
13   const {
14     data: users,
15     isLoading,
16     isError,
17     error,
18     refetch,
19     isFetching
20   } = useQuery({
21     queryKey: ['users'],
22     queryFn: fetchUsers
23   });
24
25   if (isLoading) return <div>Caricamento iniziale...</div>;
26   if (isError) return <div>Errore: {error.message}</div>;
27
28   return (
29     <div>
30       <button onClick={() => refetch()}>
31         {isFetching ? 'Aggiornamento...' : 'Aggiorna'}
32       </button>
33
34       <ul>
35         {users.map(user => (
36           <li key={user.id}>{user.name}</li>
37         ))}
38       </ul>
39     </div>
40   );
41 }
```

### 11.5.4 Query con Parametri

Listing 11.16: useQuery con parametri dinamici

```
1 import { useQuery } from '@tanstack/react-query';
2 import axios from 'axios';
3
4 const fetchUser = async (userId) => {
5   const { data } = await axios.get(
6     'https://jsonplaceholder.typicode.com/users/${userId}'
7   );
8   return data;
9 };
10
11 function UserProfile({ userId }) {
12   const {
13     data: user,
14     isLoading,
15     isError,
16     error
17   } = useQuery({
18     queryKey: ['user', userId], // Include userId nella key
19     queryFn: () => fetchUser(userId),
20     enabled: !!userId // Esegue solo se userId esiste
21   });
22
23   if (!userId) return <div>Seleziona un utente</div>;
24   if (isLoading) return <div>Caricamento...</div>;
25   if (isError) return <div>Errore: {error.message}</div>;
26
27   return (
28     <div>
29       <h2>{user.name}</h2>
30       <p>Email: {user.email}</p>
31       <p>Website: {user.website}</p>
32     </div>
33   );
34 }
```

### 11.5.5 useMutation Hook

Listing 11.17: Mutations per CREATE, UPDATE, DELETE

```
1 import { useMutation, useQueryClient } from '@tanstack/react-query';
2 import axios from 'axios';
3
4 const createPost = async (newPost) => {
5   const { data } = await axios.post(
6     'https://jsonplaceholder.typicode.com/posts',
7     newPost
8   );
9   return data;
10 };
11
12 function CreatePost() {
13   const queryClient = useQueryClient();
14
15   const mutation = useMutation({
```

```

16     mutationFn: createPost,
17     onSuccess: (data) => {
18         // Invalida e refetch
19         queryClient.invalidateQueries({ queryKey: ['posts'] });
20
21         // Oppure aggiungi manualmente alla cache
22         // queryClient.setQueryData(['posts'], (old) => [...old, data]);
23
24         console.log('Post creato:', data);
25     },
26     onError: (error) => {
27         console.error('Errore creazione:', error);
28     }
29 });
30
31 const handleSubmit = (e) => {
32     e.preventDefault();
33
34     mutation.mutate({
35         title: 'Nuovo Post',
36         body: 'Contenuto del post',
37         userId: 1
38     });
39 };
40
41 return (
42     <form onSubmit={handleSubmit}>
43         <button type="submit" disabled={mutation.isPending}>
44             {mutation.isPending ? 'Creazione...' : 'Crea Post'}
45         </button>
46
47         {mutation.isError && (
48             <div>Errore: {mutation.error.message}</div>
49         )}
50
51         {mutation.isSuccess && (
52             <div>Post creato con successo!</div>
53         )}
54     </form>
55 );
56 }

```

### 11.5.6 Esempio Completo: CRUD con React Query

Listing 11.18: CRUD completo con React Query

```

1  import { useQuery, useMutation, useQueryClient } from '@tanstack/react-
   query';
2  import axios from 'axios';
3  import { useState } from 'react';
4
5  const API_URL = 'https://jsonplaceholder.typicode.com/posts';
6
7  // API functions
8  const fetchPosts = async () => {
9      const { data } = await axios.get(API_URL);
10     return data;

```

```

11 };
12
13 const createPost = async (newPost) => {
14   const { data } = await axios.post(API_URL, newPost);
15   return data;
16 };
17
18 const updatePost = async ({ id, updates }) => {
19   const { data } = await axios.put(`${API_URL}/${id}`, updates);
20   return data;
21 };
22
23 const deletePost = async (id) => {
24   await axios.delete(`${API_URL}/${id}`);
25   return id;
26 };
27
28 function PostsManager() {
29   const queryClient = useQueryClient();
30   const [editingId, setEditingId] = useState(null);
31
32   // GET - Fetch all posts
33   const {
34     data: posts,
35     isLoading,
36     isError,
37     error
38   } = useQuery({
39     queryKey: ['posts'],
40     queryFn: fetchPosts
41   });
42
43   // CREATE
44   const createMutation = useMutation({
45     mutationFn: createPost,
46     onSuccess: () => {
47       queryClient.invalidateQueries({ queryKey: ['posts'] });
48     }
49   });
50
51   // UPDATE
52   const updateMutation = useMutation({
53     mutationFn: updatePost,
54     onSuccess: () => {
55       queryClient.invalidateQueries({ queryKey: ['posts'] });
56       setEditingId(null);
57     }
58   });
59
60   // DELETE
61   const deleteMutation = useMutation({
62     mutationFn: deletePost,
63     onSuccess: () => {
64       queryClient.invalidateQueries({ queryKey: ['posts'] });
65     }
66   });
67
68   const handleCreate = () => {

```

```
69     createMutation.mutate({
70       title: 'Nuovo Post',
71       body: 'Contenuto',
72       userId: 1
73     });
74   };
75
76   const handleUpdate = (id) => {
77     updateMutation.mutate({
78       id,
79       updates: {
80         title: 'Titolo Aggiornato',
81         body: 'Corpo Aggiornato'
82       }
83     });
84   };
85
86   const handleDelete = (id) => {
87     if (confirm('Eliminare questo post?')) {
88       deleteMutation.mutate(id);
89     }
90   };
91
92   if (isLoading) return <div>Caricamento...</div>;
93   if (isError) return <div>Errore: {error.message}</div>;
94
95   return (
96     <div>
97       <h1>Posts Manager</h1>
98
99       <button onClick={handleCreate} disabled={createMutation.isPending}
100         {createMutation.isPending ? 'Creazione...' : 'Crea Nuovo Post'}>
101         </button>
102
103       <div className="posts">
104         {posts.map(post => (
105           <div key={post.id} className="post">
106             <h3>{post.title}</h3>
107             <p>{post.body}</p>
108
109             <button
110               onClick={() => handleUpdate(post.id)}
111               disabled={updateMutation.isPending}>
112               >
113               Update
114             </button>
115
116             <button
117               onClick={() => handleDelete(post.id)}
118               disabled={deleteMutation.isPending}>
119               >
120               Delete
121             </button>
122           </div>
123         ))}
124       </div>
125     </div>
```

```

126   );
127 }

```

### 11.5.7 Infinite Queries

Listing 11.19: Infinite scrolling con useInfiniteQuery

```

1  import { useInfiniteQuery } from '@tanstack/react-query';
2  import axios from 'axios';
3  import { useEffect, useRef } from 'react';
4
5  const fetchPosts = async ({ pageParam = 1 }) => {
6    const { data } = await axios.get(
7      `https://jsonplaceholder.typicode.com/posts?_page=${pageParam}&
       _limit=10`
8    );
9    return {
10     posts: data,
11     nextPage: pageParam + 1,
12     hasMore: data.length > 0
13   };
14 };
15
16 function InfinitePostList() {
17   const {
18     data,
19     fetchNextPage,
20     hasNextPage,
21     isFetchingNextPage,
22     isLoading,
23     isError,
24     error
25   } = useInfiniteQuery({
26     queryKey: ['posts', 'infinite'],
27     queryFn: fetchPosts,
28     getNextPageParam: (lastPage) =>
29       lastPage.hasMore ? lastPage.nextPage : undefined,
30     initialPageParam: 1
31   });
32
33   const observerTarget = useRef(null);
34
35   useEffect(() => {
36     const observer = new IntersectionObserver(
37       entries => {
38         if (entries[0].isIntersecting && hasNextPage) {
39           fetchNextPage();
40         }
41       },
42       { threshold: 1 }
43     );
44
45     if (observerTarget.current) {
46       observer.observe(observerTarget.current);
47     }
48
49     return () => observer.disconnect();

```

```

50   }, [hasNextPage, fetchNextPage]);
51
52   if (isLoading) return <div>Caricamento...</div>;
53   if (isError) return <div>Errore: {error.message}</div>;
54
55   return (
56     <div>
57       <h1>Infinite Scroll Posts</h1>
58
59       {data.pages.map((page, pageIndex) => (
60         <div key={pageIndex}>
61           {page.posts.map(post => (
62             <article key={post.id} className="post">
63               <h3>{post.title}</h3>
64               <p>{post.body}</p>
65             </article>
66           ))}
67         </div>
68       ))}
69
70       <div ref={observerTarget} className="observer">
71         {isFetchingNextPage && <div>Caricamento altri post...</div>}
72         {!hasNextPage && <div>Fine dei post</div>}
73       </div>
74     </div>
75   );
76 }

```

### 11.5.8 Optimistic Updates

Listing 11.20: Aggiornamenti ottimistici

```

1  import { useMutation, useQueryClient } from '@tanstack/react-query';
2  import axios from 'axios';
3
4  function TodoList() {
5    const queryClient = useQueryClient();
6
7    const toggleTodoMutation = useMutation({
8      mutationFn: async ({ id, completed }) => {
9        const { data } = await axios.patch(
10          'https://api.example.com/todos/${id}',
11          { completed: !completed }
12        );
13        return data;
14      },
15
16      // Optimistic update
17      onMutate: async ({ id, completed }) => {
18        // Cancella queries in uscita
19        await queryClient.cancelQueries({ queryKey: ['todos'] });
20
21        // Snapshot del valore precedente
22        const previousTodos = queryClient.getQueryData(['todos']);
23
24        // Aggiorna ottimisticamente
25        queryClient.setQueryData(['todos'], (old) =>

```



```

26         old.map(todo =>
27             todo.id === id
28             ? { ...todo, completed: !completed }
29             : todo
30         )
31     );
32
33     // Ritorna context con snapshot
34     return { previousTodos };
35 },
36
37     // Se la mutation fallisce, rollback
38     onError: (err, variables, context) => {
39         queryClient.setQueryData(['todos'], context.previousTodos);
40     },
41
42     // Sempre refetch dopo error o success
43     onSettled: () => {
44         queryClient.invalidateQueries({ queryKey: ['todos'] });
45     }
46 });
47
48 return (
49     // JSX component
50     <div>Todos</div>
51 );
52 }

```

## 11.6 Error Handling Avanzato

### 11.6.1 Error Boundary per API Errors

Listing 11.21: Error boundary per gestione errori

```

1  import { Component } from 'react';
2
3  class ApiErrorBoundary extends Component {
4      constructor(props) {
5          super(props);
6          this.state = { hasError: false, error: null };
7      }
8
9      static getDerivedStateFromError(error) {
10         return { hasError: true, error };
11     }
12
13     componentDidCatch(error, errorInfo) {
14         console.error('API Error:', error, errorInfo);
15
16         // Log to error reporting service
17         // logErrorToService(error, errorInfo);
18     }
19
20     render() {
21         if (this.state.hasError) {
22             return (
23                 <div className="error-container">

```

```

24         <h2>Qualcosa è andato storto</h2>
25         <p>{this.state.error?.message}</p>
26         <button onClick={() => this.setState({ hasError: false })}>
27             Riprova
28         </button>
29     </div>
30     );
31 }
32
33     return this.props.children;
34 }
35 }
36
37 // Utilizzo
38 function App() {
39     return (
40         <ApiErrorBoundary>
41             <DataFetchingComponent />
42         </ApiErrorBoundary>
43     );
44 }

```

### 11.6.2 Toast Notifications per Errori

Listing 11.22: Sistema di notifiche per errori API

```

1  import { createContext, useContext, useState } from 'react';
2
3  const ToastContext = createContext(null);
4
5  export function ToastProvider({ children }) {
6      const [toasts, setToasts] = useState([]);
7
8      const addToast = (message, type = 'info') => {
9          const id = Date.now();
10         setToasts(prev => [...prev, { id, message, type }]);
11
12         // Auto-remove dopo 5 secondi
13         setTimeout(() => {
14             removeToast(id);
15         }, 5000);
16     };
17
18     const removeToast = (id) => {
19         setToasts(prev => prev.filter(toast => toast.id !== id));
20     };
21
22     return (
23         <ToastContext.Provider value={{ addToast, removeToast }}>
24             {children}
25
26             <div className="toast-container">
27                 {toasts.map(toast => (
28                     <div key={toast.id} className={`toast toast-${toast.type}`}>
29                         {toast.message}
30                         <button onClick={() => removeToast(toast.id)}> </button>
31                     </div>

```

```

32     ))}
33     </div>
34     </ToastContext.Provider>
35   );
36 }
37
38 export function useToast() {
39   return useContext(ToastContext);
40 }
41
42 // Utilizzo con API
43 function MyComponent() {
44   const { addToast } = useToast();
45
46   const fetchData = async () => {
47     try {
48       const response = await fetch('https://api.example.com/data');
49
50       if (!response.ok) {
51         throw new Error('Failed to fetch data');
52       }
53
54       const data = await response.json();
55       addToast('Dati caricati con successo!', 'success');
56     } catch (error) {
57       addToast(error.message, 'error');
58     }
59   };
60
61   return <button onClick={fetchData}>Fetch Data</button>;
62 }

```

### 11.6.3 Retry Logic

Listing 11.23: Logica di retry personalizzata

```

1  import { useState } from 'react';
2
3  async function fetchWithRetry(url, options = {}, maxRetries = 3) {
4    let lastError;
5
6    for (let i = 0; i <= maxRetries; i++) {
7      try {
8        const response = await fetch(url, options);
9
10       if (!response.ok) {
11         throw new Error('HTTP error! status: ${response.status}');
12       }
13
14       return await response.json();
15     } catch (error) {
16       lastError = error;
17
18       // Non ritentare se è l'ultimo tentativo
19       if (i === maxRetries) {
20         break;
21       }

```

