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

```
// Codice jQuery per aggiornare l'interfaccia
$('#counter').text('0');

$('#increment').click(function() {
    var currentValue = parseInt($('#counter').text());
    $('#counter').text(currentValue + 1);

    // Gestione manuale della logica di visualizzazione
    if (currentValue + 1 > 10) {
        $('#counter').addClass('high-value');
    }

    // Aggiornamento di altri elementi correlati
    $('#status').text('Counter incrementato');
});
```

2. Template Engine con Two-Way Binding:

Listing 1.2: Approccio AngularJS

```
// AngularJS controller
app.controller('CounterController', function($scope) {
    $scope.counter = 0;

    $scope.increment = function() {
        $scope.counter++;
    };
});
```

La Soluzione React:

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

Listing 1.3: Approccio React moderno

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

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

    return (
        <div>
            <p className={count > 10 ? 'high-value' : ''}>
                Contatore: {count}
            </p>
            <button onClick={() => setCount(count + 1)}>
                Incrementa
            </button>
            <p>Stato: Counter {count > 0 ? 'incrementato' : 'iniziale'
              }</p>
        </div>
    );
}
```

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

```
// Componente Button riutilizzabile
function Button({ onClick, children, variant = 'primary' }) {
    return (
        <button
            className={`btn btn-${variant}`}
            onClick={onClick}
        >
            {children}
        </button>
    );
}

// Componente Card riutilizzabile
function Card({ title, description, onAction }) {
    return (
        <div className="card">
            <h3>{title}</h3>
            <p>{description}</p>
            <Button onClick={onAction} variant="secondary">
                Leggi di più
            </Button>
        </div>
    );
}

// Composizione per creare una dashboard
function Dashboard() {
    const handleCardAction = (id) => {
        console.log(`Azione su card ${id}`);
    };

    return (
        <div className="dashboard">
            <h1>Dashboard</h1>
            <div className="cards-grid">
                <Card
                    title="Vendite"
                    description="Visualizza le vendite mensili"
                    onAction={() => handleCardAction('sales')}
                />
                <Card
```

```
                    title="Utenti"
                    description="Gestisci gli utenti registrati"
                    onAction={() => handleCardAction('users')}
                />
                <Card
                    title="Analytics"
                    description="Analizza le metriche"
                    onAction={() => handleCardAction('analytics')}
                />
            </div>
        </div>
    );
}
```

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

```
function TodoList() {
    const [todos, setTodos] = useState([
        { id: 1, text: 'Imparare React', completed: false },
        { id: 2, text: 'Costruire un progetto', completed: false },
        { id: 3, text: 'Deploy in produzione', completed: false }
    ]);

    // Quando toggli un todo, React:
    // 1. Crea un nuovo Virtual DOM
    // 2. Confronta con il precedente
    // 3. Aggiorna solo il singolo elemento modificato nel DOM reale
    const toggleTodo = (id) => {
        setTodos(todos.map(todo =>
            todo.id === id
                ? { ...todo, completed: !todo.completed }
                : todo
        ));
    };

    return (
        <ul>
            {todos.map(todo => (
                <li
                    key={todo.id}
                    onClick={() => toggleTodo(todo.id)}
                    style={{
                        textDecoration: todo.completed ? 'line-through'
                            : 'none'
                    }}
                >
                    {todo.text}
                </li>
            ))}
        </ul>
    );
}
```

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)

```
// Descrive COME fare qualcosa
const form = document.getElementById('login-form');
const errorDiv = document.getElementById('error');
const submitBtn = document.getElementById('submit');

form.addEventListener('submit', function(e) {
    e.preventDefault();

    // Rimuovi errori precedenti
    errorDiv.innerHTML = '';
    errorDiv.classList.remove('visible');

    // Disabilita il bottone
    submitBtn.disabled = true;
    submitBtn.textContent = 'Caricamento...';

    // Fai la richiesta
    fetch('/api/login', { method: 'POST', body: new FormData(form) })
        .then(response => {
            if (!response.ok) {
                // Mostra errore
                errorDiv.textContent = 'Login fallito';
                errorDiv.classList.add('visible');
                // Riabilita bottone
                submitBtn.disabled = false;
                submitBtn.textContent = 'Login';
            } else {
                // Redirect
                window.location.href = '/dashboard';
            }
        });
});
```

Listing 1.7: Approccio Dichiarativo (React)

```
// Descrive COSA mostrare
function LoginForm() {
    const [isLoading, setIsLoading] = useState(false);
    const [error, setError] = useState(null);

    const handleSubmit = async (e) => {
        e.preventDefault();
        setIsLoading(true);
        setError(null);

        try {
            const response = await fetch('/api/login', {
                method: 'POST',
                body: new FormData(e.target)
            });

            if (!response.ok) {
                setError('Login fallito');
            } else {
                window.location.href = '/dashboard';
            }
        } finally {
            setIsLoading(false);
        }
    };

    // React si occupa di aggiornare il DOM basandosi sullo stato
    return (
        <form onSubmit={handleSubmit}>
            {error && <div className="error visible">{error}</div>}

            <input type="email" name="email" required />
            <input type="password" name="password" required />

            <button type="submit" disabled={isLoading}>
                {isLoading ? 'Caricamento...' : 'Login'}
            </button>
        </form>
    );
}
```

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

```
// I dati fluiscono sempre dall'alto verso il basso
function App() {
    const [user, setUser] = useState({
        name: 'Mario Rossi',
        role: 'Admin'
    });

    // App passa dati ai componenti figli
    return (
        <div>
            <Header user={user} />
            <Dashboard user={user} onUpdateUser={setUser} />
        </div>
    );
}

function Header({ user }) {
    // Header riceve dati e li visualizza
    // Non può modificarli direttamente
    return (
        <header>
            <h1>Benvenuto, {user.name}</h1>
            <span>Ruolo: {user.role}</span>
        </header>
    );
}

function Dashboard({ user, onUpdateUser }) {
    // Dashboard può richiedere modifiche tramite callback
    const handleRoleChange = () => {
        onUpdateUser({
            ...user,
            role: user.role === 'Admin' ? 'User' : 'Admin'
        });
    };

    return (
        <div>
            <p>Dashboard per {user.name}</p>
            <button onClick={handleRoleChange}>
                Cambia Ruolo
            </button>
        </div>
    );
}
```

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

```
// components/Button.js - Funziona sia su web che mobile
import { TouchableOpacity, Text } from 'react-native'; // Mobile
// import './Button.css'; // Web

function Button({ onPress, title, style }) {
    // Logica condivisa
    const handlePress = () => {
        console.log('Button pressed');
        onPress?.();
    };

    // React Native (Mobile)
    return (
        <TouchableOpacity
            onPress={handlePress}
            style={[styles.button, style]}
        >
            <Text style={styles.text}>{title}</Text>
        </TouchableOpacity>
    );

    // React DOM (Web)
    // return (
    //     <button onClick={handlePress} className="button" style={style
    }>
    //         {title}
    //     </button>
    // );
}
```

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

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';

function App() {
    return (
        <BrowserRouter>
            <Navigation />
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/products" element={<Products />} />
                <Route path="/products/:id" element={<ProductDetail />}
                    />
                <Route path="/cart" element={<Cart />} />
                <Route path="/checkout" element={<Checkout />} />
            </Routes>
        </BrowserRouter>
    );
}
```

2. **Dashboard e Admin Panels:**
   Interfacce con molti componenti interattivi e aggiornamenti in tempo reale.

Listing 1.11: Dashboard interattiva

```
function AdminDashboard() {
    const [stats, setStats] = useState(null);
    const [liveUsers, setLiveUsers] = useState([]);

    // Polling per aggiornamenti in tempo reale
    useEffect(() => {
        const interval = setInterval(async () => {
            const data = await fetchLiveStats();
            setStats(data.stats);
            setLiveUsers(data.users);
        }, 5000);

        return () => clearInterval(interval);
    }, []);

    return (
        <div className="dashboard">
            <StatsCards stats={stats} />
            <LiveUsersTable users={liveUsers} />
            <SalesChart data={stats?.sales} />
            <RecentOrders limit={10} />
        </div>
    );
}
```

3. **Applicazioni con UI Complesse:**
   Interfacce con molti stati e interazioni.

Listing 1.12: UI complessa con drag and drop

```
import { DndContext, closestCenter } from '@dnd-kit/core';

function KanbanBoard() {
    const [columns, setColumns] = useState({
        todo: ['Task 1', 'Task 2'],
        inProgress: ['Task 3'],
        done: ['Task 4', 'Task 5']
    });

    const handleDragEnd = (event) => {
        const { active, over } = event;

        if (!over) return;

        // Logica per spostare task tra colonne
        const sourceColumn = findColumn(active.id);
        const destColumn = over.id;

        moveTask(active.id, sourceColumn, destColumn);
    };

    return (
        <DndContext onDragEnd={handleDragEnd} collisionDetection={
            closestCenter}>
            <div className="kanban">
                <Column id="todo" title="To Do" tasks={columns.todo} />
                <Column id="inProgress" title="In Progress" tasks={
                    columns.inProgress} />
                <Column id="done" title="Done" tasks={columns.done} />
            </div>
        </DndContext>
    );
}
```

4. **Applicazioni Real-Time:**
Chat, collaboration tools, live updates.

Listing 1.13: Chat real-time con WebSocket

```
function ChatRoom({ roomId }) {
    const [messages, setMessages] = useState([]);
    const [ws, setWs] = useState(null);

    useEffect(() => {
        const websocket = new WebSocket(`ws://api.example.com/chat/${
            roomId}`);

        websocket.onmessage = (event) => {
            const message = JSON.parse(event.data);
            setMessages(prev => [...prev, message]);
        };

        setWs(websocket);

        return () => websocket.close();
    }, [roomId]);

    const sendMessage = (text) => {
        ws.send(JSON.stringify({ text, roomId }));
    };

    return (
        <div className="chat">
            <MessageList messages={messages} />
            <MessageInput onSend={sendMessage} />
        </div>
    );
}
```

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

```
// app/products/page.js - Server Component (Next.js 13+)
async function ProductsPage() {
    // Questo codice gira SOLO sul server
    // Non aumenta il bundle JavaScript del client
    const products = await db.products.findMany();

    return (
        <div>
            <h1>I Nostri Prodotti</h1>
            <ProductList products={products} />
        </div>
    );
}

// components/ProductList.js - Client Component
'use client'; // Direttiva per Client Component

import { useState } from 'react';

function ProductList({ products }) {
    const [filter, setFilter] = useState('all');

    // Questo codice gira sul client e può usare interattività
    return (
        <div>
            <FilterBar value={filter} onChange={setFilter} />
            {products
```

```
                .filter(p => filter === 'all' || p.category === filter)
                .map(product => (
                    <ProductCard key={product.id} product={product} />
                ))
            }
        </div>
    );
}
```

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

```
import { useTransition, useDeferredValue } from 'react';

function SearchPage() {
    const [query, setQuery] = useState('');
    const [isPending, startTransition] = useTransition();

    // Gli aggiornamenti nella transition sono interrompibili
    const handleChange = (e) => {
        startTransition(() => {
            setQuery(e.target.value);
        });
    };

    return (
        <div>
            <input onChange={handleChange} />
            {isPending && <Spinner />}
            <SearchResults query={query} />
        </div>
    );
}
```

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

```javascript
// 1. Arrow Functions
const greet = (name) => `Hello, ${name}`;
const add = (a, b) => a + b;

// 2. Destructuring
const user = { name: 'Mario', age: 30 };
const { name, age } = user;

const numbers = [1, 2, 3];
const [first, second] = numbers;

// 3. Spread Operator
const newUser = { ...user, city: 'Roma' };
const newNumbers = [...numbers, 4, 5];

// 4. Template Literals
const message = `${name} ha ${age} anni`;

// 5. Array Methods
const doubled = numbers.map(n => n * 2);
const evens = numbers.filter(n => n % 2 === 0);
const sum = numbers.reduce((acc, n) => acc + n, 0);

// 6. Async/Await
const fetchData = async () => {
    try {
        const response = await fetch('/api/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error(error);
    }
};

// 7. Modules
export const API_URL = 'https://api.example.com';
export default function Component() { /* ... */ }

// 8. Optional Chaining
const userName = user?.profile?.name ?? 'Guest';

// 9. Nullish Coalescing
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...

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

function Welcome() {
    const [started, setStarted] = useState(false);

    return (
        <div className="welcome">
            <h1>Benvenuto nel Mondo di React!</h1>
            {!started ? (
                <button onClick={() => setStarted(true)}>
                    Inizia il Viaggio
                </button>
            ) : (
                <p>Sei pronto a costruire cose incredibili!     </p>
            )}
        </div>
    );
}

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

```
// React fornisce
import { useState } from 'react';

function App() {
    return <div>Hello React</div>;
}

// Tu scegli il resto:
// - Routing: React Router, TanStack Router, etc.
// - State: Redux, Zustand, MobX, etc.
// - Data fetching: TanStack Query, SWR, etc.
// - Forms: React Hook Form, Formik, etc.
```

2. Component-Based

Tutto in React è un componente riutilizzabile:

Listing 2.2: Componenti come building blocks

```
// Componente Button
function Button({ children, onClick }) {
    return <button onClick={onClick}>{children}</button>;
}

// Componente Form che usa Button
function LoginForm() {
    return (
        <form>
            <input type="email" />
            <input type="password" />
            <Button onClick={() => console.log('Login')}>
```

```
                Accedi
            </Button>
        </form>
    );
}
```

3. Dichiarativo
Descrivi cosa vuoi vedere, React si occupa del resto:

Listing 2.3: Codice dichiarativo

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

    // Dichiari COSA mostrare in base allo stato
    return (
        <div>
            <p>Hai cliccato {count} volte</p>
            <button onClick={() => setCount(count + 1)}>
                Incrementa
            </button>
        </div>
    );
    // React aggiorna il DOM automaticamente quando count cambia
}
```

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

```
// JSX
<div className="container">
    <h1>Titolo</h1>
    <p>Paragrafo</p>
</div>

// Viene trasformato in Virtual DOM object
{
    type: 'div',
    props: {
        className: 'container',
        children: [
            {
                type: 'h1',
                props: {
                    children: 'Titolo'
                }
            },
            {
                type: 'p',
                props: {
                    children: 'Paragrafo'
                }
            }
        ]
    }
}
```

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

```
function App() {
    const [items, setItems] = useState(['A', 'B', 'C']);

    return (
        <ul>
            {items.map(item => <li key={item}>{item}</li>)}
        </ul>
    );
}

// React crea Virtual DOM:
{
    type: 'ul',
    props: {
        children: [
            { type: 'li', props: { children: 'A', key: 'A' } },
            { type: 'li', props: { children: 'B', key: 'B' } },
            { type: 'li', props: { children: 'C', key: 'C' } }
        ]
    }
}

// React crea il DOM reale basandosi sul Virtual DOM
```

Fase 2: Update

Listing 2.6: Aggiornamento stato

```
// Quando lo stato cambia (aggiungiamo 'D')
setItems(['A', 'B', 'C', 'D']);

// React crea un NUOVO Virtual DOM tree:
{
    type: 'ul',
    props: {
        children: [
            { type: 'li', props: { children: 'A', key: 'A' } },
            { type: 'li', props: { children: 'B', key: 'B' } },
            { type: 'li', props: { children: 'C', key: 'C' } },
            { type: 'li', props: { children: 'D', key: 'D' } } // NUOVO
        ]
    }
}
```

Fase 3: Diffing
React confronta il nuovo Virtual DOM con quello precedente:

Listing 2.7: Algoritmo di Diffing

```
// Pseudo-codice del diffing
function diff(oldVTree, newVTree) {
    // Confronta tipo
    if (oldVTree.type !== newVTree.type) {
        return { action: 'REPLACE', node: newVTree };
    }

    // Confronta props
    const propsDiff = diffProps(oldVTree.props, newVTree.props);

    // Confronta children ricorsivamente
    const childrenDiff = diffChildren(
        oldVTree.props.children,
        newVTree.props.children
    );

    return {
        action: 'UPDATE',
        propChanges: propsDiff,
        childChanges: childrenDiff
    };
}

// Nel nostro esempio, React identifica:
// - 3 elementi uguali (A, B, C)
// - 1 elemento nuovo (D) da aggiungere
```

Fase 4: Reconciliation
React applica solo le modifiche necessarie al DOM reale:

Listing 2.8: Applicazione modifiche

```
// Invece di ricreare tutta la lista:
//     ul.innerHTML = '<li>A</li><li>B</li><li>C</li><li>D</li>';

// React fa solo:
//     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

```
//      Manipolazione diretta del DOM (LENTA)
function updateList(items) {
    const ul = document.getElementById('list');
    ul.innerHTML = ''; // Rimuove tutto

    items.forEach(item => {
        const li = document.createElement('li');
        li.textContent = item;
        ul.appendChild(li); // Ogni append causa reflow
    });
}

//      Con React Virtual DOM (VELOCE)
function List({ items }) {
    return (
        <ul>
            {items.map(item => <li key={item}>{item}</li>)}
        </ul>
    );
    // React aggiorna solo gli elementi modificati
}
```

2. Batch Updates

React raggruppa più aggiornamenti:

Listing 2.10: Batching automatico

```
function MultiUpdate() {
    const [count, setCount] = useState(0);
    const [flag, setFlag] = useState(false);

    const handleClick = () => {
        setCount(count + 1);  // Update 1
        setFlag(!flag);       // Update 2
        // React raggruppa entrambi e fa un solo re-render
    };

    console.log('Render!'); // Viene chiamato UNA sola volta

    return (
        <div>
            <p>Count: {count}</p>
            <p>Flag: {flag.toString()}</p>
            <button onClick={handleClick}>Update</button>
        </div>
    );
}
```

3. Prevedibilità
Il rendering diventa deterministico:

Listing 2.11: Rendering prevedibile

```
// Stesso stato = stesso output SEMPRE
function UserCard({ user }) {
    return (
        <div>
            <h3>{user.name}</h3>
            <p>{user.email}</p>
        </div>
    );
}

// Se user è identico, output identico
// Facile da testare e debuggare
```

2.3 React vs Altri Framework

2.3.1 React vs Angular

Angular

Listing 2.12: Componente Angular

```
// user.component.ts
import { Component, Input } from '@angular/core';

@Component({
    selector: 'app-user',
    template: `
        <div class="user-card">
            <h3>{{ user.name }}</h3>
            <p>{{ user.email }}</p>
            <button (click)="onEdit()">Edit</button>
        </div>
    `,
    styleUrls: ['./user.component.css']
})
export class UserComponent {
    @Input() user: User;

    onEdit() {
        console.log('Editing', this.user);
    }
}
```

React

Listing 2.13: Componente React equivalente

```jsx
// UserCard.jsx
import './UserCard.css';

function UserCard({ user }) {
    const handleEdit = () => {
        console.log('Editing', user);
    };

    return (
        <div className="user-card">
            <h3>{user.name}</h3>
            <p>{user.email}</p>
            <button onClick={handleEdit}>Edit</button>
        </div>
    );
}

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

```vue
<!-- UserCard.vue -->
<template>
    <div class="user-card">
        <h3>{{ user.name }}</h3>
        <p>{{ user.email }}</p>
        <button @click="handleEdit">Edit</button>
    </div>
</template>

<script setup>
import { defineProps } from 'vue';

const props = defineProps({
    user: Object
});

```

```
const handleEdit = () => {
    console.log('Editing', props.user);
};
</script>

<style scoped>
.user-card {
    border: 1px solid #ccc;
}
</style>
```

React

Listing 2.15: Componente React equivalente

```
// UserCard.jsx
import styles from './UserCard.module.css';

function UserCard({ user }) {
    const handleEdit = () => {
        console.log('Editing', user);
    };

    return (
        <div className={styles.userCard}>
            <h3>{user.name}</h3>
            <p>{user.email}</p>
            <button onClick={handleEdit}>Edit</button>
        </div>
    );
}

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

```
# Node.js (versione 18 o superiore)
node --version
# Output: v20.10.0

# npm (viene installato con Node.js)
npm --version
# Output: 10.2.3

# Git (opzionale ma raccomandato)
git --version
# 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

```
# Crea nuovo progetto
npm create vite@latest my-react-app -- --template react

# Entra nella directory
cd my-react-app

# Installa dipendenze
npm install

# Avvia development server
npm run dev
# 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

```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
)
```

Listing 2.19: App.jsx - Componente principale

```jsx
import { useState } from 'react'
import './App.css'

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

    return (
        <div className="App">
            <h1>Vite + React</h1>
            <div className="card">
                <button onClick={() => setCount((count) => count + 1)}>
                    count is {count}
                </button>
            </div>
        </div>
    )
}

export default App
```

Listing 2.20: vite.config.js - Configurazione

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

```

export default defineConfig({
    plugins: [react()],
    server: {
        port: 3000, // Cambia porta se necessario
        open: true  // Apri browser automaticamente
    }
})
```

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

```
# Crea nuovo progetto
npx create-react-app my-app

# Entra nella directory
cd my-app

# Avvia development server
npm start
# 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

```
# Crea directory e inizializza progetto
mkdir my-react-app
cd my-react-app
npm init -y

# Installa React e ReactDOM
npm install react react-dom

# Installa Vite come dev dependency
npm install -D vite @vitejs/plugin-react

# Crea struttura
mkdir src public
touch src/main.jsx src/App.jsx index.html vite.config.js
```