```

22
23 // Exponential backoff: attendi 1s, 2s, 4s, etc.
24 const delay = Math.pow(2, i) * 1000;
25 console.log('Retry ${i + 1}/${maxRetries} after ${delay}ms');
26
27 await new Promise(resolve => setTimeout(resolve, delay));
28 }
29 }
30
31 throw lastError;
32 }
33
34 // Utilizzo
35 function DataComponent() {
36   const [data, setData] = useState(null);
37   const [loading, setLoading] = useState(false);
38   const [error, setError] = useState(null);
39
40   const loadData = async () => {
41     try {
42       setLoading(true);
43       setError(null);
44
45       const result = await fetchWithRetry(
46         'https://api.example.com/data',
47         {},
48         3 // max 3 tentativi
49       );
50
51       setData(result);
52     } catch (error) {
53       setError('Failed after retries: ${error.message}');
54     } finally {
55       setLoading(false);
56     }
57   };
58
59   return (
60     <div>
61       <button onClick={loadData} disabled={loading}>
62         {loading ? 'Loading...' : 'Load Data'}
63       </button>
64       {error && <div className="error">{error}</div>}
65       {data && <pre>{JSON.stringify(data, null, 2)}</pre>}
66     </div>
67   );
68 }

```

## 11.7 Loading States Avanzati

### 11.7.1 Skeleton Screens

Listing 11.24: Skeleton loading component

```

1 function Skeleton({ width, height, className = '' }) {
2   return (
3     <div

```

```

4      className={`skeleton ${className}`}
5      style={{
6        width,
7        height,
8        background: `linear-gradient(90deg, #f0f0f0 25%, #e0e0e0 50%, #
          f0f0f0 75%)`,
9        backgroundSize: `200% 100%`,
10       animation: `loading 1.5s ease-in-out infinite`,
11       borderRadius: `4px`
12     }}
13   />
14 );
15 }
16
17 // CSS
18 // @keyframes loading {
19 //   0% {
20 //     background-position: 200% 0;
21 //   }
22 //   100% {
23 //     background-position: -200% 0;
24 //   }
25 // }
26
27 function UserCardSkeleton() {
28   return (
29     <div className="user-card">
30       <Skeleton width={64} height={64} className="avatar" />
31       <div>
32         <Skeleton width={200} height={20} />
33         <Skeleton width={150} height={16} />
34       </div>
35     </div>
36   );
37 }
38
39 // Utilizzo
40 function UserList() {
41   const { data: users, isLoading } = useQuery({
42     queryKey: ['users'],
43     queryFn: fetchUsers
44   });
45
46   if (isLoading) {
47     return (
48       <div>
49         {[...Array(5)].map((_, i) => (
50           <UserCardSkeleton key={i} />
51         ))}
52       </div>
53     );
54   }
55
56   return (
57     <div>
58       {users.map(user => (
59         <UserCard key={user.id} user={user} />
60       ))}

```

```

61     </div>
62   );
63 }

```

### 11.7.2 Progress Indicator

Listing 11.25: Progress bar per upload

```

1  import { useState } from 'react';
2  import axios from 'axios';
3
4  function FileUpload() {
5    const [file, setFile] = useState(null);
6    const [progress, setProgress] = useState(0);
7    const [uploading, setUploading] = useState(false);
8
9    const handleUpload = async () => {
10      if (!file) return;
11
12      const formData = new FormData();
13      formData.append('file', file);
14
15      try {
16        setUploading(true);
17
18        await axios.post('https://api.example.com/upload', formData, {
19          headers: {
20            'Content-Type': 'multipart/form-data'
21          },
22          onUploadProgress: (progressEvent) => {
23            const percentCompleted = Math.round(
24              (progressEvent.loaded * 100) / progressEvent.total
25            );
26            setProgress(percentCompleted);
27          }
28        });
29
30        console.log('Upload completato!');
31      } catch (error) {
32        console.error('Upload fallito:', error);
33      } finally {
34        setUploading(false);
35        setProgress(0);
36      }
37    };
38
39    return (
40      <div>
41        <input
42          type="file"
43          onChange={(e) => setFile(e.target.files[0])}
44        />
45
46        <button onClick={handleUpload} disabled={!file || uploading}>
47          {uploading ? 'Uploading...' : 'Upload'}
48        </button>
49      </div>

```

```
50     {uploading && (  
51       <div className="progress-bar">  
52         <div  
53           className="progress-fill"  
54           style={{ width: `${progress}%` }}  
55         >  
56           {progress}%  
57         </div>  
58       </div>  
59     )}  
60   </div>  
61 );  
62 }
```

## 11.8 Caching e Revalidation

### 11.8.1 React Query Cache Config

Listing 11.26: Configurazione cache avanzata

```
1 import { QueryClient } from '@tanstack/react-query';  
2  
3 const queryClient = new QueryClient({  
4   defaultOptions: {  
5     queries: {  
6       // Dati considerati fresh per 5 minuti  
7       staleTime: 1000 * 60 * 5,  
8  
9       // Dati mantenuti in cache per 10 minuti  
10      cacheTime: 1000 * 60 * 10,  
11  
12      // Refetch quando la finestra riceve focus  
13      refetchOnWindowFocus: true,  
14  
15      // Refetch quando si riconnette  
16      refetchOnReconnect: true,  
17  
18      // Refetch on mount se dati stale  
19      refetchOnMount: true,  
20  
21      // Retry fallite requests  
22      retry: 2,  
23  
24      // Retry delay (exponential backoff)  
25      retryDelay: (attemptIndex) =>  
26        Math.min(1000 * 2 ** attemptIndex, 30000)  
27    },  
28    mutations: {  
29      // Retry mutations  
30      retry: 1  
31    }  
32  }  
33 });
```

### 11.8.2 Manual Cache Updates

Listing 11.27: Aggiornamento manuale della cache

```
1 import { useQueryClient } from '@tanstack/react-query';
2
3 function UserProfile({ userId }) {
4   const queryClient = useQueryClient();
5
6   const updateUserName = (newName) => {
7     // Update cache direttamente
8     queryClient.setQueryData(['user', userId], (oldData) => ({
9       ...oldData,
10      name: newName
11    }));
12   };
13
14   const invalidateUser = () => {
15     // Invalida cache (forza refetch)
16     queryClient.invalidateQueries({ queryKey: ['user', userId] });
17   };
18
19   const prefetchUser = async (nextUserId) => {
20     // Prefetch dati per navigazione anticipata
21     await queryClient.prefetchQuery({
22       queryKey: ['user', nextUserId],
23       queryFn: () => fetchUser(nextUserId)
24     });
25   };
26
27   return <div>User Profile</div>;
28 }
```

## 11.9 Best Practices

### API Integration Best Practices

- Usa React Query per data fetching complesso
- Implementa proper error handling con retry logic
- Mostra loading states significativi (skeleton, spinner)
- Cancella richieste quando componenti unmount
- Usa AbortController o cancel tokens
- Implementa caching per ridurre richieste
- Normalizza dati API per evitare duplicazioni
- Usa interceptors per logica globale (auth, logging)
- Gestisci stati di rete (offline/online)
- Implementa optimistic updates per UX migliore
- Testa error cases e edge cases
- Usa TypeScript per type safety delle risposte API



## 11.10 Conclusioni

L'integrazione con API è fondamentale per applicazioni React moderne. Le opzioni principali sono:

- **Fetch API:** Nativa, leggera, sufficiente per app semplici
- **Axios:** Feature-rich, interceptors, auto-JSON parsing
- **React Query:** Migliore per caching, sincronizzazione, states complessi

Per applicazioni moderne, React Query è la scelta consigliata per la gestione completa di server state, caching e sincronizzazione automatica.



## Capitolo 12

# Ottimizzazione delle Performance

### 12.1 Introduzione

L'ottimizzazione delle performance è cruciale per creare applicazioni React veloci e reattive. Questo capitolo copre tecniche di memoization, code splitting, lazy loading e profiling.

#### 12.1.1 Quando Ottimizzare

##### Regola d'Oro

##### **Ottimizza solo quando necessario!**

Premature optimization è la radice di tutti i mali. Prima misura, poi ottimizza.

- Misura le performance con React DevTools Profiler
- Identifica bottleneck reali
- Ottimizza solo componenti problematici
- Non ottimizzare tutto preventivamente

### 12.2 React.memo

`React.memo` è un Higher Order Component che memoizza i componenti per evitare re-render inutili.

#### 12.2.1 Utilizzo Base

Listing 12.1: React.memo base

```
1 import { memo } from 'react';
2
3 // Componente senza memo (ri-renderizza sempre)
4 function ExpensiveComponent({ data }) {
5   console.log('Rendering ExpensiveComponent');
6
7   // Calcoli costosi...
8   const processedData = data.map(item => {
9     // elaborazione complessa
10    return item * 2;
11  });
12 }
```

```

13   return (
14     <div>
15       {processedData.map((item, i) => (
16         <div key={i}>{item}</div>
17       ))}
18     </div>
19   );
20 }
21
22 // Componente con memo (ri-renderizza solo se props cambiano)
23 const MemoizedExpensiveComponent = memo(ExpensiveComponent);
24
25 // Utilizzo
26 function Parent() {
27   const [count, setCount] = useState(0);
28   const [data, setData] = useState([1, 2, 3, 4, 5]);
29
30   return (
31     <div>
32       <button onClick={() => setCount(count + 1)}>Count: {count}</button>
33
34       {/* Questo NON ri-renderizza quando count cambia */}
35       <MemoizedExpensiveComponent data={data} />
36     </div>
37   );
38 }

```

## 12.2.2 Custom Comparison Function

Listing 12.2: React.memo con comparatore personalizzato

```

1  import { memo } from 'react';
2
3  function UserCard({ user, onEdit }) {
4    console.log('Rendering UserCard for', user.name);
5
6    return (
7      <div className="user-card">
8        <h3>{user.name}</h3>
9        <p>{user.email}</p>
10       <button onClick={() => onEdit(user.id)}>Edit</button>
11     </div>
12   );
13 }
14
15 // Custom comparison: ri-renderizza solo se user.id o user.name cambiano
16 const MemoizedUserCard = memo(UserCard, (prevProps, nextProps) => {
17   // Ritorna true se props sono uguali (NON re-render)
18   // Ritorna false se props sono diverse (re-render)
19   return (
20     prevProps.user.id === nextProps.user.id &&
21     prevProps.user.name === nextProps.user.name
22   );
23 });
24
25 export default MemoizedUserCard;

```

### 12.2.3 Quando NON usare React.memo

Listing 12.3: Casi in cui React.memo non aiuta

```

1 import { memo, useState } from 'react';
2
3 // SBAGLIATO: Component che cambia sempre
4 const BadMemo = memo(({ timestamp }) => {
5   // timestamp cambia sempre, memo inutile
6   return <div>Time: {timestamp}</div>;
7 });
8
9 // SBAGLIATO: Props non primitive sempre nuove
10 function Parent() {
11   const [count, setCount] = useState(0);
12
13   return (
14     <div>
15       <button onClick={() => setCount(count + 1)}>Increment</button>
16
17       {/* Oggetto creato ogni render, memo inutile */}
18       <MemoizedChild config={{ theme: 'dark' }} />
19
20       {/* Funzione creata ogni render, memo inutile */}
21       <MemoizedChild onClick={() => console.log('click')} />
22     </div>
23   );
24 }
25
26 // CORRETTO: Usa useCallback e useMemo
27 function ParentFixed() {
28   const [count, setCount] = useState(0);
29
30   const config = useMemo(() => ({ theme: 'dark' }), []);
31   const handleClick = useCallback(() => console.log('click'), []);
32
33   return (
34     <div>
35       <button onClick={() => setCount(count + 1)}>Increment</button>
36       <MemoizedChild config={config} onClick={handleClick} />
37     </div>
38   );
39 }

```

## 12.3 useMemo Hook

useMemo memoizza il risultato di un calcolo costoso.

### 12.3.1 Utilizzo Base

Listing 12.4: useMemo per calcoli costosi

```

1 import { useState, useMemo } from 'react';
2
3 function ProductList({ products }) {
4   const [filter, setFilter] = useState('');
5

```

```

6 // Calcolo costoso memoizzato
7 const filteredProducts = useMemo(() => {
8   console.log('Filtering products...');
9
10   return products.filter(product =>
11     product.name.toLowerCase().includes(filter.toLowerCase())
12   );
13 }, [products, filter]); // Ri-calcola solo se products o filter
    cambiano
14
15 // Calcolo costoso: somma prezzi
16 const totalPrice = useMemo(() => {
17   console.log('Calculating total price...');
18
19   return filteredProducts.reduce((sum, p) => sum + p.price, 0);
20 }, [filteredProducts]);
21
22 return (
23   <div>
24     <input
25       value={filter}
26       onChange={(e) => setFilter(e.target.value)}
27       placeholder="Filtra prodotti..."
28     />
29
30     <p>Totale: {totalPrice.toFixed(2)}    </p>
31
32     <ul>
33       {filteredProducts.map(product => (
34         <li key={product.id}>
35           {product.name} - {product.price}
36         </li>
37       ))}
38     </ul>
39   </div>
40 );
41 }

```

### 12.3.2 Esempi Pratici

Listing 12.5: useMemo per vari scenari

```

1 import { useMemo } from 'react';
2
3 function DataAnalysis({ data }) {
4   // 1. Calcolo statistiche
5   const stats = useMemo(() => {
6     const sum = data.reduce((acc, val) => acc + val, 0);
7     const avg = sum / data.length;
8     const max = Math.max(...data);
9     const min = Math.min(...data);
10
11     return { sum, avg, max, min };
12   }, [data]);
13
14   // 2. Ordinamento
15   const sortedData = useMemo(() => {

```