Listing 2.23: index.html

```html
<!DOCTYPE html>
<html lang="it">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0
       ">
    <title>My React App</title>
</head>
<body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

Listing 2.24: package.json - Aggiungi scripts

```json
{
    "name": "my-react-app",
    "version": "1.0.0",
    "scripts": {
        "dev": "vite",
        "build": "vite build",
        "preview": "vite preview"
    },
    "dependencies": {
        "react": "^18.2.0",
        "react-dom": "^18.2.0"
    },
    "devDependencies": {
        "vite": "^5.0.0",
        "@vitejs/plugin-react": "^4.2.0"
    }
}
```

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

```bash
npx create-next-app@latest my-next-app

# Scegli opzioni:
#     TypeScript? No/Yes
#     ESLint? Yes
#     Tailwind CSS? No/Yes
#     App Router? Yes (raccomandato)
#     Customize import alias? No

cd my-next-app
npm run dev
```

Remix:

Listing 2.26: Setup Remix

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

```
cd my-remix-app
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

```
npm create vite@latest react-todo -- --template react
cd react-todo
npm install
npm run dev
```

2.5.1 App.jsx - TODO List Completa

Listing 2.28: Applicazione TODO completa

```
import { useState } from 'react';
import './App.css';

function App() {
    const [todos, setTodos] = useState([
        { id: 1, text: 'Imparare React', completed: false },
        { id: 2, text: 'Costruire un progetto', completed: false }
    ]);
    const [inputValue, setInputValue] = useState('');

    // Aggiungi nuovo todo
    const addTodo = () => {
        if (inputValue.trim() === '') return;

        const newTodo = {
            id: Date.now(),
            text: inputValue,
            completed: false
        };

        setTodos([...todos, newTodo]);
        setInputValue('');
    };

    // Toggle completamento
    const toggleTodo = (id) => {
        setTodos(todos.map(todo =>
            todo.id === id
                ? { ...todo, completed: !todo.completed }
                : todo
        ));
    };

    // Elimina todo
    const deleteTodo = (id) => {
        setTodos(todos.filter(todo => todo.id !== id));
    };

    // Gestisci Enter key
    const handleKeyPress = (e) => {
        if (e.key === 'Enter') {
            addTodo();
        }
    };

    // Statistiche
    const totalTodos = todos.length;
    const completedTodos = todos.filter(t => t.completed).length;
    const pendingTodos = totalTodos - completedTodos;

    return (
        <div className="app">
            <div className="container">
                <h1>    My TODO List</h1>

                {/* Statistiche */}
                <div className="stats">
                    <span>Totale: {totalTodos}</span>
                    <span>Completati: {completedTodos}</span>
                    <span>Da fare: {pendingTodos}</span>
                </div>

                {/* Input nuovo todo */}
                <div className="input-section">
                    <input
                        type="text"
                        value={inputValue}
                        onChange={(e) => setInputValue(e.target.value)}
                        onKeyPress={handleKeyPress}
                        placeholder="Aggiungi un nuovo task..."
                    />
                    <button onClick={addTodo}>Aggiungi</button>
                </div>

                {/* Lista todos */}
                <ul className="todo-list">
                    {todos.length === 0 ? (
                        <li className="empty">Nessun task. Aggiungine uno!</li>
                    ) : (
                        todos.map(todo => (
                            <li
                                key={todo.id}
                                className={todo.completed ? 'completed' : ''}
                            >
                                <input
                                    type="checkbox"
                                    checked={todo.completed}
                                    onChange={() => toggleTodo(todo.id)}
                                />
                                <span onClick={() => toggleTodo(todo.id)}>
                                    {todo.text}
                                </span>
                                <button
                                    className="delete-btn"
                                    onClick={() => deleteTodo(todo.id)}
                                >

                                </button>
                            </li>
                        ))
                    )}
                </ul>

                {/* Clear completed */}
                {completedTodos > 0 && (
                    <button
                        className="clear-completed"
                        onClick={() => setTodos(todos.filter(t => !t.
                            completed))}
                    >
                        Rimuovi completati ({completedTodos})
                    </button>
                )}
            </div>
        </div>
    );
}

export default App;
```

2.5.2 App.css - Styling

Listing 2.29: Stili per TODO app

```
.app {
    min-height: 100vh;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 20px;
}

.container {
    background: white;
    border-radius: 15px;
    padding: 30px;
    max-width: 500px;
    width: 100%;
    box-shadow: 0 10px 40px rgba(0,0,0,0.2);
}

h1 {
    text-align: center;
    color: #333;
    margin-bottom: 20px;
}

.stats {
    display: flex;
    justify-content: space-around;
```

```css
    margin-bottom: 20px;
    padding: 15px;
    background: #f5f5f5;
    border-radius: 8px;
}

.stats span {
    font-size: 14px;
    color: #666;
}

.input-section {
    display: flex;
    gap: 10px;
    margin-bottom: 20px;
}

.input-section input {
    flex: 1;
    padding: 12px;
    border: 2px solid #e0e0e0;
    border-radius: 8px;
    font-size: 16px;
}

.input-section input:focus {
    outline: none;
    border-color: #667eea;
}

.input-section button {
    padding: 12px 24px;
    background: #667eea;
    color: white;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-weight: 600;
}

.input-section button:hover {
    background: #5568d3;
}

.todo-list {
    list-style: none;
    padding: 0;
    margin: 0;
}

.todo-list li {
    display: flex;
    align-items: center;
    padding: 15px;
    border-bottom: 1px solid #e0e0e0;
    gap: 10px;
}

```

```css
.todo-list li.completed span {
    text-decoration: line-through;
    color: #999;
}

.todo-list li span {
    flex: 1;
    cursor: pointer;
}

.todo-list li input[type="checkbox"] {
    width: 20px;
    height: 20px;
    cursor: pointer;
}

.delete-btn {
    background: none;
    border: none;
    cursor: pointer;
    font-size: 20px;
    opacity: 0.6;
    transition: opacity 0.2s;
}

.delete-btn:hover {
    opacity: 1;
}

.clear-completed {
    width: 100%;
    padding: 12px;
    margin-top: 15px;
    background: #f44336;
    color: white;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-weight: 600;
}

.clear-completed:hover {
    background: #da190b;
}

.empty {
    text-align: center;
    color: #999;
    padding: 40px;
    font-style: italic;
}
```

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

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

function App() {
    const onRenderCallback = (
        id,
        phase,
        actualDuration,
        baseDuration,
        startTime,
        commitTime
    ) => {
        console.log(`${id} (${phase}) took ${actualDuration}ms`);
    };

    return (
        <Profiler id="App" onRender={onRenderCallback}>
            <TodoList />
        </Profiler>
    );
}
```

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

```
// Componenti: PascalCase
function TodoList() {}
function UserProfile() {}

// File componenti: stesso nome del componente
// TodoList.jsx, UserProfile.jsx

// Funzioni/variabili: camelCase
const handleClick = () => {};
const userData = {};

// Costanti: UPPER_SNAKE_CASE
const API_URL = 'https://api.example.com';
const MAX_ITEMS = 100;

// Custom hooks: use + PascalCase
function useTodos() {}
function useLocalStorage() {}
```

2.7.3 3. Componenti Puliti

Listing 2.32: Componente ben strutturato

```
import { useState, useEffect } from 'react';
import PropTypes from 'prop-types';
import './TodoList.css';

// 1. Imports

// 2. Costanti
const FILTER_OPTIONS = ['all', 'active', 'completed'];

// 3. Componente
function TodoList({ initialTodos = [] }) {
    // 3.1 State
    const [todos, setTodos] = useState(initialTodos);
    const [filter, setFilter] = useState('all');

    // 3.2 Effects
    useEffect(() => {
        console.log('Todos changed:', todos);
    }, [todos]);

    // 3.3 Handlers
    const handleAddTodo = (text) => {
        // ...
    };

    // 3.4 Computed values
    const filteredTodos = todos.filter(todo => {
        if (filter === 'active') return !todo.completed;
        if (filter === 'completed') return todo.completed;
        return true;
    });

    // 3.5 Render
    return (
        <div className="todo-list">
            {/* JSX */}
        </div>
    );
}

// 4. PropTypes
TodoList.propTypes = {
    initialTodos: PropTypes.arrayOf(PropTypes.shape({
        id: PropTypes.number.isRequired,
        text: PropTypes.string.isRequired,
        completed: PropTypes.bool.isRequired
    }))
};

// 5. Export
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

```
// Questo è JSX
const element = <h1>Hello, World!</h1>;

// Viene trasformato in JavaScript puro:
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

```
function Greeting() {
    const name = 'Mario';
    const age = 25;
    const hobbies = ['coding', 'reading', 'gaming'];

    return (
        <div>
            {/* Variabili */}
            <h1>Ciao, {name}!</h1>

            {/* Operazioni */}
            <p>Tra 5 anni avrai {age + 5} anni</p>

            {/* Condizioni ternarie */}
            <p>Sei {age >= 18 ? 'maggiorenne' : 'minorenne'}</p>

            {/* Chiamate a funzioni */}
            <p>Nome uppercase: {name.toUpperCase()}</p>

            {/* Map per liste */}
            <ul>
                {hobbies.map((hobby, index) => (
                    <li key={index}>{hobby}</li>
                ))}
            </ul>

            {/* Template literals */}
            <p>{`${name} ha ${age} anni`}</p>
        </div>
    );
}
```

2. Attributi in JSX
Gli attributi seguono la convenzione camelCase:

Listing 3.3: Attributi JSX

```
function AttributesExample() {
    const imageUrl = 'https://example.com/image.jpg';
    const altText = 'Una bella immagine';
    const isActive = true;

    return (
        <div>
            {/* className invece di class */}
            <div className="container"></div>

            {/* htmlFor invece di for */}
            <label htmlFor="input-id">Nome:</label>
            <input id="input-id" type="text" />

            {/* Attributi booleani */}
            <input type="checkbox" checked={isActive} />
            <input type="text" disabled />
            <button type="submit" autoFocus>Invia</button>

            {/* onClick invece di onclick */}
            <button onClick={() => console.log('Click!')}>
                Click me
            </button>

            {/* Style come oggetto */}
            <div style={{
                color: 'red',
                fontSize: '20px',
                backgroundColor: '#f0f0f0'
            }}>
                Testo stilizzato
            </div>

            {/* Attributi dinamici */}
            <img src={imageUrl} alt={altText} />

            {/* Data attributes */}
            <div data-user-id="123" data-role="admin">
                User info
            </div>

            {/* Spread attributes */}
            <input {...{type: 'text', placeholder: 'Nome'}} />
        </div>
    );
}
```

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
<br>	<br />	Tag auto-chiudenti

4. Rendering Condizionale

Listing 3.4: Condizioni in JSX

```
function ConditionalRendering({ isLoggedIn, username, role }) {
    // 1. If-else con ternario
    return (
        <div>
            {isLoggedIn ? (
                <h1>Benvenuto, {username}!</h1>
            ) : (
                <h1>Per favore, effettua il login</h1>
            )}
        </div>
    );

    // 2. Logical AND (&&) per rendering condizionale
    return (
        <div>
            {isLoggedIn && <h1>Benvenuto, {username}!</h1>}
            {role === 'admin' && <button>Pannello Admin</button>}
        </div>
    );

    // 3. Variabile intermedia
    let content;
    if (isLoggedIn) {
        content = <h1>Benvenuto, {username}!</h1>;
    } else {
        content = <h1>Per favore, effettua il login</h1>;
    }
    return <div>{content}</div>;

    // 4. Early return
    if (!isLoggedIn) {
        return <LoginForm />;
    }
    return <Dashboard username={username} />;

    // 5. Switch con oggetto
    const statusMessages = {
        loading: <Spinner />,
        error: <ErrorMessage />,
        success: <SuccessMessage />,
        idle: null
    };
    return statusMessages[status];
}
```

5. Liste e Keys

Listing 3.5: Rendering liste

```
function ListRendering() {
    const users = [
        { id: 1, name: 'Mario', age: 25 },
        { id: 2, name: 'Laura', age: 30 },
        { id: 3, name: 'Giuseppe', age: 28 }
    ];

    return (
        <div>
            {/*     Con key unica (id) */}
            <ul>
                {users.map(user => (
                    <li key={user.id}>
                        {user.name} - {user.age} anni
                    </li>
                ))}
            </ul>

            {/*     Senza key (warning in console) */}
            <ul>
                {users.map(user => (
                    <li>{user.name}</li>
                ))}
            </ul>

            {/*        Con index come key (evitare se possibile) */}
            <ul>
                {users.map((user, index) => (
                    <li key={index}>{user.name}</li>
                ))}
            </ul>

            {/*     Liste complesse */}
            <div>
                {users
                    .filter(user => user.age >= 28)
                    .sort((a, b) => a.name.localeCompare(b.name))
                    .map(user => (
                        <UserCard key={user.id} user={user} />
                    ))
                }
            </div>
        </div>
    );
}
```

Perché le Keys sono Importanti:

Listing 3.6: Importanza delle keys

```
// Senza keys corrette, React può confondersi
// nei riordinamenti

function TodoList() {
    const [todos, setTodos] = useState([
        { id: 1, text: 'Primo' },
        { id: 2, text: 'Secondo' },
        { id: 3, text: 'Terzo' }
    ]);

    // Se riordini e usi index come key
    // React potrebbe aggiornare l'elemento sbagliato

    //     Male - usa index
    return (
        <ul>
            {todos.map((todo, index) => (
                <li key={index}>
                    <input type="checkbox" />
                    {todo.text}
                </li>
            ))}
        </ul>
    );

    //     Bene - usa id univoco
    return (
        <ul>
            {todos.map(todo => (
                <li key={todo.id}>
                    <input type="checkbox" />
                    {todo.text}
                </li>
            ))}
        </ul>
    );
}
```

3.1.3 JSX Avanzato

1. Fragments

Raggruppa elementi senza aggiungere nodi DOM:

Listing 3.7: React Fragments

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

function Table() {
    //      Aggiunge un div extra nel DOM
    return (
        <div>
            <td>Cella 1</td>
            <td>Cella 2</td>
        </div>
    );

    //      Sintassi lunga
    return (
        <Fragment>
            <td>Cella 1</td>
            <td>Cella 2</td>
        </Fragment>
    );

    //      Sintassi corta (più comune)
    return (
        <>
            <td>Cella 1</td>
            <td>Cella 2</td>
        </>
    );

    //      Con key (necessario per liste)
    return items.map(item => (
        <Fragment key={item.id}>
            <td>{item.name}</td>
            <td>{item.value}</td>
        </Fragment>
    ));
}
```

2. **Spread Props**

Listing 3.8: Spread di props

```
function Button({ type = 'button', className, ...rest }) {
    // Passa tutte le altre props al button
    return (
        <button
            type={type}
            className={`btn ${className}`}
            {...rest}
        />
    );
}

// Uso
function App() {
    return (
        <Button
            onClick={() => console.log('Click')}
            disabled
            aria-label="Submit button"
            data-testid="submit-btn"
        >
            Invia
        </Button>
    );
}
```

3. Children come Function

Listing 3.9: Render Props pattern

```
function DataFetcher({ url, children }) {
    const [data, setData] = useState(null);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        fetch(url)
            .then(res => res.json())
            .then(data => {
                setData(data);
                setLoading(false);
            });
    }, [url]);

    // Children è una funzione
    return children({ data, loading });
}

// Uso
function App() {
    return (
        <DataFetcher url="/api/users">
            {({ data, loading }) => (
                loading ? <Spinner /> : <UserList users={data} />
            )}
        </DataFetcher>
    );
}
```

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. Imports
import { useState, useEffect } from 'react';
import PropTypes from 'prop-types';
import './UserCard.css';

// 2. Componente
function UserCard({ user, onEdit, onDelete }) {
    // 3. State locale
    const [isExpanded, setIsExpanded] = useState(false);

    // 4. Effects
    useEffect(() => {
        console.log('UserCard montato');
        return () => console.log('UserCard smontato');
    }, []);

    // 5. Event handlers
    const handleToggle = () => {
        setIsExpanded(!isExpanded);
    };

    // 6. Computed values
    const fullName = `${user.firstName} ${user.lastName}`;
    const initials = `${user.firstName[0]}${user.lastName[0]}`;

    // 7. Render
    return (
        <div className="user-card">
            <div className="avatar">{initials}</div>

            <div className="info">
                <h3>{fullName}</h3>
                <p>{user.email}</p>

                {isExpanded && (
                    <div className="details">
                        <p>Ruolo: {user.role}</p>
                        <p>Registrato: {user.registeredAt}</p>
                    </div>
                )}
            </div>

            <div className="actions">
                <button onClick={handleToggle}>
                    {isExpanded ? 'Nascondi' : 'Mostra'}
                </button>
                <button onClick={() => onEdit(user.id)}>
                    Modifica
                </button>
                <button onClick={() => onDelete(user.id)}>
                    Elimina
                </button>
            </div>
        </div>
    );
}

// 8. PropTypes
UserCard.propTypes = {
    user: PropTypes.shape({
        id: PropTypes.number.isRequired,
        firstName: PropTypes.string.isRequired,
        lastName: PropTypes.string.isRequired,
        email: PropTypes.string.isRequired,
        role: PropTypes.string,
        registeredAt: PropTypes.string
    }).isRequired,
    onEdit: PropTypes.func.isRequired,
    onDelete: PropTypes.func.isRequired
};

// 9. Export
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

```
// Parent component
function App() {
    const user = {
        name: 'Mario Rossi',
        age: 30,
        email: 'mario@example.com'
    };

    return (
        <div>
            {/* Passaggio props */}
            <Greeting name="Mario" />
            <UserInfo user={user} />
            <Button label="Click me" onClick={() => alert('Clicked!')}
               />
        </div>
    );
}

// Child components
function Greeting({ name }) {
    return <h1>Ciao, {name}!</h1>;
}

function UserInfo({ user }) {
    return (
        <div>
            <p>Nome: {user.name}</p>
            <p>Età: {user.age}</p>
            <p>Email: {user.email}</p>
        </div>
    );
}

function Button({ label, onClick }) {
    return <button onClick={onClick}>{label}</button>;
}
```

2. Props Destructuring

Listing 3.12: Destructuring di props

```
//     Senza destructuring
function UserCard(props) {
    return (
        <div>
            <h3>{props.name}</h3>
            <p>{props.email}</p>
        </div>
    );
}

//      Con destructuring (raccomandato)
function UserCard({ name, email }) {
    return (
        <div>
            <h3>{name}</h3>
            <p>{email}</p>
        </div>
    );
}

//      Con valori di default
function UserCard({
    name = 'Guest',
    email = 'No email',
    role = 'user',
    isActive = true
}) {
    return (
        <div>
            <h3>{name}</h3>
            <p>{email}</p>
            <span>{role}</span>
            <span>{isActive ? 'Active' : 'Inactive'}</span>
        </div>
    );
}

//      Rest props
function Button({ children, className, ...rest }) {
    return (
        <button className={`btn ${className}`} {...rest}>
            {children}
        </button>
    );
}
```

3. Children Prop

Listing 3.13: Children prop

```
// Children come contenuto
function Card({ children, title }) {
    return (
        <div className="card">
            <h2>{title}</h2>
            <div className="card-body">
                {children}
            </div>
        </div>
    );
}

```

```
// Uso
function App() {
    return (
        <Card title="User Info">
            <p>Nome: Mario</p>
            <p>Email: mario@example.com</p>
            <button>Contatta</button>
        </Card>
    );
}

// Children multipli
function Layout({ header, sidebar, children, footer }) {
    return (
        <div className="layout">
            <header>{header}</header>
            <aside>{sidebar}</aside>
            <main>{children}</main>
            <footer>{footer}</footer>
        </div>
    );
}

// Uso
function App() {
    return (
        <Layout
            header={<Header />}
            sidebar={<Sidebar />}
            footer={<Footer />}
        >
            <h1>Contenuto principale</h1>
            <p>Lorem ipsum...</p>
        </Layout>
    );
}
```

4. Callback Props

Listing 3.14: Props come callbacks

```
// Parent passa funzioni ai child
function App() {
    const [count, setCount] = useState(0);

    const handleIncrement = () => {
        setCount(count + 1);
    };

    const handleDecrement = () => {
        setCount(count - 1);
    };

    const handleReset = () => {
        setCount(0);
    };

    return (
        <div>
```

```
            <Display value={count} />
            <Controls
                onIncrement={handleIncrement}
                onDecrement={handleDecrement}
                onReset={handleReset}
            />
        </div>
    );
}

function Display({ value }) {
    return <h1>Contatore: {value}</h1>;
}

function Controls({ onIncrement, onDecrement, onReset }) {
    return (
        <div>
            <button onClick={onDecrement}>-</button>
            <button onClick={onReset}>Reset</button>
            <button onClick={onIncrement}>+</button>
        </div>
    );
}
```