```

16     return [...data].sort((a, b) => a - b);
17 }, [data]);
18
19 // 3. Trasformazione complessa
20 const processedData = useMemo(() => {
21     return data.map(item => ({
22         value: item,
23         squared: item * item,
24         isEven: item % 2 === 0
25     }));
26 }, [data]);
27
28 // 4. Creazione oggetto config
29 const chartConfig = useMemo(() => ({
30     type: 'line',
31     data: sortedData,
32     options: {
33         responsive: true,
34         plugins: {
35             legend: {
36                 position: 'top'
37             }
38         }
39     })
40 }, [sortedData]);
41
42 return (
43     <div>
44         <h3>Statistiche</h3>
45         <p>Media: {stats.avg.toFixed(2)}</p>
46         <p>Max: {stats.max}</p>
47         <p>Min: {stats.min}</p>
48     </div>
49 );
50 }

```

### 12.3.3 Quando NON usare useMemo

Listing 12.6: useMemo inutile o dannoso

```

1 import { useMemo } from 'react';
2
3 function Component() {
4     // SBAGLIATO: Calcolo semplice, overhead inutile
5     const doubleValue = useMemo(() => value * 2, [value]);
6
7     // CORRETTO: Calcolo diretto
8     const doubleValue = value * 2;
9
10    // SBAGLIATO: String interpolation
11    const greeting = useMemo(() => `Hello, ${name}`, [name]);
12
13    // CORRETTO: Calcolo diretto
14    const greeting = `Hello, ${name}`;
15
16    // SBAGLIATO: Array.map semplice
17    const doubled = useMemo(() => numbers.map(n => n * 2), [numbers]);

```

```
18
19 // CORRETTO: Solo se numbers è MOLTO grande
20 // Altrimenti calcolo diretto
21 const doubled = numbers.map(n => n * 2);
22 }
```

## 12.4 useCallback Hook

useCallback memoizza funzioni per evitare che siano ricreate ad ogni render.

### 12.4.1 Utilizzo Base

Listing 12.7: useCallback con React.memo

```
1 import { useState, useCallback, memo } from 'react';
2
3 // Child component memoizzato
4 const Button = memo(({ onClick, children }) => {
5   console.log('Rendering Button:', children);
6   return <button onClick={onClick}>{children}</button>;
7 });
8
9 function Parent() {
10   const [count, setCount] = useState(0);
11   const [otherState, setOtherState] = useState(0);
12
13   // SBAGLIATO: Funzione ricreata ogni render
14   const handleClick = () => {
15     setCount(count + 1);
16   };
17
18   // CORRETTO: Funzione memoizzata
19   const handleClickMemo = useCallback(() => {
20     setCount(prev => prev + 1);
21   }, []); // Dipendenze vuote: funzione mai ricreata
22
23   return (
24     <div>
25       <p>Count: {count}</p>
26       <p>Other: {otherState}</p>
27
28       {/* Questo ri-renderizza sempre */}
29       <Button onClick={handleClick}>Increment</Button>
30
31       {/* Questo NON ri-renderizza quando otherState cambia */}
32       <Button onClick={handleClickMemo}>Increment Memo</Button>
33
34       <button onClick={() => setOtherState(otherState + 1)}>
35         Update Other
36       </button>
37     </div>
38   );
39 }
```

### 12.4.2 useCallback con Dipendenze



Listing 12.8: useCallback con dipendenze

```

1 import { useState, useCallback } from 'react';
2
3 function TodoList() {
4   const [todos, setTodos] = useState([]);
5   const [filter, setFilter] = useState('all');
6
7   // Callback con dipendenza da filter
8   const handleAddTodo = useCallback((text) => {
9     const newTodo = {
10       id: Date.now(),
11       text,
12       completed: false,
13       category: filter
14     };
15
16     setTodos(prev => [...prev, newTodo]);
17   }, [filter]); // Ri-crea quando filter cambia
18
19   // Callback senza dipendenze esterne
20   const handleToggleTodo = useCallback((id) => {
21     setTodos(prev =>
22       prev.map(todo =>
23         todo.id === id
24           ? { ...todo, completed: !todo.completed }
25           : todo
26       )
27   ), []]; // Mai ri-creata
28
29   // Callback con multiple dipendenze
30   const handleDeleteCompleted = useCallback(() => {
31     setTodos(prev => prev.filter(todo => !todo.completed));
32     console.log('Deleted in category:', filter);
33   }, [filter]); // Dipende da filter
34
35   return (
36     <div>
37       <AddTodoForm onAdd={handleAddTodo} />
38       <TodoItems todos={todos} onToggle={handleToggleTodo} />
39       <button onClick={handleDeleteCompleted}>
40         Delete Completed
41       </button>
42     </div>
43   );
44 }
45

```

### 12.4.3 Pattern: Event Handlers

Listing 12.9: useCallback per event handlers

```

1 import { useCallback } from 'react';
2
3 function UserProfile({ user, onUpdate }) {
4   // Handler per form submit
5   const handleSubmit = useCallback((e) => {
6     e.preventDefault();

```

```

7   const formData = new FormData(e.target);
8   const updates = Object.fromEntries(formData);
9   onUpdate(user.id, updates);
10  }, [user.id, onUpdate]);
11
12  // Handler per campo specifico
13  const handleNameChange = useCallback((e) => {
14    onUpdate(user.id, { name: e.target.value });
15  }, [user.id, onUpdate]);
16
17  // Handler per click con parametro
18  const handleDelete = useCallback(() => {
19    if (confirm('Delete ${user.name}?')) {
20      onUpdate(user.id, { deleted: true });
21    }
22  }, [user.id, user.name, onUpdate]);
23
24  return (
25    <form onSubmit={handleSubmit}>
26      <input
27        name="name"
28        defaultValue={user.name}
29        onChange={handleNameChange}
30      />
31      <button type="submit">Save</button>
32      <button type="button" onClick={handleDelete}>Delete</button>
33    </form>
34  );
35 }

```

## 12.5 Code Splitting e Lazy Loading

Il code splitting divide il bundle in chunks più piccoli, caricati on-demand.

### 12.5.1 React.lazy e Suspense

Listing 12.10: Lazy loading di componenti

```

1  import { lazy, Suspense } from 'react';
2
3  // Lazy import
4  const Dashboard = lazy(() => import('./pages/Dashboard'));
5  const Profile = lazy(() => import('./pages/Profile'));
6  const Settings = lazy(() => import('./pages/Settings'));
7
8  // Loading component
9  function LoadingSpinner() {
10    return (
11      <div className="loading">
12        <div className="spinner"></div>
13        <p>Caricamento...</p>
14      </div>
15    );
16  }
17
18  function App() {

```

```

19   return (
20     <Suspense fallback={<LoadingSpinner />}>
21       <Routes>
22         <Route path="/dashboard" element={<Dashboard />} />
23         <Route path="/profile" element={<Profile />} />
24         <Route path="/settings" element={<Settings />} />
25       </Routes>
26     </Suspense>
27   );
28 }

```

## 12.5.2 Route-based Code Splitting

Listing 12.11: Code splitting per route

```

1  import { lazy, Suspense } from 'react';
2  import { Routes, Route } from 'react-router-dom';
3
4  // Lazy load route components
5  const Home = lazy(() => import('./routes/Home'));
6  const About = lazy(() => import('./routes/About'));
7  const Products = lazy(() => import('./routes/Products'));
8  const ProductDetail = lazy(() => import('./routes/ProductDetail'));
9  const Admin = lazy(() => import('./routes/Admin'));
10
11 function App() {
12   return (
13     <Suspense fallback={<div>Loading...</div>}>
14       <Routes>
15         <Route path="/" element={<Home />} />
16         <Route path="/about" element={<About />} />
17         <Route path="/products" element={<Products />} />
18         <Route path="/products/:id" element={<ProductDetail />} />
19         <Route path="/admin/*" element={<Admin />} />
20       </Routes>
21     </Suspense>
22   );
23 }

```

## 12.5.3 Component-based Splitting

Listing 12.12: Lazy loading di componenti pesanti

```

1  import { lazy, Suspense, useState } from 'react';
2
3  // Componenti pesanti caricati solo quando necessario
4  const ChartComponent = lazy(() => import('./components/Chart'));
5  const DataTable = lazy(() => import('./components/DataTable'));
6  const VideoPlayer = lazy(() => import('./components/VideoPlayer'));
7
8  function Dashboard() {
9    const [showChart, setShowChart] = useState(false);
10   const [showTable, setShowTable] = useState(false);
11
12   return (
13     <div>

```

```

14     <h1>Dashboard</h1>
15
16     <button onClick={() => setShowChart(true)}>Show Chart</button>
17     <button onClick={() => setShowTable(true)}>Show Table</button>
18
19     {showChart && (
20       <Suspense fallback={<div>Loading chart...</div>}>
21         <ChartComponent data={chartData} />
22       </Suspense>
23     )}
24
25     {showTable && (
26       <Suspense fallback={<div>Loading table...</div>}>
27         <DataTable data={tableData} />
28       </Suspense>
29     )}
30   </div>
31 );
32 }

```

### 12.5.4 Preloading

Listing 12.13: Preload componenti al hover

```

1  import { lazy } from 'react';
2
3  const HeavyComponent = lazy(() => import('./HeavyComponent'));
4
5  // Preload function
6  const preloadHeavyComponent = () => {
7    import('./HeavyComponent');
8  };
9
10 function Navigation() {
11   return (
12     <nav>
13       <Link
14         to="/heavy"
15         onMouseEnter={preloadHeavyComponent}
16         onFocus={preloadHeavyComponent}
17       >
18         Heavy Page
19       </Link>
20     </nav>
21   );
22 }

```

## 12.6 Virtualizzazione Liste

Per liste molto lunghe, renderizza solo elementi visibili.

### 12.6.1 React Window

Listing 12.14: Liste virtualizzate con react-window

```

1  import { FixedSizeList } from 'react-window';
2
3  function VirtualizedList({ items }) {
4    const Row = ({ index, style }) => (
5      <div style={style} className="list-item">
6        {items[index].name}
7      </div>
8    );
9
10   return (
11     <FixedSizeList
12       height={600}
13       itemCount={items.length}
14       itemSize={50}
15       width="100%"
16     >
17       {Row}
18     </FixedSizeList>
19   );
20 }
21
22 // Utilizzo
23 function App() {
24   const items = Array.from({ length: 10000 }, (_, i) => ({
25     id: i,
26     name: `Item ${i}`
27   }));
28
29   return <VirtualizedList items={items} />;
30 }

```

### 12.6.2 Liste a Dimensioni Variabili

Listing 12.15: Liste con altezze variabili

```

1  import { VariableSizeList } from 'react-window';
2
3  function VariableList({ items }) {
4    // Funzione per calcolare altezza
5    const getItemSize = (index) => {
6      const item = items[index];
7      // Calcola in base al contenuto
8      return item.content.length > 100 ? 100 : 50;
9    };
10
11   const Row = ({ index, style }) => (
12     <div style={style} className="list-item">
13       <h3>{items[index].title}</h3>
14       <p>{items[index].content}</p>
15     </div>
16   );
17
18   return (
19     <VariableSizeList
20       height={600}
21       itemCount={items.length}
22       itemSize={getItemSize}

```

```
23     width="100%"
24   >
25     {Row}
26   </VariableSizeList>
27 );
28 }
```

## 12.7 React Profiler

Strumento per misurare performance e identificare bottleneck.

### 12.7.1 Utilizzo del Profiler

Listing 12.16: Profiler Component

```
1 import { Profiler } from 'react';
2
3 function onRenderCallback(
4   id, // id del Profiler
5   phase, // "mount" o "update"
6   actualDuration, // tempo speso per render
7   baseDuration, // tempo stimato senza memoization
8   startTime, // quando React ha iniziato
9   commitTime, // quando React ha committato
10  interactions // Set di interazioni
11 ) {
12   console.log(`${id} (${phase}):`, {
13     actualDuration,
14     baseDuration
15   });
16
17   // Log su servizio analytics
18   // analytics.track('render', { id, phase, actualDuration });
19 }
20
21 function App() {
22   return (
23     <Profiler id="App" onRender={onRenderCallback}>
24       <Dashboard />
25     </Profiler>
26   );
27 }
28
29 // Multiple profilers
30 function ComplexApp() {
31   return (
32     <div>
33       <Profiler id="Navigation" onRender={onRenderCallback}>
34         <Navigation />
35       </Profiler>
36
37       <Profiler id="Content" onRender={onRenderCallback}>
38         <Content />
39       </Profiler>
40
41       <Profiler id="Sidebar" onRender={onRenderCallback}>
```

```

42     <Sidebar />
43   </Profiler>
44 </div>
45 );
46 }

```

## 12.7.2 Custom Performance Hook

Listing 12.17: Hook per misurare performance

```

1  import { useEffect, useRef } from 'react';
2
3  function useRenderTime(componentName) {
4    const renderCount = useRef(0);
5    const startTime = useRef(performance.now());
6
7    useEffect(() => {
8      renderCount.current += 1;
9      const endTime = performance.now();
10     const renderTime = endTime - startTime.current;
11
12     console.log(
13       `${componentName} render #${renderCount.current}: ${renderTime.
14         toFixed(2)}ms`
15     );
16
17     startTime.current = performance.now();
18   });
19 }
20
21 // Utilizzo
22 function ExpensiveComponent() {
23   useRenderTime('ExpensiveComponent');
24
25   // ... component logic
26   return <div>Component</div>;
27 }

```

## 12.8 Ottimizzazioni Varie

### 12.8.1 Debounce e Throttle

Listing 12.18: Debounce per input

```

1  import { useState, useEffect } from 'react';
2
3  function useDebounce(value, delay) {
4    const [debouncedValue, setDebouncedValue] = useState(value);
5
6    useEffect(() => {
7      const timer = setTimeout(() => {
8        setDebouncedValue(value);
9      }, delay);
10
11     return () => {
12       clearTimeout(timer);
13     };
14   }, [value]);
15 }

```

```

13     };
14     }, [value, delay]);
15
16     return debouncedValue;
17 }
18
19 // Utilizzo
20 function SearchComponent() {
21     const [searchTerm, setSearchTerm] = useState('');
22     const debouncedSearch = useDebounce(searchTerm, 500);
23
24     useEffect(() => {
25         if (debouncedSearch) {
26             // Esegui ricerca API
27             console.log('Searching for:', debouncedSearch);
28         }
29     }, [debouncedSearch]);
30
31     return (
32         <input
33             value={searchTerm}
34             onChange={(e) => setSearchTerm(e.target.value)}
35             placeholder="Search..."
36         />
37     );
38 }

```

## 12.8.2 Immagini Ottimizzate

Listing 12.19: Lazy loading immagini