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

```
// Componenti atomici
function Avatar({ src, alt }) {
    return <img src={src} alt={alt} className="avatar" />;
}

function UserName({ name }) {
    return <h3 className="username">{name}</h3>;
}

function UserEmail({ email }) {
    return <p className="email">{email}</p>;
}

// Componente composto
function UserCard({ user }) {
    return (
        <div className="user-card">
            <Avatar src={user.avatar} alt={user.name} />
            <div>
                <UserName name={user.name} />
                <UserEmail email={user.email} />
            </div>
        </div>
    );
}

// Lista di UserCard
```

```
function UserList({ users }) {
    return (
        <div className="user-list">
            {users.map(user => (
                <UserCard key={user.id} user={user} />
            ))}
        </div>
    );
}
```

2. Container/Presentational Pattern

Listing 3.16: Container e Presentational components

```
// Presentational Component (UI pura)
function TodoListView({ todos, onToggle, onDelete }) {
    return (
        <ul className="todo-list">
            {todos.map(todo => (
                <TodoItem
                    key={todo.id}
                    todo={todo}
                    onToggle={onToggle}
                    onDelete={onDelete}
                />
            ))}
        </ul>
    );
}

function TodoItem({ todo, onToggle, onDelete }) {
    return (
        <li className={todo.completed ? 'completed' : ''}>
            <input
                type="checkbox"
                checked={todo.completed}
                onChange={() => onToggle(todo.id)}
            />
            <span>{todo.text}</span>
            <button onClick={() => onDelete(todo.id)}>Delete</button>
        </li>
    );
}

// Container Component (logica)
function TodoListContainer() {
    const [todos, setTodos] = useState([]);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        fetchTodos().then(data => {
            setTodos(data);
            setLoading(false);
        });
    }, []);

    const handleToggle = (id) => {
        setTodos(todos.map(todo =>
            todo.id === id
```

```
                ? { ...todo, completed: !todo.completed }
                : todo
        ));
    };

    const handleDelete = (id) => {
        setTodos(todos.filter(todo => todo.id !== id));
    };

    if (loading) return <Spinner />;

    return (
        <TodoListView
            todos={todos}
            onToggle={handleToggle}
            onDelete={handleDelete}
        />
    );
}
```

3. Specializzazione

Listing 3.17: Componenti specializzati

```
// Componente generico
function Dialog({ title, message, onClose, children }) {
    return (
        <div className="dialog">
            <div className="dialog-header">
                <h2>{title}</h2>
                <button onClick={onClose}>  </button>
            </div>
            <div className="dialog-body">
                {message && <p>{message}</p>}
                {children}
            </div>
        </div>
    );
}

// Componenti specializzati
function WelcomeDialog({ onClose }) {
    return (
        <Dialog
            title="Benvenuto"
            message="Grazie per esserti registrato!"
            onClose={onClose}
        />
    );
}

function ConfirmDialog({ title, message, onConfirm, onCancel }) {
    return (
        <Dialog title={title} message={message} onClose={onCancel}>
            <div className="dialog-actions">
                <button onClick={onCancel}>Annulla</button>
                <button onClick={onConfirm}>Conferma</button>
            </div>
        </Dialog>
    );
}

function AlertDialog({ message, onClose }) {
    return (
        <Dialog title="Attenzione" message={message} onClose={onClose}>
            <button onClick={onClose}>OK</button>
        </Dialog>
    );
}
```

4. Compound Components

Listing 3.18: Compound Components pattern

```
// Tabs compound component
function Tabs({ children, defaultActive = 0 }) {
    const [activeIndex, setActiveIndex] = useState(defaultActive);

    return (
        <div className="tabs">
            {React.Children.map(children, (child, index) => {
                return React.cloneElement(child, {
                    isActive: index === activeIndex,
                    onClick: () => setActiveIndex(index),
                    index
                });
            })}
        </div>
    );
}

function Tab({ label, children, isActive, onClick }) {
    return (
        <div className={`tab ${isActive ? 'active' : ''}`}>
            <button onClick={onClick}>{label}</button>
            {isActive && <div className="tab-content">{children}</div>}
        </div>
    );
}

// Uso
function App() {
    return (
        <Tabs defaultActive={1}>
            <Tab label="Tab 1">
                <p>Contenuto Tab 1</p>
            </Tab>
            <Tab label="Tab 2">
                <p>Contenuto Tab 2</p>
            </Tab>
            <Tab label="Tab 3">
                <p>Contenuto Tab 3</p>
            </Tab>
        </Tabs>
    );
}
```

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

```
// App - Root component
function App() {
    const [user, setUser] = useState({
        name: 'Mario Rossi',
        avatar: '/avatar.jpg'
    });

    return (
        <div className="app">
            <Header user={user} onLogout={() => setUser(null)} />
            <TodoApp />
            <Footer />
        </div>
    );
}

// Header component
function Header({ user, onLogout }) {
    return (
        <header className="header">
            <Logo />
            <UserMenu user={user} onLogout={onLogout} />
        </header>
    );
}

function Logo() {
    return <div className="logo">     TodoApp</div>;
}

function UserMenu({ user, onLogout }) {
    const [isOpen, setIsOpen] = useState(false);

    return (
        <div className="user-menu">
            <Avatar src={user.avatar} onClick={() => setIsOpen(!isOpen)}
                />
            {isOpen && (
                <Dropdown>
                    <MenuItem label="Profile" onClick={() => {}} />
                    <MenuItem label="Settings" onClick={() => {}} />
                    <MenuItem label="Logout" onClick={onLogout} />
                </Dropdown>
            )}
        </div>
    );
}

// TodoApp component
function TodoApp() {
    const [todos, setTodos] = useState([]);
    const [filter, setFilter] = useState('all');

    const addTodo = (text) => {
        setTodos([...todos, {
            id: Date.now(),
            text,
            completed: false
        }]);
    };

    const toggleTodo = (id) => {
        setTodos(todos.map(todo =>
            todo.id === id
```

```
                ? { ...todo, completed: !todo.completed }
                : todo
        ));
    };

    const deleteTodo = (id) => {
        setTodos(todos.filter(todo => todo.id !== id));
    };

    const filteredTodos = todos.filter(todo => {
        if (filter === 'active') return !todo.completed;
        if (filter === 'completed') return todo.completed;
        return true;
    });

    return (
        <main className="todo-app">
            <TodoInput onAdd={addTodo} />
            <FilterBar filter={filter} onFilterChange={setFilter} />
            <TodoList
                todos={filteredTodos}
                onToggle={toggleTodo}
                onDelete={deleteTodo}
            />
            <TodoStats todos={todos} />
        </main>
    );
}

// Componenti foglia (leaf components)
function TodoInput({ onAdd }) {
    const [text, setText] = useState('');

    const handleSubmit = (e) => {
        e.preventDefault();
        if (text.trim()) {
            onAdd(text);
            setText('');
        }
    };

    return (
        <form onSubmit={handleSubmit} className="todo-input">
            <Input
                value={text}
                onChange={setText}
                placeholder="Nuovo task..."
            />
            <Button type="submit">Aggiungi</Button>
        </form>
    );
}

function TodoList({ todos, onToggle, onDelete }) {
    if (todos.length === 0) {
        return <EmptyState />;
    }

    return (
        <ul className="todo-list">
            {todos.map(todo => (
                <TodoItem
                    key={todo.id}
                    todo={todo}
                    onToggle={onToggle}
                    onDelete={onDelete}
                />
            ))}
        </ul>
    );
}

function TodoStats({ todos }) {
    const total = todos.length;
    const completed = todos.filter(t => t.completed).length;
    const pending = total - completed;

    return (
        <div className="todo-stats">
            <Stat label="Totale" value={total} />
            <Stat label="Completati" value={completed} />
            <Stat label="Da fare" value={pending} />
        </div>
    );
}
```

3.4 Best Practices

3.4.1 1. Naming e Organizzazione

Listing 3.20: Convenzioni di naming

```
//     Componenti: PascalCase
function UserProfile() {}
function TodoList() {}

//     File: stesso nome del componente
// UserProfile.jsx
// TodoList.jsx

//     Props: camelCase descrittivo
function Button({ onClick, isDisabled, primaryColor }) {}

//     Event handlers: handle + Event
const handleClick = () => {};
const handleSubmit = () => {};
const handleInputChange = () => {};

//     Boolean props: is/has/should prefix
const isLoading = false;
const hasError = true;
const shouldRender = true;

//     Evitare nomi generici
function Component() {} // Troppo generico
function MyComponent() {} // "My" non aggiunge informazioni
```

3.4.2 2. Componenti Piccoli e Focalizzati

Listing 3.21: Single Responsibility

```
//     Componente che fa troppo
function UserDashboard() {
    // 100+ righe di logica mista
    // Fetching dati, gestione form, rendering complesso
}

//     Dividi responsabilità
function UserDashboard() {
    return (
        <div>
            <UserStats />
            <UserActivity />
            <UserSettings />
        </div>
    );
}

function UserStats() {
    // Solo statistiche
}

function UserActivity() {
    // Solo attività recente
}

function UserSettings() {
    // Solo impostazioni
}
```

3.4.3 3. Props Validation

Listing 3.22: PropTypes per validazione

```
import PropTypes from 'prop-types';

function UserCard({ user, onEdit, onDelete, isSelected }) {
    // ...
}

UserCard.propTypes = {
    user: PropTypes.shape({
        id: PropTypes.number.isRequired,
        name: PropTypes.string.isRequired,
        email: PropTypes.string.isRequired,
        avatar: PropTypes.string,
        role: PropTypes.oneOf(['admin', 'user', 'guest'])
    }).isRequired,
    onEdit: PropTypes.func.isRequired,
    onDelete: PropTypes.func,
    isSelected: PropTypes.bool
};

UserCard.defaultProps = {
    isSelected: false,
    onDelete: () => {}
};
```

3.4.4 4. Evitare Prop Drilling Eccessivo

Listing 3.23: Problema prop drilling

```
//      Prop drilling eccessivo
function App() {
    const [theme, setTheme] = useState('light');
    return <Layout theme={theme} setTheme={setTheme} />;
}

function Layout({ theme, setTheme }) {
    return <Sidebar theme={theme} setTheme={setTheme} />;
}

function Sidebar({ theme, setTheme }) {
    return <ThemeToggle theme={theme} setTheme={setTheme} />;
}

function ThemeToggle({ theme, setTheme }) {
    return <button onClick={() => setTheme(/* ... */)}>Toggle</button>;
}

//      Usa Context (vedi capitoli successivi)
const ThemeContext = createContext();

function App() {
    const [theme, setTheme] = useState('light');
    return (
        <ThemeContext.Provider value={{ theme, setTheme }}>
            <Layout />
        </ThemeContext.Provider>
    );
}

function ThemeToggle() {
    const { theme, setTheme } = useContext(ThemeContext);
    return <button onClick={() => setTheme(/* ... */)}>Toggle</button>;
}
```