```

1 import { useState, useEffect, useRef } from 'react';
2
3 function LazyImage({ src, alt, placeholder }) {
4     const [imageSrc, setImageSrc] = useState(placeholder);
5     const [imageRef, setImageRef] = useState();
6
7     useEffect(() => {
8         let observer;
9
10        if (imageRef && imageSrc === placeholder) {
11            observer = new IntersectionObserver(
12                entries => {
13                    entries.forEach(entry => {
14                        if (entry.isIntersecting) {
15                            setImageSrc(src);
16                            observer.unobserve(imageRef);
17                        }
18                    });
19                }
20            );
21
22            observer.observe(imageRef);
23        }
24
25        return () => {
26            if (observer && imageRef) {

```



```

27     observer.unobserve(imageRef);
28   }
29 };
30 }, [imageRef, imageSrc, placeholder, src]);
31
32 return (
33   <img
34     ref={setImageRef}
35     src={imageSrc}
36     alt={alt}
37     className={imageSrc === placeholder ? 'loading' : 'loaded'}
38   />
39 );
40 }
41
42 // Utilizzo
43 function Gallery({ images }) {
44   return (
45     <div className="gallery">
46       {images.map(img => (
47         <LazyImage
48           key={img.id}
49           src={img.url}
50           alt={img.title}
51           placeholder="/placeholder.jpg"
52         />
53       ))}
54     </div>
55   );
56 }

```

### 12.8.3 Web Workers

Listing 12.20: Calcoli pesanti in Web Worker

```

1 // worker.js
2 self.addEventListener('message', (e) => {
3   const { data } = e.data;
4
5   // Calcolo pesante
6   const result = data.map(item => {
7     // Elaborazione complessa
8     return item * item * Math.random();
9   });
10
11   self.postMessage({ result });
12 });
13
14 // Componente React
15 import { useState, useEffect } from 'react';
16
17 function HeavyCalculation({ data }) {
18   const [result, setResult] = useState(null);
19   const [loading, setLoading] = useState(false);
20
21   useEffect(() => {
22     const worker = new Worker(new URL('./worker.js', import.meta.url));

```

```
23  
24     worker.onmessage = (e) => {  
25         setResult(e.data.result);  
26         setLoading(false);  
27     };  
28  
29     setLoading(true);  
30     worker.postMessage({ data });  
31  
32     return () => {  
33         worker.terminate();  
34     };  
35 }, [data]);  
36  
37 if (loading) return <div>Calculating...</div>;  
38  
39 return (  
40     <div>  
41         Result: {result?.length} items processed  
42     </div>  
43 );  
44 }
```

## 12.9 Best Practices

### Performance Best Practices

- Misura prima di ottimizzare (React DevTools Profiler)
- Usa React.memo solo per componenti costosi
- useMemo/useCallback solo se migliora performance misurate
- Implementa code splitting per route
- Virtualizza liste molto lunghe
- Lazy load immagini e componenti pesanti
- Debounce input che triggherano API calls
- Evita inline functions in props di componenti memoizzati
- Usa key stabili in liste
- Minimizza re-render con state colocation
- Considera Web Workers per calcoli pesanti
- Ottimizza bundle size (tree shaking, minification)

## 12.10 Conclusioni

L'ottimizzazione delle performance richiede un approccio misurato e strategico. Gli strumenti principali sono:

- **React.memo**: Per evitare re-render di componenti

- **useMemo/useCallback:** Per memoizzare valori e funzioni
- **Code Splitting:** Per ridurre bundle size iniziale
- **Virtualizzazione:** Per liste molto lunghe
- **Profiler:** Per identificare bottleneck

Ricorda: premature optimization è inutile. Misura, identifica problemi reali, poi ottimizza in modo mirato.



# Capitolo 13

## Testing in React

### 13.1 Introduzione al Testing

Il testing è fondamentale per creare applicazioni affidabili e manutenibili. Questo capitolo copre Jest e React Testing Library per unit e integration tests.

#### 13.1.1 Tipi di Test

- **Unit Tests:** Testano singole funzioni o componenti isolati
- **Integration Tests:** Testano interazione tra componenti
- **End-to-End Tests:** Testano flussi utente completi (Cypress, Playwright)

#### 13.1.2 Setup

Listing 13.1: Installazione dipendenze test

```
1 # Con Create React App, già configurato
2 # Altrimenti installa:
3
4 npm install --save-dev jest @testing-library/react @testing-library/jest
  -dom @testing-library/user-event
5
6 # o con yarn
7 yarn add --dev jest @testing-library/react @testing-library/jest-dom
  @testing-library/user-event
```

### 13.2 React Testing Library

React Testing Library promuove best practices: test what users see, not implementation details.

#### 13.2.1 Primo Test

Listing 13.2: Test base di un componente

```
1 // Button.jsx
2 function Button({ onClick, children }) {
3   return <button onClick={onClick}>{children}</button>;
4 }
5
6 export default Button;
```

```

7
8 // Button.test.jsx
9 import { render, screen } from '@testing-library/react';
10 import userEvent from '@testing-library/user-event';
11 import Button from './Button';
12
13 describe('Button Component', () => {
14   test('renders button with text', () => {
15     render(<Button>Click me</Button>);
16
17     const buttonElement = screen.getByText(/click me/i);
18     expect(buttonElement).toBeInTheDocument();
19   });
20
21   test('calls onClick when clicked', async () => {
22     const handleClick = jest.fn();
23     const user = userEvent.setup();
24
25     render(<Button onClick={handleClick}>Click</Button>);
26
27     const button = screen.getByText(/click/i);
28     await user.click(button);
29
30     expect(handleClick).toHaveBeenCalledTimes(1);
31   });
32 });

```

### 13.2.2 Query Methods

Listing 13.3: Metodi per trovare elementi

```

1 import { render, screen } from '@testing-library/react';
2
3 function MyComponent() {
4   return (
5     <div>
6       <h1>Hello World</h1>
7       <button aria-label="submit">Submit</button>
8       <input type="text" placeholder="Enter name" />
9       
10    </div>
11  );
12 }
13
14 test('query methods', () => {
15   render(<MyComponent />);
16
17   // getBy* - Trova elemento, error se non trovato
18   const heading = screen.getByText(/hello world/i);
19   const button = screen.getByRole('button', { name: /submit/i });
20   const input = screen.getByPlaceholderText(/enter name/i);
21   const image = screen.getByAltText(/logo/i);
22
23   // queryBy* - Trova elemento, null se non trovato
24   const missing = screen.queryByText(/not here/i);
25   expect(missing).not.toBeInTheDocument();
26

```

```

27 // findBy* - Async, aspetta elemento (per caricamenti asincroni)
28 const asyncElement = await screen.findByText(/loaded/i);
29
30 // getAllBy*, queryAllBy*, findAllBy* - Multipli elementi
31 const buttons = screen.getAllByRole('button');
32 expect(buttons).toHaveLength(1);
33 });

```

### 13.2.3 Test di Componenti con State

Listing 13.4: Test componente con useState

```

1 // Counter.jsx
2 import { useState } from 'react';
3
4 function Counter() {
5   const [count, setCount] = useState(0);
6
7   return (
8     <div>
9       <p>Count: {count}</p>
10      <button onClick={() => setCount(count + 1)}>Increment</button>
11      <button onClick={() => setCount(count - 1)}>Decrement</button>
12      <button onClick={() => setCount(0)}>Reset</button>
13    </div>
14  );
15 }
16
17 export default Counter;
18
19 // Counter.test.jsx
20 import { render, screen } from '@testing-library/react';
21 import userEvent from '@testing-library/user-event';
22 import Counter from './Counter';
23
24 describe('Counter', () => {
25   test('renders initial count of 0', () => {
26     render(<Counter />);
27     expect(screen.getByText(/count: 0/i)).toBeInTheDocument();
28   });
29
30   test('increments count when increment button is clicked', async () => {
31     {
32       const user = userEvent.setup();
33       render(<Counter />);
34
35       const incrementBtn = screen.getByRole('button', { name: /increment/i });
36       await user.click(incrementBtn);
37
38       expect(screen.getByText(/count: 1/i)).toBeInTheDocument();
39     }
40   });
41
42   test('decrements count when decrement button is clicked', async () => {
43     {
44       const user = userEvent.setup();
45       render(<Counter />);

```

```

43
44     const decrementBtn = screen.getByRole('button', { name: /decrement/i
45         });
46     await user.click(decrementBtn);
47
48     expect(screen.getByText(/count: -1/i)).toBeInTheDocument();
49 });
50
51 test('resets count to 0', async () => {
52     const user = userEvent.setup();
53     render(<Counter />);
54
55     // Incrementa a 5
56     const incrementBtn = screen.getByRole('button', { name: /increment/i
57         });
58     await user.click(incrementBtn);
59     await user.click(incrementBtn);
60     await user.click(incrementBtn);
61     await user.click(incrementBtn);
62     await user.click(incrementBtn);
63
64     // Reset
65     const resetBtn = screen.getByRole('button', { name: /reset/i });
66     await user.click(resetBtn);
67
68     expect(screen.getByText(/count: 0/i)).toBeInTheDocument();
69 });

```

### 13.2.4 Test di Form

Listing 13.5: Test interazioni form

```

1 // LoginForm.jsx
2 import { useState } from 'react';
3
4 function LoginForm({ onSubmit }) {
5     const [email, setEmail] = useState('');
6     const [password, setPassword] = useState('');
7     const [error, setError] = useState('');
8
9     const handleSubmit = (e) => {
10         e.preventDefault();
11
12         if (!email || !password) {
13             setError('Email and password are required');
14             return;
15         }
16
17         onSubmit({ email, password });
18     };
19
20     return (
21         <form onSubmit={handleSubmit}>
22             <input
23                 type="email"
24                 value={email}

```



```

25     onChange={e => setEmail(e.target.value)}
26     placeholder="Email"
27     aria-label="Email"
28   />
29
30   <input
31     type="password"
32     value={password}
33     onChange={e => setPassword(e.target.value)}
34     placeholder="Password"
35     aria-label="Password"
36   />
37
38   <button type="submit">Login</button>
39
40   {error && <div role="alert">{error}</div>}
41 </form>
42 );
43 }
44
45 export default LoginForm;
46
47 // LoginForm.test.jsx
48 import { render, screen } from '@testing-library/react';
49 import userEvent from '@testing-library/user-event';
50 import LoginForm from './LoginForm';
51
52 describe('LoginForm', () => {
53   test('renders login form', () => {
54     render(<LoginForm onSubmit={() => {}} />);
55
56     expect(screen.getByLabelText(/email/i)).toBeInTheDocument();
57     expect(screen.getByLabelText(/password/i)).toBeInTheDocument();
58     expect(screen.getByRole('button', { name: /login/i })).
59       toBeInTheDocument();
60   });
61
62   test('submits form with email and password', async () => {
63     const handleSubmit = jest.fn();
64     const user = userEvent.setup();
65
66     render(<LoginForm onSubmit={handleSubmit} />);
67
68     // Compila form
69     await user.type(screen.getByLabelText(/email/i), 'test@example.com');
70     ;
71     await user.type(screen.getByLabelText(/password/i), 'password123');
72
73     // Submit
74     await user.click(screen.getByRole('button', { name: /login/i }));
75
76     // Verifica chiamata
77     expect(handleSubmit).toHaveBeenCalledWith({
78       email: 'test@example.com',
79       password: 'password123'
80     });
81   });
82 });

```

```

81 test('shows error when fields are empty', async () => {
82   const user = userEvent.setup();
83   render(<LoginForm onSubmit={() => {}} />);
84
85   // Submit senza compilare
86   await user.click(screen.getByRole('button', { name: /login/i }));
87
88   // Verifica errore
89   expect(screen.getByRole('alert')).toHaveTextContent(
90     /email and password are required/i
91   );
92 });
93 });

```

## 13.3 Test di Componenti Asincroni

### 13.3.1 Mock di Fetch/API

Listing 13.6: Test con chiamate API mockate

```

1 // UserList.jsx
2 import { useState, useEffect } from 'react';
3
4 function UserList() {
5   const [users, setUsers] = useState([]);
6   const [loading, setLoading] = useState(true);
7   const [error, setError] = useState(null);
8
9   useEffect(() => {
10     fetch('https://api.example.com/users')
11       .then(res => res.json())
12       .then(data => {
13         setUsers(data);
14         setLoading(false);
15       })
16       .catch(err => {
17         setError(err.message);
18         setLoading(false);
19       });
20   }, []);
21
22   if (loading) return <div>Loading...</div>;
23   if (error) return <div>Error: {error}</div>;
24
25   return (
26     <ul>
27       {users.map(user => (
28         <li key={user.id}>{user.name}</li>
29       ))}
30     </ul>
31   );
32 }
33
34 export default UserList;
35
36 // UserList.test.jsx
37 import { render, screen, waitFor } from '@testing-library/react';

```

```
38 import UserList from './UserList';
39
40 // Mock fetch globalmente
41 global.fetch = jest.fn();
42
43 describe('UserList', () => {
44   beforeEach(() => {
45     fetch.mockClear();
46   });
47
48   test('displays loading state initially', () => {
49     fetch.mockImplementation(() => new Promise(() => {})); // Never
       resolves
50
51     render(<UserList />);
52
53     expect(screen.getByText(/loading/i)).toBeInTheDocument();
54   });
55
56   test('displays users after successful fetch', async () => {
57     const mockUsers = [
58       { id: 1, name: 'Alice' },
59       { id: 2, name: 'Bob' }
60     ];
61
62     fetch.mockResolvedValueOnce({
63       json: async () => mockUsers
64     });
65
66     render(<UserList />);
67
68     // Aspetta che loading scompaia
69     await waitFor(() => {
70       expect(screen.queryByText(/loading/i)).not.toBeInTheDocument();
71     });
72
73     // Verifica utenti renderizzati
74     expect(screen.getByText('Alice')).toBeInTheDocument();
75     expect(screen.getByText('Bob')).toBeInTheDocument();
76   });
77
78   test('displays error on fetch failure', async () => {
79     fetch.mockRejectedValueOnce(new Error('Network error'));
80
81     render(<UserList />);
82
83     await waitFor(() => {
84       expect(screen.getByText(/error: network error/i)).
         toBeInTheDocument();
85     });
86   });
87 });
```

### 13.3.2 Mock Service Worker (MSW)

Listing 13.7: Setup MSW per mock API