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

```
function Welcome({ name }) {
    //      ERRORE: Non puoi modificare le props
    // name = 'Nuovo nome'; // TypeError in Strict Mode

    //      Le props si leggono soltanto
    return <h1>Benvenuto, {name}!</h1>;
}

function App() {
    const [userName, setUserName] = useState('Mario');

    // Per cambiare i dati, cambi lo state nel parent
    return (
        <div>
            <Welcome name={userName} />
            <button onClick={() => setUserName('Luigi')}>
                Cambia nome
            </button>
        </div>
    );
}
```

4.1.2 Tipi di Props

1. Props Primitive

Listing 4.2: Props primitive

```
function UserProfile({
    name,         // string
    age,          // number
    isActive,     // boolean
    score         // number
}) {
    return (
        <div>
            <h2>{name}</h2>
            <p>Età: {age}</p>
            <p>Stato: {isActive ? 'Attivo' : 'Inattivo'}</p>
            <p>Punteggio: {score}</p>
        </div>
    );
}

// Uso
<UserProfile
    name="Mario"
    age={30}
    isActive={true}
    score={95.5}
/>
```

2. Props Oggetti e Array

Listing 4.3: Props complesse

```
function ProductCard({ product, tags, seller }) {
    return (
        <div className="product-card">
            <h3>{product.name}</h3>
            <p>   {product.price}</p>
            <p>{product.description}</p>

            <div className="tags">
                {tags.map(tag => (
                    <span key={tag} className="tag">{tag}</span>
                ))}
            </div>

            <div className="seller">
                <img src={seller.avatar} alt={seller.name} />
                <span>{seller.name}</span>
            </div>
        </div>
    );
}

// Uso
function App() {
    const product = {
        id: 1,
        name: 'Laptop',
        price: 999,
        description: 'Un ottimo laptop'
    };

    const tags = ['Elettronica', 'Computer', 'Offerta'];

    const seller = {
        id: 10,
        name: 'TechStore',
        avatar: '/seller.jpg'
    };

    return (
        <ProductCard
            product={product}
            tags={tags}
            seller={seller}
        />
    );
}
```

3. Props Funzioni (Callbacks)

Listing 4.4: Callback props

```
function SearchBar({ onSearch, onClear, onFilterChange }) {
    const [query, setQuery] = useState('');
    const [filter, setFilter] = useState('all');

    const handleSearch = () => {
        onSearch(query, filter);
    };

    const handleClear = () => {
        setQuery('');
        onClear();
    };

    const handleFilterChange = (newFilter) => {
        setFilter(newFilter);
        onFilterChange(newFilter);
    };

    return (
        <div className="search-bar">
            <input
                value={query}
                onChange={(e) => setQuery(e.target.value)}
                placeholder="Cerca..."
            />
            <button onClick={handleSearch}>Cerca</button>
            <button onClick={handleClear}>Cancella</button>

            <select
                value={filter}
                onChange={(e) => handleFilterChange(e.target.value)}
            >
                <option value="all">Tutti</option>
                <option value="active">Attivi</option>
                <option value="archived">Archiviati</option>
            </select>
        </div>
    );
}

// Uso nel parent
function App() {
    const handleSearch = (query, filter) => {
        console.log('Ricerca:', query, 'Filtro:', filter);
        // Fetch dati...
    };

    const handleClear = () => {
        console.log('Reset ricerca');
    };

    const handleFilterChange = (filter) => {
        console.log('Filtro cambiato:', filter);
    };

    return (
        <SearchBar
            onSearch={handleSearch}
            onClear={handleClear}
            onFilterChange={handleFilterChange}
        />
    );
}
```

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

Listing 4.5: Componenti come props

```
function Layout({ header, sidebar, footer, children }) {
    return (
        <div className="layout">
            <header className="header">{header}</header>
            <div className="main-content">
                <aside className="sidebar">{sidebar}</aside>
                <main className="content">{children}</main>
            </div>
            <footer className="footer">{footer}</footer>
        </div>
    );
}

// Uso
function App() {
    return (
        <Layout
            header={<AppHeader />}
            sidebar={<Navigation />}
            footer={<AppFooter />}
        >
            <h1>Contenuto principale</h1>
            <p>Contenuto della pagina...</p>
        </Layout>
    );
}
```

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

```
// Livello 1: App
function App() {
    const [user, setUser] = useState({
        name: 'Mario',
        theme: 'dark',
        language: 'it'
    });

    return <Layout user={user} setUser={setUser} />;
}

// Livello 2: Layout (non usa user, solo passa)
function Layout({ user, setUser }) {
    return (
        <div>
            <Header user={user} setUser={setUser} />
            <Main user={user} setUser={setUser} />
        </div>
    );
}

// Livello 3: Header (non usa user, solo passa)
function Header({ user, setUser }) {
    return (
        <header>
            <Navigation user={user} setUser={setUser} />
        </header>
    );
}

// Livello 4: Navigation (non usa user, solo passa)
function Navigation({ user, setUser }) {
    return (
        <nav>
            <UserMenu user={user} setUser={setUser} />
        </nav>
    );
}

// Livello 5: UserMenu (FINALMENTE usa user!)
function UserMenu({ user, setUser }) {
    return (
        <div>
            <span>{user.name}</span>
            <button onClick={() => setUser({ ...user, theme: 'light' })
                }>
                 Cambia tema
            </button>
        </div>
    );
}
```

4.2.2 Soluzioni al Props Drilling

1. Component Composition

Listing 4.7: Composizione invece di drilling

```
//      Passa il componente completo, non le props
function App() {
    const [user, setUser] = useState({
        name: 'Mario',
        theme: 'dark'
    });

    // Costruisci UserMenu qui, dove hai accesso a user
    const userMenu = <UserMenu user={user} setUser={setUser} />;

    return <Layout userMenu={userMenu} />;
}

function Layout({ userMenu }) {
    return (
        <div>
            <Header userMenu={userMenu} />
            <Main />
        </div>
    );
}

function Header({ userMenu }) {
    return (
        <header>
            <Navigation userMenu={userMenu} />
        </header>
    );
}

function Navigation({ userMenu }) {
    return (
        <nav>
            <Logo />
            {userMenu}  {/* Renderizza direttamente */}
        </nav>
    );
}

function UserMenu({ user, setUser }) {
    return (
        <div>
            <span>{user.name}</span>
            <button onClick={() => setUser({ ...user, theme: 'light' })
               }>
                Cambia tema
            </button>
        </div>
    );
}
```

2. Context API

Listing 4.8: Context per evitare drilling

```
import { createContext, useContext, useState } from 'react';

// 1. Crea Context
```

```
const UserContext = createContext();

// 2. Provider nel componente root
function App() {
    const [user, setUser] = useState({
        name: 'Mario',
        theme: 'dark'
    });

    return (
        <UserContext.Provider value={{ user, setUser }}>
            <Layout />
        </UserContext.Provider>
    );
}

// 3. Componenti intermedi non hanno bisogno delle props
function Layout() {
    return (
        <div>
            <Header />
            <Main />
        </div>
    );
}

function Header() {
    return (
        <header>
            <Navigation />
        </header>
    );
}

function Navigation() {
    return (
        <nav>
            <Logo />
            <UserMenu />
        </nav>
    );
}

// 4. Consuma il context dove serve
function UserMenu() {
    const { user, setUser } = useContext(UserContext);

    return (
        <div>
            <span>{user.name}</span>
            <button onClick={() => setUser({ ...user, theme: 'light' })
                }>
                Cambia tema
            </button>
        </div>
    );
}
```

3. Custom Hook per Context

Listing 4.9: Custom hook per context

```
// hooks/useUser.js
import { createContext, useContext, useState } from 'react';

const UserContext = createContext();

export function UserProvider({ children }) {
    const [user, setUser] = useState({
        name: 'Mario',
        theme: 'dark',
        language: 'it'
    });

    const updateUser = (updates) => {
        setUser(prev => ({ ...prev, ...updates }));
    };

    const toggleTheme = () => {
        setUser(prev => ({
            ...prev,
            theme: prev.theme === 'dark' ? 'light' : 'dark'
        }));
    };

    return (
        <UserContext.Provider value={{
            user,
            setUser,
            updateUser,
            toggleTheme
        }}>
            {children}
        </UserContext.Provider>
    );
}

export function useUser() {
    const context = useContext(UserContext);
    if (!context) {
        throw new Error('useUser must be used within UserProvider');
    }
    return context;
}

// App.jsx
import { UserProvider } from './hooks/useUser';

function App() {
    return (
        <UserProvider>
            <Layout />
        </UserProvider>
    );
}

// Qualsiasi componente
```

```
import { useUser } from './hooks/useUser';

function UserMenu() {
    const { user, toggleTheme } = useUser();

    return (
        <div>
            <span>{user.name}</span>
            <button onClick={toggleTheme}>
                Tema: {user.theme}
            </button>
        </div>
    );
}
```

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

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

function Counter() {
    // Dichiarazione state
    const [count, setCount] = useState(0);
    //     ^        ^              ^
    //     |        |              |
    //   valore   setter      valore iniziale

    // Aggiorna state
    const increment = () => {
        setCount(count + 1);
    };

    const decrement = () => {
        setCount(count - 1);
    };

    const reset = () => {
        setCount(0);
    };

    return (
        <div>
            <h1>Contatore: {count}</h1>
            <button onClick={decrement}>-</button>
            <button onClick={reset}>Reset</button>
            <button onClick={increment}>+</button>
        </div>
    );
}
```

4.3.2 State con Oggetti

Listing 4.11: State oggetto

```
function UserForm() {
    const [user, setUser] = useState({
        firstName: '',
        lastName: '',
        email: '',
        age: ''
    });

    //     SBAGLIATO: Muta direttamente lo state
    const handleChangeBad = (field, value) => {
        user[field] = value; // NO!
        setUser(user);       // React non rileva il cambiamento
    };

    //     CORRETTO: Crea nuovo oggetto
    const handleChange = (field, value) => {
        setUser({
            ...user,          // Copia proprietà esistenti
            [field]: value    // Sovrascrivi campo specifico
        });
    };

    //     Alternativa con funzione updater
    const handleChangeUpdater = (field, value) => {
        setUser(prevUser => ({
            ...prevUser,
            [field]: value
        }));
    };

    const handleSubmit = (e) => {
        e.preventDefault();
        console.log('User data:', user);
    };

    return (
        <form onSubmit={handleSubmit}>
            <input
                type="text"
                value={user.firstName}
                onChange={(e) => handleChange('firstName', e.target.value)}
                placeholder="Nome"
            />
            <input
                type="text"
                value={user.lastName}
                onChange={(e) => handleChange('lastName', e.target.value)}
                placeholder="Cognome"
            />
            <input
                type="email"
                value={user.email}
                onChange={(e) => handleChange('email', e.target.value)}
                placeholder="Email"
            />
            <input
                type="number"
                value={user.age}
                onChange={(e) => handleChange('age', e.target.value)}
                placeholder="Età"
            />
            <button type="submit">Invia</button>