```
1 // mocks/handlers.js
2 import { rest } from 'msw';
3
4 export const handlers = [
5   rest.get('https://api.example.com/users', (req, res, ctx) => {
6     return res(
7       ctx.json([
8         { id: 1, name: 'Alice' },
9         { id: 2, name: 'Bob' }
10      ])
11    );
12  }
13 ],
14
15   rest.post('https://api.example.com/users', (req, res, ctx) => {
16     const { name } = req.body;
17
18     return res(
19       ctx.json({
20         id: 3,
21         name
22       })
23     );
24   }
25 ];
26
27 // mocks/server.js
28 import { setupServer } from 'msw/node';
29 import { handlers } from './handlers';
30
31 export const server = setupServer(...handlers);
32
33 // setupTests.js
34 import '@testing-library/jest-dom';
35 import { server } from './mocks/server';
36
37 beforeAll(() => server.listen());
38 afterAll(() => server.close());
39
40 // Test con MSW
41 import { render, screen, waitFor } from '@testing-library/react';
42 import UserList from './UserList';
43
44 test('fetches and displays users', async () => {
45   render(<UserList />);
46
47   await waitFor(() => {
48     expect(screen.getByText('Alice')).toBeInTheDocument();
49     expect(screen.getByText('Bob')).toBeInTheDocument();
50   });
51 });
```

## 13.4 Test di Context e State Management

### 13.4.1 Test con Context

Listing 13.8: Test componenti con Context

```

1 // ThemeContext.jsx
2 import { createContext, useContext, useState } from 'react';
3
4 const ThemeContext = createContext();
5
6 export function ThemeProvider({ children }) {
7   const [theme, setTheme] = useState('light');
8
9   const toggleTheme = () => {
10     setTheme(prev => prev === 'light' ? 'dark' : 'light');
11   };
12
13   return (
14     <ThemeContext.Provider value={{ theme, toggleTheme }}>
15       {children}
16     </ThemeContext.Provider>
17   );
18 }
19
20 export function useTheme() {
21   return useContext(ThemeContext);
22 }
23
24 // ThemedButton.jsx
25 import { useTheme } from './ThemeContext';
26
27 function ThemedButton() {
28   const { theme, toggleTheme } = useTheme();
29
30   return (
31     <button className={theme} onClick={toggleTheme}>
32       Current theme: {theme}
33     </button>
34   );
35 }
36
37 // ThemedButton.test.jsx
38 import { render, screen } from '@testing-library/react';
39 import userEvent from '@testing-library/user-event';
40 import { ThemeProvider } from './ThemeContext';
41 import ThemedButton from './ThemedButton';
42
43 describe('ThemedButton', () => {
44   test('displays current theme', () => {
45     render(
46       <ThemeProvider>
47         <ThemedButton />
48       </ThemeProvider>
49     );
50
51     expect(screen.getByText(/current theme: light/i)).toBeInTheDocument
52       ();
53   });
54
55   test('toggles theme on click', async () => {
56     const user = userEvent.setup();

```

```

57     render(
58       <ThemeProvider>
59         <ThemedButton />
60       </ThemeProvider>
61     );
62
63     const button = screen.getByRole('button');
64
65     // Inizialmente light
66     expect(button).toHaveTextContent(/light/i);
67
68     // Click per toggle
69     await user.click(button);
70     expect(button).toHaveTextContent(/dark/i);
71
72     // Click di nuovo
73     await user.click(button);
74     expect(button).toHaveTextContent(/light/i);
75   });
76 });

```

### 13.4.2 Custom Render con Providers

Listing 13.9: Helper per render con providers

```

1 // test-utils.jsx
2 import { render } from '@testing-library/react';
3 import { ThemeProvider } from '../ThemeContext';
4 import { AuthProvider } from '../AuthContext';
5 import { BrowserRouter } from 'react-router-dom';
6
7 function AllProviders({ children }) {
8   return (
9     <BrowserRouter>
10       <AuthProvider>
11         <ThemeProvider>
12           {children}
13         </ThemeProvider>
14       </AuthProvider>
15     </BrowserRouter>
16   );
17 }
18
19 function customRender(ui, options) {
20   return render(ui, { wrapper: AllProviders, ...options });
21 }
22
23 // Re-export tutto
24 export * from '@testing-library/react';
25 export { customRender as render };
26
27 // Utilizzo
28 import { render, screen } from './test-utils';
29
30 test('my test', () => {
31   render(<MyComponent />);
32   // Component automaticamente wrappato con providers

```

```
33 });
```

## 13.5 Test di Hooks Personalizzati

Listing 13.10: Test custom hooks

```
1 // useCounter.js
2 import { useState, useCallback } from 'react';
3
4 function useCounter(initialValue = 0) {
5   const [count, setCount] = useState(initialValue);
6
7   const increment = useCallback(() => {
8     setCount(c => c + 1);
9   }, []);
10
11   const decrement = useCallback(() => {
12     setCount(c => c - 1);
13   }, []);
14
15   const reset = useCallback(() => {
16     setCount(initialValue);
17   }, [initialValue]);
18
19   return { count, increment, decrement, reset };
20 }
21
22 export default useCounter;
23
24 // useCounter.test.js
25 import { renderHook, act } from '@testing-library/react';
26 import useCounter from './useCounter';
27
28 describe('useCounter', () => {
29   test('initializes with default value', () => {
30     const { result } = renderHook(() => useCounter());
31     expect(result.current.count).toBe(0);
32   });
33
34   test('initializes with custom value', () => {
35     const { result } = renderHook(() => useCounter(10));
36     expect(result.current.count).toBe(10);
37   });
38
39   test('increments count', () => {
40     const { result } = renderHook(() => useCounter());
41
42     act(() => {
43       result.current.increment();
44     });
45
46     expect(result.current.count).toBe(1);
47   });
48
49   test('decrements count', () => {
50     const { result } = renderHook(() => useCounter(5));
51
```

```

52     act(() => {
53         result.current.decrement();
54     });
55
56     expect(result.current.count).toBe(4);
57 });
58
59 test('resets to initial value', () => {
60     const { result } = renderHook(() => useCounter(10));
61
62     act(() => {
63         result.current.increment();
64         result.current.increment();
65     });
66
67     expect(result.current.count).toBe(12);
68
69     act(() => {
70         result.current.reset();
71     });
72
73     expect(result.current.count).toBe(10);
74 });
75 });

```

## 13.6 Snapshot Testing

Listing 13.11: Snapshot tests

```

1  import { render } from '@testing-library/react';
2  import Button from './Button';
3
4  test('matches snapshot', () => {
5      const { container } = render(<Button>Click me</Button>);
6      expect(container).toMatchSnapshot();
7  });
8
9  // Snapshot inline
10 test('inline snapshot', () => {
11     const { container } = render(<Button primary>Submit</Button>);
12
13     expect(container.firstChild).toMatchInlineSnapshot(`
14         <button
15             class="primary"
16         >
17             Submit
18         </button>
19     `);
20 });

```

## 13.7 Integration Tests

Listing 13.12: Test integrazione completa

```

1  // TodoApp.jsx

```



```

2 import { useState } from 'react';
3
4 function TodoApp() {
5   const [todos, setTodos] = useState([]);
6   const [input, setInput] = useState('');
7
8   const addTodo = () => {
9     if (input.trim()) {
10       setTodos([...todos, { id: Date.now(), text: input, completed:
11         false }]);
12       setInput('');
13     }
14   };
15
16   const toggleTodo = (id) => {
17     setTodos(todos.map(todo =>
18       todo.id === id ? { ...todo, completed: !todo.completed } : todo
19     ));
20   };
21
22   const deleteTodo = (id) => {
23     setTodos(todos.filter(todo => todo.id !== id));
24   };
25
26   return (
27     <div>
28       <h1>Todo App</h1>
29
30       <input
31         value={input}
32         onChange={(e) => setInput(e.target.value)}
33         onKeyDown={(e) => e.key === 'Enter' && addTodo()}
34         placeholder="Add todo"
35       />
36       <button onClick={addTodo}>Add</button>
37
38       <ul>
39         {todos.map(todo => (
40           <li key={todo.id}>
41             <input
42               type="checkbox"
43               checked={todo.completed}
44               onChange={() => toggleTodo(todo.id)}
45             />
46             <span style={{ textDecoration: todo.completed ? 'line-
47               through' : 'none' }}>
48               {todo.text}
49             </span>
50             <button onClick={() => deleteTodo(todo.id)}>Delete</button>
51           </li>
52         ))}
53       </ul>
54
55       <p>Total: {todos.length}, Completed: {todos.filter(t => t.
56         completed).length}</p>
57     </div>
58   );
59 }

```

```

57
58 export default TodoApp;
59
60 // TodoApp.test.jsx
61 import { render, screen } from '@testing-library/react';
62 import userEvent from '@testing-library/user-event';
63 import TodoApp from './TodoApp';
64
65 describe('TodoApp Integration', () => {
66   test('complete todo workflow', async () => {
67     const user = userEvent.setup();
68     render(<TodoApp />);
69
70     // Verifica stato iniziale
71     expect(screen.getByText(/total: 0/i)).toBeInTheDocument();
72
73     // Aggiungi primo todo
74     const input = screen.getByPlaceholderText(/add todo/i);
75     await user.type(input, 'Buy milk');
76     await user.click(screen.getByRole('button', { name: /add/i }));
77
78     // Verifica todo aggiunto
79     expect(screen.getByText('Buy milk')).toBeInTheDocument();
80     expect(screen.getByText(/total: 1/i)).toBeInTheDocument();
81
82     // Aggiungi secondo todo
83     await user.type(input, 'Walk dog');
84     await user.click(screen.getByRole('button', { name: /add/i }));
85
86     expect(screen.getByText('Walk dog')).toBeInTheDocument();
87     expect(screen.getByText(/total: 2/i)).toBeInTheDocument();
88
89     // Completa primo todo
90     const checkboxes = screen.getAllByRole('checkbox');
91     await user.click(checkboxes[0]);
92
93     expect(screen.getByText(/completed: 1/i)).toBeInTheDocument();
94
95     // Elimina secondo todo
96     const deleteButtons = screen.getAllByRole('button', { name: /delete/i });
97     await user.click(deleteButtons[1]);
98
99     expect(screen.queryByText('Walk dog')).not.toBeInTheDocument();
100     expect(screen.getByText(/total: 1/i)).toBeInTheDocument();
101   });
102 });

```

## 13.8 Best Practices

### Testing Best Practices

- Test behavior, not implementation
- Query by accessibility (role, label, text)
- Usa `userEvent` invece di `fireEvent`

- Mock API calls con MSW
- Test casi di errore
- Mantieni tests semplici e leggibili
- Un test, un comportamento
- Usa describe per raggruppare test correlati
- Setup e cleanup con beforeEach/afterEach
- Evita snapshot testing eccessivo
- Test copertura critica, non 100 percent
- Integra CI/CD per eseguire tests automaticamente

## 13.9 Coverage

Listing 13.13: Generare coverage report

```
1 # Run tests con coverage
2 npm test -- --coverage --watchAll=false
3
4 # o aggiungi script in package.json
5 {
6   "scripts": {
7     "test": "react-scripts test",
8     "test:coverage": "react-scripts test --coverage --watchAll=false"
9   }
10 }
11
12 # Coverage output mostra:
13 # - Statements: linee di codice eseguite
14 # - Branches: condizioni if/else testate
15 # - Functions: funzioni chiamate
16 # - Lines: righe coperte
```

## 13.10 Conclusioni

Il testing in React con Jest e React Testing Library permette di creare applicazioni robuste e manutenibili. I punti chiave sono:

- Test comportamento utente, non implementazione
- Mock API con MSW per tests realistici
- Test hooks con renderHook
- Integration tests per flussi completi
- CI/CD per eseguire tests automaticamente

Un buon test suite aumenta confidenza nel codice e facilita refactoring sicuri.



# Appendice A

## Appendice: Reference Hooks

### A.1 Introduzione

Questa appendice fornisce una reference completa di tutti gli hooks React built-in e pattern comuni per custom hooks.

### A.2 Hooks di Stato

#### A.2.1 useState

Hook per aggiungere stato locale a componenti funzionali.

Listing A.1: useState - Reference completa

```
1 import { useState } from 'react';
2
3 // Sintassi base
4 const [state, setState] = useState(initialState);
5
6 // Con valore primitivo
7 const [count, setCount] = useState(0);
8 const [name, setName] = useState('');
9 const [isActive, setActive] = useState(false);
10
11 // Con oggetto
12 const [user, setUser] = useState({ name: '', age: 0 });
13
14 // Con array
15 const [items, setItems] = useState([]);
16
17 // Con funzione di inizializzazione (lazy init)
18 const [expensive, setExpensive] = useState(() => {
19   return computeExpensiveValue();
20 });
21
22 // Update con valore diretto
23 setCount(5);
24 setName('Alice');
25
26 // Update con funzione (prev state)
27 setCount(prevCount => prevCount + 1);
28 setUser(prevUser => ({ ...prevUser, age: 30 }));
29 setItems(prevItems => [...prevItems, newItem]);
30
```

```

31 // Esempio completo
32 function Counter() {
33   const [count, setCount] = useState(0);
34
35   return (
36     <div>
37       <p>{count}</p>
38       <button onClick={() => setCount(count + 1)}>+</button>
39       <button onClick={() => setCount(prev => prev - 1)}>-</button>
40     </div>
41   );
42 }

```

### A.2.2 useReducer

Hook per gestire stato complesso con reducer pattern.

Listing A.2: useReducer - Reference completa

```

1  import { useReducer } from 'react';
2
3  // Sintassi
4  const [state, dispatch] = useReducer(reducer, initialState, init);
5
6  // Reducer function
7  function reducer(state, action) {
8    switch (action.type) {
9      case 'INCREMENT':
10       return { count: state.count + 1 };
11      case 'DECREMENT':
12       return { count: state.count - 1 };
13      case 'RESET':
14       return { count: 0 };
15      default:
16       throw new Error('Unknown action');
17    }
18  }
19
20  // Uso base
21  function Counter() {
22    const [state, dispatch] = useReducer(reducer, { count: 0 });
23
24    return (
25      <div>
26        <p>{state.count}</p>
27        <button onClick={() => dispatch({ type: 'INCREMENT' })}>+</button>
28        <button onClick={() => dispatch({ type: 'DECREMENT' })}>-</button>
29        <button onClick={() => dispatch({ type: 'RESET' })}>Reset</button>
30      </div>
31    );
32  }
33
34  // Con payload
35  function todoReducer(state, action) {
36    switch (action.type) {
37      case 'ADD':
38       return [...state, { id: Date.now(), text: action.payload }];
39      case 'DELETE':

```

```

40     return state.filter(todo => todo.id !== action.payload);
41   case 'TOGGLE':
42     return state.map(todo =>
43       todo.id === action.payload
44         ? { ...todo, completed: !todo.completed }
45         : todo
46     );
47   default:
48     return state;
49 }
50 }
51
52 // Con lazy initialization
53 const init = (initialCount) => ({ count: initialCount });
54
55 const [state, dispatch] = useReducer(reducer, 10, init);

```

## A.3 Hooks di Effetti

### A.3.1 useEffect

Hook per effetti collaterali (side effects).

Listing A.3: useEffect - Reference completa

```

1  import { useEffect } from 'react';
2
3  // Sintassi
4  useEffect(effectFunction, dependencies);
5
6  // 1. Esegue ad ogni render
7  useEffect(() => {
8    console.log('Ogni render');
9  });
10
11 // 2. Esegue solo al mount
12 useEffect(() => {
13   console.log('Solo al mount');
14 }, []);
15
16 // 3. Esegue quando dipendenze cambiano
17 useEffect(() => {
18   console.log('userId cambiato:', userId);
19 }, [userId]);
20
21 // Con cleanup
22 useEffect(() => {
23   const timer = setInterval(() => {
24     console.log('Tick');
25   }, 1000);
26
27   // Cleanup function
28   return () => {
29     clearInterval(timer);
30   };
31 }, []);
32
33 // Esempi pratici

```

```

34 // Fetch dati
35 useEffect(() => {
36   fetch('/api/users/${userId}')
37     .then(res => res.json())
38     .then(setUser);
39 }, [userId]);
40
41 // Event listener
42 useEffect(() => {
43   const handleResize = () => setWidth(window.innerWidth);
44   window.addEventListener('resize', handleResize);
45   return () => window.removeEventListener('resize', handleResize);
46 }, []);
47
48 // Document title
49 useEffect(() => {
50   document.title = 'Count: ${count}';
51 }, [count]);
52
53 // Subscription
54 useEffect(() => {
55   const subscription = dataSource.subscribe();
56   return () => subscription.unsubscribe();
57 }, []);

```

### A.3.2 useLayoutEffect

Come `useEffect` ma esegue sincronamente dopo DOM mutations.

Listing A.4: `useLayoutEffect` - Reference

```

1 import { useLayoutEffect, useRef } from 'react';
2
3 // Usa per misurazioni DOM o scroll position
4 useLayoutEffect(() => {
5   const { height } = ref.current.getBoundingClientRect();
6   setHeight(height);
7 }, []);
8
9 // Esempio: Tooltip positioning
10 function Tooltip({ targetRef, children }) {
11   const tooltipRef = useRef(null);
12   const [position, setPosition] = useState({ x: 0, y: 0 });
13
14   useLayoutEffect(() => {
15     const targetRect = targetRef.current.getBoundingClientRect();
16     const tooltipRect = tooltipRef.current.getBoundingClientRect();
17
18     setPosition({
19       x: targetRect.left + targetRect.width / 2 - tooltipRect.width / 2,
20       y: targetRect.top - tooltipRect.height - 10
21     });
22   }, [targetRef]);
23
24   return (
25     <div
26       ref={tooltipRef}
27       style={{ position: 'absolute', left: position.x, top: position.y

```



```
28     >
29     {children}
30   </div>
31 );
32 }
```

## A.4 Hooks di Ref

### A.4.1 useRef

Hook per valori mutabili che persistono tra render senza causare re-render.

Listing A.5: useRef - Reference completa

```
1  import { useRef } from 'react';
2
3  // Sintassi
4  const ref = useRef(initialValue);
5
6  // 1. DOM reference
7  function InputFocus() {
8    const inputRef = useRef(null);
9
10   const focusInput = () => {
11     inputRef.current.focus();
12   };
13
14   return (
15     <div>
16       <input ref={inputRef} />
17       <button onClick={focusInput}>Focus</button>
18     </div>
19   );
20 }
21
22 // 2. Valore mutabile (non causa re-render)
23 function Timer() {
24   const intervalRef = useRef(null);
25
26   const start = () => {
27     intervalRef.current = setInterval(() => {
28       console.log('Tick');
29     }, 1000);
30   };
31
32   const stop = () => {
33     clearInterval(intervalRef.current);
34   };
35
36   return (
37     <div>
38       <button onClick={start}>Start</button>
39       <button onClick={stop}>Stop</button>
40     </div>
41   );
42 }
43
44 // 3. Previous value
```

```

45 function Counter() {
46   const [count, setCount] = useState(0);
47   const prevCountRef = useRef();
48
49   useEffect(() => {
50     prevCountRef.current = count;
51   }, [count]);
52
53   return (
54     <div>
55       <p>Now: {count}, Before: {prevCountRef.current}</p>
56       <button onClick={() => setCount(count + 1)}>Increment</button>
57     </div>
58   );
59 }
60
61 // 4. Instance variable (come in class component)
62 function Component() {
63   const renderCountRef = useRef(0);
64
65   renderCountRef.current += 1;
66
67   return <div>Renders: {renderCountRef.current}</div>;
68 }

```

#### A.4.2 useImperativeHandle

Personalizza l'istanza del ref esposto a componenti padre.

Listing A.6: useImperativeHandle - Reference

```

1  import { useRef, useImperativeHandle, forwardRef } from 'react';
2
3  // Componente figlio
4  const FancyInput = forwardRef((props, ref) => {
5    const inputRef = useRef();
6
7    useImperativeHandle(ref, () => ({
8      focus: () => {
9        inputRef.current.focus();
10      },
11      scrollIntoView: () => {
12        inputRef.current.scrollIntoView();
13      },
14      getValue: () => {
15        return inputRef.current.value;
16      }
17    }));
18
19    return <input ref={inputRef} {...props} />;
20  });
21
22  // Componente padre
23  function Parent() {
24    const inputRef = useRef();
25
26    const handleClick = () => {
27      inputRef.current.focus();

```

```
28     console.log('Value:', inputRef.current.getValue());
29   };
30
31   return (
32     <div>
33       <FancyInput ref={inputRef} />
34       <button onClick={handleClick}>Focus & Get Value</button>
35     </div>
36   );
37 }
```

## A.5 Hooks di Performance

### A.5.1 useMemo

Memoizza valore calcolato.

Listing A.7: useMemo - Reference completa

```
1  import { useMemo } from 'react';
2
3  // Sintassi
4  const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b])
5    ;
6
7  // Esempi
8  // 1. Calcolo costoso
9  const sortedList = useMemo(() => {
10    return items.sort((a, b) => a.value - b.value);
11  }, [items]);
12
13 // 2. Filtro
14 const filteredUsers = useMemo(() => {
15   return users.filter(user =>
16     user.name.toLowerCase().includes(searchTerm.toLowerCase())
17   );
18 }, [users, searchTerm]);
19
20 // 3. Oggetto config
21 const chartConfig = useMemo(() => ({
22   type: 'line',
23   data: chartData,
24   options: { responsive: true }
25 }, [chartData]);
26
27 // 4. Aggregazione
28 const stats = useMemo(() => {
29   const sum = numbers.reduce((acc, n) => acc + n, 0);
30   const avg = sum / numbers.length;
31   return { sum, avg, count: numbers.length };
32 }, [numbers]);
```

### A.5.2 useCallback

Memoizza funzione.

Listing A.8: useCallback - Reference completa

```
1 import { useCallback } from 'react';
2
3 // Sintassi
4 const memoizedCallback = useCallback(fn, dependencies);
5
6 // Esempi
7 // 1. Event handler
8 const handleClick = useCallback(() => {
9   console.log('Clicked:', count);
10 }, [count]);
11
12 // 2. Con parametro
13 const handleDelete = useCallback((id) => {
14   setItems(items => items.filter(item => item.id !== id));
15 }, []);
16
17 // 3. Submit handler
18 const handleSubmit = useCallback((e) => {
19   e.preventDefault();
20   onSave({ name, email });
21 }, [name, email, onSave]);
22
23 // 4. Passato a child memoizzato
24 const MemoChild = memo(Child);
25
26 function Parent() {
27   const handleUpdate = useCallback((data) => {
28     console.log('Update:', data);
29   }, []);
30
31   return <MemoChild onUpdate={handleUpdate} />;
32 }
```

## A.6 Hooks di Context

### A.6.1 useContext

Accede a Context value.

Listing A.9: useContext - Reference completa

```
1 import { createContext, useContext } from 'react';
2
3 // Crea context
4 const ThemeContext = createContext('light');
5 const UserContext = createContext(null);
6
7 // Provider
8 function App() {
9   return (
10     <ThemeContext.Provider value="dark">
11       <UserContext.Provider value={{ name: 'Alice', role: 'admin' }}>
12         <Component />
13       </UserContext.Provider>
14     </ThemeContext.Provider>
15   );
16 }
17
```

```

18 // Consumer con useContext
19 function Component() {
20   const theme = useContext(ThemeContext);
21   const user = useContext(UserContext);
22
23   return (
24     <div className={theme}>
25       Hello, {user.name} ({user.role})
26     </div>
27   );
28 }
29
30 // Pattern: Custom hook per context
31 const AuthContext = createContext(null);
32
33 export function AuthProvider({ children }) {
34   const [user, setUser] = useState(null);
35
36   const login = (userData) => setUser(userData);
37   const logout = () => setUser(null);
38
39   return (
40     <AuthContext.Provider value={{ user, login, logout }}>
41       {children}
42     </AuthContext.Provider>
43   );
44 }
45
46 export function useAuth() {
47   const context = useContext(AuthContext);
48   if (!context) {
49     throw new Error('useAuth must be used within AuthProvider');
50   }
51   return context;
52 }

```

## A.7 Hooks di Transizione

### A.7.1 useTransition

Marca aggiornamenti come transizioni (bassa priorità).

Listing A.10: useTransition - Reference

```

1 import { useState, useTransition } from 'react';
2
3 function SearchResults() {
4   const [query, setQuery] = useState('');
5   const [results, setResults] = useState([]);
6   const [isPending, startTransition] = useTransition();
7
8   const handleChange = (e) => {
9     const value = e.target.value;
10    setQuery(value); // Alta priorità (immediato)
11
12    // Bassa priorità (può essere interrotto)
13    startTransition(() => {
14      const filtered = allItems.filter(item =>

```

```
15         item.name.includes(value)
16     );
17     setResults(filtered);
18 });
19 };
20
21 return (
22     <div>
23         <input value={query} onChange={handleChange} />
24         {isPending && <div>Loading...</div>}
25         <Results items={results} />
26     </div>
27 );
28 }
```

### A.7.2 useDeferredValue

Differisce l'aggiornamento di un valore.

Listing A.11: useDeferredValue - Reference

```
1 import { useState, useDeferredValue } from 'react';
2
3 function Search() {
4     const [query, setQuery] = useState('');
5     const deferredQuery = useDeferredValue(query);
6
7     // query si aggiorna immediatamente
8     // deferredQuery si aggiorna con delay (bassa priorità)
9
10    return (
11        <div>
12            <input value={query} onChange={(e) => setQuery(e.target.value)} />
13            <SlowList query={deferredQuery} />
14        </div>
15    );
16 }
17
18 function SlowList({ query }) {
19     // Questo componente usa deferredQuery
20     // Re-render ritardato se user sta ancora digitando
21     const items = useMemo(() => {
22         return hugeList.filter(item => item.includes(query));
23     }, [query]);
24
25     return <div>{items.map(...)}</div>;
26 }
```

## A.8 Altri Hooks

### A.8.1 useId

Genera ID unici stabili.

Listing A.12: useId - Reference

```
1 import { useId } from 'react';
2
```

```

3 function FormField({ label }) {
4   const id = useId();
5
6   return (
7     <div>
8       <label htmlFor={id}>{label}</label>
9       <input id={id} />
10    </div>
11  );
12 }
13
14 // Multiple IDs
15 function Form() {
16   const baseId = useId();
17
18   return (
19     <form>
20       <label htmlFor={` ${baseId}-name `}>Name</label>
21       <input id={` ${baseId}-name `} />
22
23       <label htmlFor={` ${baseId}-email `}>Email</label>
24       <input id={` ${baseId}-email `} />
25     </form>
26   );
27 }

```

### A.8.2 useDebugValue

Label custom hook in DevTools.

Listing A.13: useDebugValue - Reference

```

1 import { useDebugValue, useState } from 'react';
2
3 function useFriendStatus(friendID) {
4   const [isOnline, setIsOnline] = useState(null);
5
6   // Mostra "Online" o "Offline" in DevTools
7   useDebugValue(isOnline ? 'Online' : 'Offline');
8
9   // Con formatter (eseguito solo quando DevTools ispeziona)
10  useDebugValue(date, date => date.toDateString());
11
12  return isOnline;
13 }

```

## A.9 Custom Hooks Pattern

### A.9.1 useFetch

Listing A.14: Custom hook per fetch