            {/* Preview dati */}
            <pre>{JSON.stringify(user, null, 2)}</pre>
        </form>
    );
}
```

4.3.3 State con Array

Listing 4.12: State array

```
function TodoList() {
    const [todos, setTodos] = useState([
        { id: 1, text: 'Imparare React', completed: false },
        { id: 2, text: 'Costruire app', completed: false }
    ]);

    // Aggiungi elemento
    const addTodo = (text) => {
        const newTodo = {
            id: Date.now(),
            text,
            completed: false
        };

        //     SBAGLIATO
        // todos.push(newTodo);
        // setTodos(todos);

        //     CORRETTO
        setTodos([...todos, newTodo]);
        // oppure
        setTodos(prevTodos => [...prevTodos, newTodo]);
    };

    // Rimuovi elemento
    const deleteTodo = (id) => {
        //     Filter crea nuovo array
        setTodos(todos.filter(todo => todo.id !== id));
    };

    // Aggiorna elemento
    const toggleTodo = (id) => {
        //     Map crea nuovo array
        setTodos(todos.map(todo =>
            todo.id === id
                ? { ...todo, completed: !todo.completed }
                : todo
        ));
    };

    // Aggiorna elemento con indice
    const updateTodoAtIndex = (index, newText) => {
        setTodos(todos.map((todo, i) =>
            i === index
                ? { ...todo, text: newText }
                : todo
        ));
    };

    // Inserisci elemento a inizio
    const prependTodo = (text) => {
        const newTodo = { id: Date.now(), text, completed: false };
        setTodos([newTodo, ...todos]);
    };

    // Ordina array
    const sortTodos = () => {
        setTodos([...todos].sort((a, b) =>
            a.text.localeCompare(b.text)
        ));
    };

    // Rimuovi tutti completati
    const clearCompleted = () => {
        setTodos(todos.filter(todo => !todo.completed));
    };

    return (
        <div>
            {/* UI componenti... */}
        </div>
    );
}
```

4.3.4 Multiple State Updates

Listing 4.13: Aggiornamenti multipli

```
function ShoppingCart() {
    const [items, setItems] = useState([]);
    const [total, setTotal] = useState(0);
    const [discount, setDiscount] = useState(0);

    //     Problema: Usa state precedente
    const addItem = (item) => {
        setItems([...items, item]);
        setTotal(total + item.price); // total potrebbe essere stale
    };

    //     Usa funzione updater per accesso allo state più recente
    const addItemCorrect = (item) => {
        setItems(prevItems => [...prevItems, item]);
        setTotal(prevTotal => prevTotal + item.price);
    };

    //     Aggiornamenti batch automatici in React 18+
    const handleCheckout = () => {
        setItems([]);
        setTotal(0);
        setDiscount(0);
        // Tutti e tre gli aggiornamenti causano un solo re-render
    };

    // Calcola totale da items (computed value)
    const calculatedTotal = items.reduce((sum, item) => sum + item.price
        , 0);
    const finalTotal = calculatedTotal - discount;

    return (
        <div>
            <p>Items: {items.length}</p>
            <p>Totale:    {finalTotal}</p>
        </div>
    );
}
```

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

```
//     State duplicato in componenti separati
function TemperatureInput() {
    const [temperature, setTemperature] = useState('');
    return <input value={temperature} onChange={e => setTemperature(e.
        target.value)} />;
}

function App() {
    return (
        <div>
            <TemperatureInput /> {/* Celsius */}
            <TemperatureInput /> {/* Fahrenheit */}
            {/* I due input non sono sincronizzati! */}
        </div>
    );
}

//     State condiviso nel parent
function TemperatureInput({ scale, temperature, onTemperatureChange }) {
    const handleChange = (e) => {
        onTemperatureChange(e.target.value);
    };

    const scaleNames = {
        c: 'Celsius',
        f: 'Fahrenheit'
    };

    return (
        <fieldset>
```

```
            <legend>Temperatura in {scaleNames[scale]}:</legend>
            <input value={temperature} onChange={handleChange} />
        </fieldset>
    );
}

function App() {
    const [temperature, setTemperature] = useState('');
    const [scale, setScale] = useState('c');

    const handleCelsiusChange = (temp) => {
        setScale('c');
        setTemperature(temp);
    };

    const handleFahrenheitChange = (temp) => {
        setScale('f');
        setTemperature(temp);
    };

    // Conversioni
    const celsius = scale === 'f'
        ? ((temperature - 32) * 5 / 9).toFixed(1)
        : temperature;

    const fahrenheit = scale === 'c'
        ? (temperature * 9 / 5 + 32).toFixed(1)
        : temperature;

    return (
        <div>
            <TemperatureInput
                scale="c"
                temperature={celsius}
                onTemperatureChange={handleCelsiusChange}
            />
            <TemperatureInput
                scale="f"
                temperature={fahrenheit}
                onTemperatureChange={handleFahrenheitChange}
            />
            <BoilingVerdict celsius={parseFloat(celsius)} />
        </div>
    );
}

function BoilingVerdict({ celsius }) {
    if (celsius >= 100) {
        return <p>L'acqua bollirebbe.</p>;
    }
    return <p>L'acqua non bollirebbe.</p>;
}
```

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

```
function ControlledInput() {
    const [value, setValue] = useState('');

    //     Controlled: React controlla il valore
    return (
        <div>
            <input
                type="text"
                value={value}
                onChange={(e) => setValue(e.target.value)}
            />
            <p>Valore: {value}</p>
        </div>
    );

    //     Uncontrolled: DOM controlla il valore
    // return <input type="text" />;
}
```

4.4.2 Form Completo Controllato

Listing 4.16: Form completamente controllato

```
function RegistrationForm() {
    const [formData, setFormData] = useState({
        username: '',
        email: '',
        password: '',
        confirmPassword: '',
        age: '',
        country: '',
        terms: false,
        newsletter: false,
        gender: '',
        interests: []
    });

    const [errors, setErrors] = useState({});

    // Handler generico per input text/email/password/number
    const handleInputChange = (e) => {
        const { name, value } = e.target;
        setFormData(prev => ({
            ...prev,
            [name]: value
        }));
    };

    // Handler per checkbox
    const handleCheckboxChange = (e) => {
        const { name, checked } = e.target;
        setFormData(prev => ({
            ...prev,
            [name]: checked
        }));
    };

    // Handler per checkbox multipli (interests)
    const handleInterestChange = (interest) => {
        setFormData(prev => ({
            ...prev,
            interests: prev.interests.includes(interest)
                ? prev.interests.filter(i => i !== interest)
                : [...prev.interests, interest]
        }));
    };

    // Validazione
    const validate = () => {
        const newErrors = {};

        if (!formData.username) {
            newErrors.username = 'Username richiesto';
        } else if (formData.username.length < 3) {
            newErrors.username = 'Username troppo corto';
        }

        if (!formData.email) {
            newErrors.email = 'Email richiesta';
        } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
            newErrors.email = 'Email non valida';
        }

        if (!formData.password) {
            newErrors.password = 'Password richiesta';
        } else if (formData.password.length < 8) {
            newErrors.password = 'Password troppo corta (min 8 caratteri
                )';
        }

        if (formData.password !== formData.confirmPassword) {
            newErrors.confirmPassword = 'Le password non coincidono';
        }

        if (!formData.terms) {
            newErrors.terms = 'Devi accettare i termini';
        }

        setErrors(newErrors);
        return Object.keys(newErrors).length === 0;
    };

    const handleSubmit = (e) => {
        e.preventDefault();

        if (validate()) {
            console.log('Form valido:', formData);
            // Invia dati al server...
        } else {
            console.log('Form non valido:', errors);
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            {/* Text Input */}
            <div>
                <label htmlFor="username">Username:</label>
                <input
                    id="username"
                    type="text"
                    name="username"
                    value={formData.username}
                    onChange={handleInputChange}
                />
                {errors.username && <span className="error">{errors.username}</span>}
            </div>

            {/* Email Input */}
            <div>
                <label htmlFor="email">Email:</label>
                <input
                    id="email"
                    type="email"
                    name="email"
                    value={formData.email}
                    onChange={handleInputChange}
                />
                {errors.email && <span className="error">{errors.email}</span>}
            </div>

            {/* Password Input */}
            <div>
                <label htmlFor="password">Password:</label>
                <input
                    id="password"
                    type="password"
                    name="password"
                    value={formData.password}
                    onChange={handleInputChange}
                />
                {errors.password && <span className="error">{errors.password}</span>}
            </div>

            {/* Confirm Password */}
            <div>
                <label htmlFor="confirmPassword">Conferma Password:</label>
                <input
                    id="confirmPassword"
                    type="password"
                    name="confirmPassword"
                    value={formData.confirmPassword}
                    onChange={handleInputChange}
                />
                {errors.confirmPassword && <span className="error">{errors.confirmPassword}</span>}
            </div>

            {/* Number Input */}
            <div>
                <label htmlFor="age">Età:</label>
                <input
                    id="age"
                    type="number"
                    name="age"
                    value={formData.age}
                    onChange={handleInputChange}
                    min="18"
                    max="120"
                />
            </div>

            {/* Select */}
            <div>
                <label htmlFor="country">Paese:</label>
                <select
                    id="country"
                    name="country"
                    value={formData.country}
                    onChange={handleInputChange}
                >
                    <option value="">Seleziona...</option>
                    <option value="IT">Italia</option>
                    <option value="US">USA</option>
                    <option value="UK">UK</option>
                </select>
            </div>

            {/* Radio Buttons */}
            <div>
                <label>Genere:</label>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="male"
                        checked={formData.gender === 'male'}
                        onChange={handleInputChange}
                    />
                    Maschio
                </label>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="female"
                        checked={formData.gender === 'female'}
                        onChange={handleInputChange}
                    />
                    Femmina
                </label>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="other"
                        checked={formData.gender === 'other'}
                        onChange={handleInputChange}
                    />
                    Altro
                </label>
            </div>

            {/* Checkboxes Multiple */}
            <div>
                <label>Interessi:</label>
                {['Sport', 'Musica', 'Lettura', 'Viaggi'].map(interest => (
                    <label key={interest}>
                        <input
                            type="checkbox"
                            checked={formData.interests.includes(interest)}
                            onChange={() => handleInterestChange(interest)}
                        />
                        {interest}
                    </label>
                ))}
            </div>

            {/* Checkbox Singola */}
            <div>
                <label>
                    <input
                        type="checkbox"
                        name="terms"
                        checked={formData.terms}
                        onChange={handleCheckboxChange}
                    />
                    Accetto i termini e condizioni
                </label>
                {errors.terms && <span className="error">{errors.terms}</span>}
            </div>