```

1 function useFetch(url) {
2   const [data, setData] = useState(null);
3   const [loading, setLoading] = useState(true);
4   const [error, setError] = useState(null);
5

```

```
6   useEffect(() => {
7     const controller = new AbortController();
8
9     setLoading(true);
10
11    fetch(url, { signal: controller.signal })
12      .then(res => res.json())
13      .then(setData)
14      .catch(setError)
15      .finally(() => setLoading(false));
16
17    return () => controller.abort();
18  }, [url]);
19
20  return { data, loading, error };
21 }
```

### A.9.2 useLocalStorage

Listing A.15: Custom hook per localStorage

```
1  function useLocalStorage(key, initialValue) {
2    const [value, setValue] = useState(() => {
3      const stored = localStorage.getItem(key);
4      return stored ? JSON.parse(stored) : initialValue;
5    });
6
7    useEffect(() => {
8      localStorage.setItem(key, JSON.stringify(value));
9    }, [key, value]);
10
11    return [value, setValue];
12  }
```

### A.9.3 useToggle

Listing A.16: Custom hook per toggle

```
1  function useToggle(initialValue = false) {
2    const [value, setValue] = useState(initialValue);
3
4    const toggle = useCallback(() => {
5      setValue(v => !v);
6    }, []);
7
8    return [value, toggle];
9  }
```

### A.9.4 usePrevious

Listing A.17: Custom hook per previous value

```
1  function usePrevious(value) {
2    const ref = useRef();
3  }
```



```

4   useEffect(() => {
5     ref.current = value;
6   }, [value]);
7
8   return ref.current;
9 }

```

### A.9.5 useOnClickOutside

Listing A.18: Custom hook per click fuori elemento

```

1 function useOnClickOutside(ref, handler) {
2   useEffect(() => {
3     const listener = (event) => {
4       if (!ref.current || ref.current.contains(event.target)) {
5         return;
6       }
7       handler(event);
8     };
9
10    document.addEventListener('mousedown', listener);
11    document.addEventListener('touchstart', listener);
12
13    return () => {
14      document.removeEventListener('mousedown', listener);
15      document.removeEventListener('touchstart', listener);
16    };
17  }, [ref, handler]);
18 }

```

## A.10 Regole degli Hooks

### Regole Fondamentali

1. **Solo al Top Level:** Non chiamare hooks dentro loop, condizioni o funzioni nidificate
2. **Solo in React Functions:** Chiamare hooks solo in function components o custom hooks
3. **Ordine Consistente:** L'ordine delle chiamate agli hooks deve essere lo stesso ad ogni render

Listing A.19: Esempi di violazioni

```

1 // SBAGLIATO: Hook in condizione
2 if (condition) {
3   useEffect(() => {});
4 }
5
6 // SBAGLIATO: Hook in loop
7 for (let i = 0; i < 10; i++) {
8   useState(i);
9 }
10
11 // SBAGLIATO: Hook in funzione normale

```

```
12 function regularFunction() {  
13   const [state, setState] = useState(0);  
14 }  
15  
16 // CORRETTO: Hook al top level  
17 function Component() {  
18   const [state, setState] = useState(0);  
19  
20   useEffect(() => {  
21     if (condition) {  
22       // Logica qui  
23     }  
24   });  
25  
26   return <div>{state}</div>;  
27 }
```

## A.11 Conclusioni

Gli hooks React forniscono un'API potente e flessibile per gestire stato, effetti e logica riutilizzabile. La chiave è:

- Comprendere quando usare ciascun hook
- Rispettare le regole degli hooks
- Creare custom hooks per logica riutilizzabile
- Usare memoization solo quando necessario
- Testare hooks con renderHook

## Appendice B

# Appendice: Progetti Completi

### B.1 Introduzione

Questa appendice presenta tre progetti completi funzionanti che dimostrano best practices, pattern e tecniche avanzate di React.

### B.2 Progetto 1: Todo App Completa

Un'applicazione Todo con tutte le features moderne: filtri, local storage, editing inline e statistiche.

#### B.2.1 Struttura del Progetto

Listing B.1: Struttura directory Todo App

```
1 todo-app/  
2     src/  
3         components/  
4             TodoForm.jsx  
5             TodoList.jsx  
6             TodoItem.jsx  
7             TodoFilters.jsx  
8             TodoStats.jsx  
9         hooks/  
10             useTodos.js  
11             useLocalStorage.js  
12     App.jsx  
13     main.jsx  
14     package.json  
15     vite.config.js
```

#### B.2.2 Custom Hooks

Listing B.2: useLocalStorage hook

```
1 // hooks/useLocalStorage.js  
2 import { useState, useEffect } from 'react';  
3  
4 function useLocalStorage(key, initialValue) {  
5     const [value, setValue] = useState(() => {  
6         try {  
7             const item = window.localStorage.getItem(key);  
8             return item ? JSON.parse(item) : initialValue;  
9         }  
10    })  
11    ;  
12    ;  
13    ;  
14    ;  
15    ;
```

```

9      } catch (error) {
10        console.error('Error loading from localStorage:', error);
11        return initialValue;
12      }
13    });
14
15    useEffect(() => {
16      try {
17        window.localStorage.setItem(key, JSON.stringify(value));
18      } catch (error) {
19        console.error('Error saving to localStorage:', error);
20      }
21    }, [key, value]);
22
23    return [value, setValue];
24  }
25
26  export default useLocalStorage;

```

Listing B.3: useTodos hook

```

1  // hooks/useTodos.js
2  import { useCallback } from 'react';
3  import useLocalStorage from './useLocalStorage';
4
5  function useTodos() {
6    const [todos, setTodos] = useLocalStorage('todos', []);
7
8    const addTodo = useCallback((text) => {
9      const newTodo = {
10        id: Date.now(),
11        text,
12        completed: false,
13        createdAt: new Date().toISOString()
14      };
15      setTodos(prev => [...prev, newTodo]);
16    }, [setTodos]);
17
18    const toggleTodo = useCallback((id) => {
19      setTodos(prev =>
20        prev.map(todo =>
21          todo.id === id
22            ? { ...todo, completed: !todo.completed }
23            : todo
24        )
25      );
26    }, [setTodos]);
27
28    const deleteTodo = useCallback((id) => {
29      setTodos(prev => prev.filter(todo => todo.id !== id));
30    }, [setTodos]);
31
32    const updateTodo = useCallback((id, newText) => {
33      setTodos(prev =>
34        prev.map(todo =>
35          todo.id === id
36            ? { ...todo, text: newText }
37            : todo
38        )

```

```

39     );
40   }, [setTodos]);
41
42   const clearCompleted = useCallback(() => {
43     setTodos(prev => prev.filter(todo => !todo.completed));
44   }, [setTodos]);
45
46   return {
47     todos,
48     addTodo,
49     toggleTodo,
50     deleteTodo,
51     updateTodo,
52     clearCompleted
53   };
54 }
55
56 export default useTodos;

```

### B.2.3 Componenti

Listing B.4: TodoForm component

```

1  // components/TodoForm.jsx
2  import { useState } from 'react';
3
4  function TodoForm({ onAdd }) {
5    const [input, setInput] = useState('');
6
7    const handleSubmit = (e) => {
8      e.preventDefault();
9      if (input.trim()) {
10        onAdd(input.trim());
11        setInput('');
12      }
13    };
14
15    return (
16      <form onSubmit={handleSubmit} className="todo-form">
17        <input
18          type="text"
19          value={input}
20          onChange={(e) => setInput(e.target.value)}
21          placeholder="Cosa devi fare?"
22          className="todo-input"
23        />
24        <button type="submit" className="todo-add-btn">
25          Aggiungi
26        </button>
27      </form>
28    );
29  }
30
31  export default TodoForm;

```

Listing B.5: TodoItem component

```

1 // components/ToDoItem.jsx
2 import { useState } from 'react';
3
4 function ToDoItem({ todo, onToggle, onDelete, onUpdate }) {
5   const [isEditing, setIsEditing] = useState(false);
6   const [editText, setEditText] = useState(todo.text);
7
8   const handleSave = () => {
9     if (editText.trim()) {
10       onUpdate(todo.id, editText.trim());
11       setIsEditing(false);
12     }
13   };
14
15   const handleCancel = () => {
16     setEditText(todo.text);
17     setIsEditing(false);
18   };
19
20   if (isEditing) {
21     return (
22       <li className="todo-item editing">
23         <input
24           type="text"
25           value={editText}
26           onChange={(e) => setEditText(e.target.value)}
27           onKeyDown={(e) => {
28             if (e.key === 'Enter') handleSave();
29             if (e.key === 'Escape') handleCancel();
30           }}
31           className="todo-edit-input"
32           autoFocus
33         />
34         <button onClick={handleSave} className="save-btn">Salva</button>
35         <button onClick={handleCancel} className="cancel-btn">Annulla</button>
36       </li>
37     );
38   }
39
40   return (
41     <li className={`todo-item ${todo.completed ? 'completed' : ''}`}>
42       <input
43         type="checkbox"
44         checked={todo.completed}
45         onChange={() => onToggle(todo.id)}
46         className="todo-checkbox"
47       />
48
49       <span
50         onClick={() => setIsEditing(true)}
51         className="todo-text"
52       >
53         {todo.text}
54       </span>
55
56       <div className="todo-actions">
57         <button

```

```

58         onClick={() => setIsEditing(true)}
59         className="edit-btn"
60         title="Modifica"
61     >
62
63     </button>
64     <button
65         onClick={() => onDelete(todo.id)}
66         className="delete-btn"
67         title="Elimina"
68     >
69
70     </button>
71 </div>
72 </li>
73 );
74 }
75
76 export default TodoItem;

```

Listing B.6: TodoList component

```

1 // components/TodoList.jsx
2 import TodoItem from './TodoItem';
3
4 function TodoList({ todos, onToggle, onDelete, onUpdate }) {
5     if (todos.length === 0) {
6         return (
7             <div className="empty-state">
8                 <p>Nessun todo. Aggiungine uno!</p>
9             </div>
10        );
11    }
12
13    return (
14        <ul className="todo-list">
15            {todos.map(todo => (
16                <TodoItem
17                    key={todo.id}
18                    todo={todo}
19                    onToggle={onToggle}
20                    onDelete={onDelete}
21                    onUpdate={onUpdate}
22                />
23            ))}
24        </ul>
25    );
26 }
27
28 export default TodoList;

```

Listing B.7: TodoFilters component

```

1 // components/TodoFilters.jsx
2 function TodoFilters({ filter, setFilter }) {
3     const filters = [
4         { value: 'all', label: 'Tutti' },
5         { value: 'active', label: 'Attivi' },
6         { value: 'completed', label: 'Completati' }

```

```

7   ];
8
9   return (
10    <div className="todo-filters">
11      {filters.map(f => (
12        <button
13          key={f.value}
14          onClick={() => setFilter(f.value)}
15          className={`filter-btn ${filter === f.value ? 'active' : ''}`}
16        >
17          {f.label}
18        </button>
19      ))}
20    </div>
21  );
22 }
23
24 export default TodoFilters;

```

Listing B.8: TodoStats component

```

1  // components/TodoStats.jsx
2  function TodoStats({ todos }) {
3    const total = todos.length;
4    const completed = todos.filter(t => t.completed).length;
5    const active = total - completed;
6    const percentage = total > 0 ? Math.round((completed / total) * 100) :
7      0;
8
9    return (
10     <div className="todo-stats">
11       <div className="stat">
12         <span className="stat-label">Totale:</span>
13         <span className="stat-value">{total}</span>
14       </div>
15
16       <div className="stat">
17         <span className="stat-label">Attivi:</span>
18         <span className="stat-value">{active}</span>
19       </div>
20
21       <div className="stat">
22         <span className="stat-label">Completati:</span>
23         <span className="stat-value">{completed}</span>
24       </div>
25
26       <div className="stat">
27         <span className="stat-label">Progresso:</span>
28         <span className="stat-value">{percentage}%</span>
29       </div>
30
31       <div className="progress-bar">
32         <div
33           className="progress-fill"
34           style={{ width: `${percentage}%` }}
35         />
36       </div>
37     </div>
38   );

```



```

38 }
39
40 export default TodoStats;

```

### B.2.4 App Principale

Listing B.9: App.jsx completo

```

1  // App.jsx
2  import { useState, useMemo } from 'react';
3  import useTodos from './hooks/useTodos';
4  import TodoForm from './components/TodoForm';
5  import TodoList from './components/TodoList';
6  import TodoFilters from './components/TodoFilters';
7  import TodoStats from './components/TodoStats';
8  import './App.css';
9
10 function App() {
11   const {
12     todos,
13     addTodo,
14     toggleTodo,
15     deleteTodo,
16     updateTodo,
17     clearCompleted
18   } = useTodos();
19
20   const [filter, setFilter] = useState('all');
21
22   // Filtra todos in base al filtro selezionato
23   const filteredTodos = useMemo(() => {
24     switch (filter) {
25       case 'active':
26         return todos.filter(todo => !todo.completed);
27       case 'completed':
28         return todos.filter(todo => todo.completed);
29       default:
30         return todos;
31     }
32   }, [todos, filter]);
33
34   const hasCompleted = todos.some(todo => todo.completed);
35
36   return (
37     <div className="app">
38       <header className="app-header">
39         <h1>    Todo App</h1>
40       </header>
41
42       <main className="app-main">
43         <TodoForm onAdd={addTodo} />
44
45         <TodoStats todos={todos} />
46
47         <TodoFilters filter={filter} setFilter={setFilter} />
48
49         <TodoList

```

```

50     todos={filteredTodos}
51     onToggle={toggleTodo}
52     onDelete={deleteTodo}
53     onUpdate={updateTodo}
54   />
55
56   {hasCompleted && (
57     <button
58       onClick={clearCompleted}
59       className="clear-completed-btn"
60     >
61       Elimina Completati
62     </button>
63   )}
64 </main>
65 </div>
66 );
67 }
68
69 export default App;

```

## B.3 Progetto 2: E-commerce Product Catalog

Un catalogo prodotti e-commerce con carrello, filtri e checkout.

### B.3.1 Struttura

Listing B.10: Struttura E-commerce App

```

1 ecommerce-app/
2   src/
3     components/
4       ProductCard.jsx
5       ProductList.jsx
6       ProductFilters.jsx
7       Cart.jsx
8       CartItem.jsx
9       Checkout.jsx
10    context/
11      CartContext.jsx
12    data/
13      products.js
14    App.jsx
15    main.jsx

```

### B.3.2 Context per Carrello

Listing B.11: CartContext.jsx

```

1 // context/CartContext.jsx
2 import { createContext, useContext, useReducer } from 'react';
3
4 const CartContext = createContext();
5
6 function cartReducer(state, action) {

```

```

7   switch (action.type) {
8     case 'ADD_ITEM':
9       const existingIndex = state.items.findIndex(
10         item => item.id === action.payload.id
11       );
12
13       if (existingIndex >= 0) {
14         const newItems = [...state.items];
15         newItems[existingIndex].quantity += 1;
16         return { ...state, items: newItems };
17       }
18
19       return {
20         ...state,
21         items: [...state.items, { ...action.payload, quantity: 1 }]
22       };
23
24     case 'REMOVE_ITEM':
25       return {
26         ...state,
27         items: state.items.filter(item => item.id !== action.payload)
28       };
29
30     case 'UPDATE_QUANTITY':
31       return {
32         ...state,
33         items: state.items.map(item =>
34           item.id === action.payload.id
35             ? { ...item, quantity: action.payload.quantity }
36             : item
37         )
38       };
39
40     case 'CLEAR_CART':
41       return { ...state, items: [] };
42
43     default:
44       return state;
45   }
46 }
47
48 export function CartProvider({ children }) {
49   const [state, dispatch] = useReducer(cartReducer, { items: [] });
50
51   const addItem = (product) => {
52     dispatch({ type: 'ADD_ITEM', payload: product });
53   };
54
55   const removeItem = (productId) => {
56     dispatch({ type: 'REMOVE_ITEM', payload: productId });
57   };
58
59   const updateQuantity = (productId, quantity) => {
60     if (quantity <= 0) {
61       removeItem(productId);
62     } else {
63       dispatch({ type: 'UPDATE_QUANTITY', payload: { id: productId,
64         quantity } });

```

```

64     }
65   };
66
67   const clearCart = () => {
68     dispatch({ type: 'CLEAR_CART' });
69   };
70
71   const totalItems = state.items.reduce((sum, item) => sum + item.
72     quantity, 0);
73
74   const totalPrice = state.items.reduce(
75     (sum, item) => sum + (item.price * item.quantity),
76     0
77   );
78
79   return (
80     <CartContext.Provider
81       value={{
82         items: state.items,
83         addItem,
84         removeItem,
85         updateQuantity,
86         clearCart,
87         totalItems,
88         totalPrice
89       }}
90     >
91     {children}
92   </CartContext.Provider>
93   );
94 }
95
96 export function useCart() {
97   const context = useContext(CartContext);
98   if (!context) {
99     throw new Error('useCart must be used within CartProvider');
100   }
101   return context;
102 }

```

### B.3.3 Componenti Prodotto

Listing B.12: ProductCard component

```

1  // components/ProductCard.jsx
2  import { useCart } from '../context/CartContext';
3
4  function ProductCard({ product }) {
5    const { addItem } = useCart();
6
7    return (
8      <div className="product-card">
9        <img
10          src={product.image}
11          alt={product.name}
12          className="product-image"
13        />

```

```

14
15     <div className="product-info">
16       <h3 className="product-name">{product.name}</h3>
17       <p className="product-category">{product.category}</p>
18       <p className="product-description">{product.description}</p>
19
20       <div className="product-footer">
21         <span className="product-price">{product.price.toFixed(2)}
22           </span>
23
24         <button
25           onClick={() => addItem(product)}
26           className="add-to-cart-btn"
27         >
28           Aggiungi al Carrello
29         </button>
30       </div>
31     </div>
32   );
33 }
34
35 export default ProductCard;

```

Listing B.13: ProductList component

```

1 // components/ProductList.jsx
2 import { useState, useMemo } from 'react';
3 import ProductCard from './ProductCard';
4 import ProductFilters from './ProductFilters';
5
6 function ProductList({ products }) {
7   const [searchTerm, setSearchTerm] = useState('');
8   const [category, setCategory] = useState('all');
9   const [sortBy, setSortBy] = useState('name');
10
11   // Filtra e ordina prodotti
12   const filteredProducts = useMemo(() => {
13     let result = products;
14
15     // Filtra per ricerca
16     if (searchTerm) {
17       result = result.filter(p =>
18         p.name.toLowerCase().includes(searchTerm.toLowerCase())
19       );
20     }
21
22     // Filtra per categoria
23     if (category !== 'all') {
24       result = result.filter(p => p.category === category);
25     }
26
27     // Ordina
28     result = [...result].sort((a, b) => {
29       switch (sortBy) {
30         case 'price-asc':
31           return a.price - b.price;
32         case 'price-desc':
33           return b.price - a.price;

```

```

34         case 'name':
35         default:
36             return a.name.localeCompare(b.name);
37     }
38     });
39
40     return result;
41 }, [products, searchTerm, category, sortBy]);
42
43 return (
44     <div className="product-list-container">
45         <ProductFilters
46             searchTerm={searchTerm}
47             setSearchTerm={setSearchTerm}
48             category={category}
49             setCategory={setCategory}
50             sortBy={sortBy}
51             setSortBy={setSortBy}
52         />
53
54         <div className="product-grid">
55             {filteredProducts.length === 0 ? (
56                 <p>Nessun prodotto trovato</p>
57             ) : (
58                 filteredProducts.map(product => (
59                     <ProductCard key={product.id} product={product} />
60                 ))
61             )}
62         </div>
63     </div>
64 );
65 }
66
67 export default ProductList;

```

Listing B.14: Cart component

```

1  // components/Cart.jsx
2  import { useCart } from '../context/CartContext';
3  import CartItem from './CartItem';
4
5  function Cart({ isOpen, onClose }) {
6      const { items, totalItems, totalPrice, clearCart } = useCart();
7
8      if (!isOpen) return null;
9
10     return (
11         <div className="cart-overlay" onClick={onClose}>
12             <div className="cart-sidebar" onClick={(e) => e.stopPropagation()}>
13                 <div className="cart-header">
14                     <h2>Carrello ({totalItems})</h2>
15                     <button onClick={onClose} className="close-btn"> </button>
16                 </div>
17
18                 <div className="cart-items">
19                     {items.length === 0 ? (
20                         <p className="empty-cart">Il carrello è vuoto</p>
21                     ) : (

```

```

22         items.map(item => (
23             <CartItem key={item.id} item={item} />
24         ))
25     )}
26 </div>
27
28     {items.length > 0 && (
29         <div className="cart-footer">
30             <div className="cart-total">
31                 <span>Totale:</span>
32                 <span className="total-price">{totalPrice.toFixed(2)} </
                    span>
33             </div>
34
35             <button className="checkout-btn">
36                 Procedi al Checkout
37             </button>
38
39             <button onClick={clearCart} className="clear-cart-btn">
40                 Svuota Carrello
41             </button>
42         </div>
43     )}
44 </div>
45 </div>
46 );
47 }
48
49 export default Cart;

```

### B.3.4 App Principale E-commerce

Listing B.15: App E-commerce completo

```

1 // App.jsx
2 import { useState } from 'react';
3 import { CartProvider } from '../context/CartContext';
4 import ProductList from '../components/ProductList';
5 import Cart from '../components/Cart';
6 import { products } from '../data/products';
7 import './App.css';
8
9 function App() {
10     const [cartOpen, setCartOpen] = useState(false);
11
12     return (
13         <CartProvider>
14             <div className="app">
15                 <header className="app-header">
16                     <h1> E-Shop </h1>
17
18                     <button
19                         onClick={() => setCartOpen(true)}
20                         className="cart-icon-btn"
21                     >
22                         Carrello
23                 </button>

```

```

24     </header>
25
26     <main className="app-main">
27       <ProductList products={products} />
28     </main>
29
30     <Cart isOpen={cartOpen} onClose={() => setCartOpen(false)} />
31   </div>
32 </CartProvider>
33 );
34 }
35
36 export default App;

```

## B.4 Progetto 3: Dashboard Analytics

Una dashboard con grafici, metriche e data fetching.

### B.4.1 Struttura Dashboard

Listing B.16: Struttura Dashboard

```

1 dashboard-app/
2   src/
3     components/
4       MetricCard.jsx
5       Chart.jsx
6       RecentActivity.jsx
7       UserTable.jsx
8       Sidebar.jsx
9     hooks/
10      useAnalytics.js
11     App.jsx
12     main.jsx

```

### B.4.2 Hook per Analytics

Listing B.17: useAnalytics hook

```

1 // hooks/useAnalytics.js
2 import { useState, useEffect } from 'react';
3
4 function useAnalytics() {
5   const [data, setData] = useState(null);
6   const [loading, setLoading] = useState(true);
7   const [error, setError] = useState(null);
8
9   useEffect(() => {
10     const fetchAnalytics = async () => {
11       try {
12         setLoading(true);
13
14         // Simula API call
15         await new Promise(resolve => setTimeout(resolve, 1000));
16

```



```

17     const mockData = {
18       metrics: {
19         totalUsers: 15420,
20         activeUsers: 8732,
21         revenue: 245689,
22         conversion: 3.2
23       },
24       chartData: Array.from({ length: 12 }, (_, i) => ({
25         month: ['Gen', 'Feb', 'Mar', 'Apr', 'Mag', 'Giu',
26               'Lug', 'Ago', 'Set', 'Ott', 'Nov', 'Dic'][i],
27         value: Math.floor(Math.random() * 5000) + 2000
28       })),
29       recentActivity: [
30         { id: 1, user: 'Mario Rossi', action: 'Nuovo ordine', time:
31           '2 min fa' },
32         { id: 2, user: 'Laura Bianchi', action: 'Registrazione',
33           time: '15 min fa' },
34         { id: 3, user: 'Giuseppe Verdi', action: 'Pagamento', time:
35           '1 ora fa' }
36       ]
37     };
38
39     setData(mockData);
40   } catch (err) {
41     setError(err.message);
42   } finally {
43     setLoading(false);
44   }
45 };
46
47 fetchAnalytics();
48
49 // Refresh ogni 30 secondi
50 const interval = setInterval(fetchAnalytics, 30000);
51
52 return () => clearInterval(interval);
53 }, []);
54
55 return { data, loading, error };
56 }
57
58 export default useAnalytics;

```

### B.4.3 Dashboard App Completo

Listing B.18: Dashboard App.jsx

```

1 // App.jsx
2 import useAnalytics from '../hooks/useAnalytics';
3 import MetricCard from '../components/MetricCard';
4 import Chart from '../components/Chart';
5 import RecentActivity from '../components/RecentActivity';
6
7 function App() {
8   const { data, loading, error } = useAnalytics();
9
10   if (loading) {

```

```

11     return (
12         <div className="loading-container">
13             <div className="spinner"></div>
14             <p>Caricamento dashboard...</p>
15         </div>
16     );
17 }
18
19 if (error) {
20     return (
21         <div className="error-container">
22             <p>Errore: {error}</p>
23         </div>
24     );
25 }
26
27 return (
28     <div className="dashboard">
29         <header className="dashboard-header">
30             <h1> Analytics Dashboard </h1>
31         </header>
32
33         <main className="dashboard-main">
34             <section className="metrics-grid">
35                 <MetricCard
36                     title="Utenti Totali"
37                     value={data.metrics.totalUsers.toLocaleString()}
38                     icon=""
39                     trend="+12%"
40                 />
41
42                 <MetricCard
43                     title="Utenti Attivi"
44                     value={data.metrics.activeUsers.toLocaleString()}
45                     icon=""
46                     trend="+8%"
47                 />
48
49                 <MetricCard
50                     title="Revenue"
51                     value={`$ ${(data.metrics.revenue / 1000).toFixed(1)}K`}
52                     icon=""
53                     trend="+23%"
54                 />
55
56                 <MetricCard
57                     title="Conversione"
58                     value={`${data.metrics.conversion}%`}
59                     icon=""
60                     trend="+0.5%"
61                 />
62             </section>
63
64             <section className="chart-section">
65                 <Chart data={data.chartData} />
66             </section>
67
68             <section className="activity-section">

```

```
69         <RecentActivity activities={data.recentActivity} />
70     </section>
71 </main>
72 </div>
73 );
74 }
75
76 export default App;
```

## B.5 Best Practices Implementate

### Best Practices nei Progetti

- **Separazione delle Responsabilità:** Componenti piccoli e focalizzati
- **Custom Hooks:** Logica riutilizzabile estratta in hooks
- **Context API:** State management per dati globali (carrello)
- **Memoization:** useMemo per calcoli costosi e filtri
- **Local Storage:** Persistenza dati (todo app)
- **Error Handling:** Gestione errori in data fetching
- **Loading States:** Feedback visivo durante operazioni async
- **Accessibility:** Uso corretto di semantica HTML
- **Performance:** Ottimizzazioni con React.memo dove necessario
- **Code Organization:** Struttura chiara e manutenibile

## B.6 Deployment

### B.6.1 Build per Produzione

Listing B.19: Build e deployment

```
1 # Build con Vite
2 npm run build
3
4 # Preview build locale
5 npm run preview
6
7 # Deploy su Netlify
8 netlify deploy --prod --dir=dist
9
10 # Deploy su Vercel
11 vercel --prod
12
13 # Deploy su GitHub Pages
14 npm run build
15 git add dist -f
16 git commit -m "Deploy"
17 git subtree push --prefix dist origin gh-pages
```

## B.7 Conclusioni

Questi tre progetti dimostrano applicazioni React complete e funzionanti con:

- **Todo App:** State management locale, custom hooks, local storage
- **E-commerce:** Context API, reducer pattern, carrello funzionale
- **Dashboard:** Data fetching, visualizzazione dati, analytics

Studiare e modificare questi progetti è il modo migliore per imparare React in modo pratico e approfondito.