            <div>
                <label>
                    <input
                        type="checkbox"
                        name="newsletter"
                        checked={formData.newsletter}
                        onChange={handleCheckboxChange}
                    />
                    Iscriviti alla newsletter
                </label>
            </div>

            {/* Submit */}
            <button type="submit">Registrati</button>

            {/* Debug */}
            <pre>{JSON.stringify(formData, null, 2)}</pre>
        </form>
    );
}
```

4.4.3 Controlled vs Uncontrolled

Listing 4.17: Controlled vs Uncontrolled

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

function ComparisonExample() {
    // Controlled
    const [controlledValue, setControlledValue] = useState('');

    // Uncontrolled
    const uncontrolledRef = useRef();

    const handleControlledSubmit = (e) => {
        e.preventDefault();
        console.log('Controlled:', controlledValue);
    };

    const handleUncontrolledSubmit = (e) => {
        e.preventDefault();
        console.log('Uncontrolled:', uncontrolledRef.current.value);
    };

    return (
        <div>
            {/* Controlled Input */}
            <form onSubmit={handleControlledSubmit}>
                <h3>Controlled</h3>
                <input
                    type="text"
                    value={controlledValue}
                    onChange={(e) => setControlledValue(e.target.value)}
                />
                <button type="submit">Invia</button>
                <p>Live value: {controlledValue}</p>
            </form>

            {/* Uncontrolled Input */}
            <form onSubmit={handleUncontrolledSubmit}>
                <h3>Uncontrolled</h3>
                <input
                    type="text"
                    ref={uncontrolledRef}
                    defaultValue="Initial"
                />
                <button type="submit">Invia</button>
                <p>Value disponibile solo al submit</p>
            </form>
        </div>
    );
}

```

```
// Quando usare controlled vs uncontrolled:

//     Usa CONTROLLED quando:
// - Hai bisogno di validazione in tempo reale
// - Vuoi formattare l'input mentre digiti
// - Devi disabilitare il submit in base all'input
// - Hai input dipendenti tra loro
// - Vuoi mostrare l'input in tempo reale

//     Usa UNCONTROLLED quando:
// - Form molto grandi con molti campi
// - Integri con librerie non-React
// - Performance critiche
// - File uploads
// - 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

```
function ShoppingApp() {
    //     State centralizzato
    const [cart, setCart] = useState({
        items: [],
        total: 0,
        itemCount: 0
    });

    // Computed values derivati dallo state
    const subtotal = cart.items.reduce((sum, item) => sum + item.price * item.quantity, 0);
    const tax = subtotal * 0.22;
    const total = subtotal + tax;

    const addItem = (product) => {
        setCart(prev => {
            const existingItem = prev.items.find(item => item.id === product.id);

            if (existingItem) {
                // Incrementa quantità
                return {
                    ...prev,
                    items: prev.items.map(item =>
                        item.id === product.id
                            ? { ...item, quantity: item.quantity + 1 }
                            : item
                    )
                };
            } else {
                // Aggiungi nuovo item
                return {
                    ...prev,
                    items: [...prev.items, { ...product, quantity: 1 }]
                };
            }
        });
    };

    return (
        <div>
            <ProductList onAddToCart={addItem} />
            <Cart
                items={cart.items}
                subtotal={subtotal}
                tax={tax}
                total={total}
            />
        </div>
    );
}
```

4.5.2 Derived State (Computed Values)

Listing 4.19: Evita state ridondante

```
function UserList() {
    const [users, setUsers] = useState([...]);
    const [searchQuery, setSearchQuery] = useState('');
    const [sortBy, setSortBy] = useState('name');

    //      SBAGLIATO: State ridondante
    // const [filteredUsers, setFilteredUsers] = useState([]);
    // const [sortedUsers, setSortedUsers] = useState([]);

    //      CORRETTO: Calcola al render
    const filteredUsers = users.filter(user =>
        user.name.toLowerCase().includes(searchQuery.toLowerCase())
    );

    const sortedUsers = [...filteredUsers].sort((a, b) => {
        if (sortBy === 'name') return a.name.localeCompare(b.name);
        if (sortBy === 'age') return a.age - b.age;
        return 0;
    });

    return (
        <div>
            <input
                value={searchQuery}
                onChange={(e) => setSearchQuery(e.target.value)}
                placeholder="Cerca..."
            />
            <select value={sortBy} onChange={(e) => setSortBy(e.target.
               value)}>
                <option value="name">Nome</option>
                <option value="age">Età</option>
            </select>

            <ul>
                {sortedUsers.map(user => (
                    <li key={user.id}>{user.name} - {user.age}</li>
                ))}
            </ul>
        </div>
    );
}
```

4.5.3 State Reducer Pattern

Listing 4.20: Pattern reducer per state complesso

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

// Reducer function
function cartReducer(state, action) {
    switch (action.type) {
        case 'ADD_ITEM':
            const existingIndex = state.items.findIndex(
                item => item.id === action.payload.id
            );

            if (existingIndex >= 0) {
                const newItems = [...state.items];
                newItems[existingIndex].quantity += 1;
                return { ...state, items: newItems };
            }

            return {
                ...state,
                items: [...state.items, { ...action.payload, quantity: 1
                    }]
            };

        case 'REMOVE_ITEM':
            return {
                ...state,
                items: state.items.filter(item => item.id !== action.
                   payload)
            };

        case 'UPDATE_QUANTITY':
            return {
                ...state,
                items: state.items.map(item =>
                    item.id === action.payload.id
                        ? { ...item, quantity: action.payload.quantity }
                        : item
                )
            };

        case 'CLEAR_CART':
            return { ...state, items: [] };

        default:
            return state;
    }
}

```

```
function ShoppingCart() {
    const [state, dispatch] = useReducer(cartReducer, { items: [] });

    const addItem = (product) => {
        dispatch({ type: 'ADD_ITEM', payload: product });
    };

    const removeItem = (id) => {
        dispatch({ type: 'REMOVE_ITEM', payload: id });
    };

    const updateQuantity = (id, quantity) => {
        dispatch({ type: 'UPDATE_QUANTITY', payload: { id, quantity } });
    };

    const clearCart = () => {
        dispatch({ type: 'CLEAR_CART' });
    };

    return (
        <div>
            {/* UI usando dispatch */}
        </div>
    );
}
```

4.6 Best Practices

4.6.1 1. Mantieni State Minimo

Listing 4.21: State minimo necessario

```
//     State ridondante
function BadExample() {
    const [firstName, setFirstName] = useState('');
    const [lastName, setLastName] = useState('');
    const [fullName, setFullName] = useState(''); // Ridondante!

    const handleFirstNameChange = (name) => {
        setFirstName(name);
        setFullName(`${name} ${lastName}`); // Duplicazione logica
    };
}

//     Calcola fullName
function GoodExample() {
    const [firstName, setFirstName] = useState('');
    const [lastName, setLastName] = useState('');

    // Computed value
    const fullName = `${firstName} ${lastName}`.trim();

    return <h1>{fullName}</h1>;
}
```

4.6.2 2. Normalizza State Complesso

Listing 4.22: Normalizzazione state

```
//     State denormalizzato
const [data, setData] = useState({
    posts: [
        {
            id: 1,
            title: 'Post 1',
            author: { id: 10, name: 'Mario' },
            comments: [
                { id: 100, text: 'Comment 1', author: { id: 10, name: '
                    Mario' } }
            ]
        }
    ]
});

//     State normalizzato
const [data, setData] = useState({
    posts: {
        1: { id: 1, title: 'Post 1', authorId: 10, commentIds: [100] }
    },
    users: {
        10: { id: 10, name: 'Mario' }
    },
    comments: {
        100: { id: 100, text: 'Comment 1', authorId: 10 }
    }
});
```

4.6.3 3. Usa Funzione Updater

Listing 4.23: Updater function

```
//     Può causare problemi
const handleClick = () => {
    setCount(count + 1);
    setCount(count + 1); // Usa lo stesso valore di count!
    // count aumenta solo di 1, non 2
};

//     Usa updater function
const handleClick = () => {
    setCount(prev => prev + 1);
    setCount(prev => prev + 1);
    // count aumenta di 2
};
```

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

```
function Component() {
    //      CORRETTO: Top level
    const [count, setCount] = useState(0);
    const [name, setName] = useState('');

    //      SBAGLIATO: Dentro condizione
    if (count > 5) {
        const [error, setError] = useState(''); // NO!
    }

    //      SBAGLIATO: Dentro loop
    for (let i = 0; i < 10; i++) {
        const [value, setValue] = useState(i); // NO!
    }

    //      SBAGLIATO: Dentro funzione normale
    function handleClick() {
        const [temp, setTemp] = useState(0); // NO!
    }

    //      CORRETTO: Hook chiamato al top level
    useEffect(() => {
        // Effetto qui dentro è OK
    }, [count]);

    return <div>{count}</div>;
}

//      CORRETTO: Custom hook
function useCustomHook() {
    const [state, setState] = useState(0);
    return [state, setState];
}
```

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

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

function Counter() {
    // Dichiarazione
    const [count, setCount] = useState(0);
    //     ^       ^              ^
    //     |       |              |
    //  valore  setter    valore iniziale

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

5.2.2 useState con Lazy Initialization

Listing 5.3: Lazy initialization

```
function ExpensiveComponent() {
    //      Funzione eseguita ad ogni render
    const [data, setData] = useState(expensiveComputation());

    //      Funzione eseguita solo al mount
    const [data, setData] = useState(() => expensiveComputation());

    // Esempio pratico
    const [user, setUser] = useState(() => {
        const savedUser = localStorage.getItem('user');
        return savedUser ? JSON.parse(savedUser) : null;
    });

    return <div>{user?.name}</div>;
}
```

```

function expensiveComputation() {
    console.log('Computing...');
    let result = 0;
    for (let i = 0; i < 1000000; i++) {
        result += i;
    }
    return result;
}
```

5.2.3 useState con Oggetti Complessi

Listing 5.4: State oggetti

```
function UserProfile() {
    const [user, setUser] = useState({
        name: '',
        email: '',
        age: 0,
        preferences: {
            theme: 'light',
            notifications: true
        }
    });

    // Aggiorna campo top-level
    const updateName = (name) => {
        setUser(prev => ({ ...prev, name }));
    };

    // Aggiorna campo nested
    const updateTheme = (theme) => {
        setUser(prev => ({
            ...prev,
            preferences: {
                ...prev.preferences,
                theme
            }
        }));
    };

    // Aggiorna multipli campi
    const updateMultiple = () => {
        setUser(prev => ({
            ...prev,
            name: 'Mario',
            age: 30
        }));
    };

    return (
        <div>
            <input
                value={user.name}
                onChange={(e) => updateName(e.target.value)}
            />
            <select
```

```
                value={user.preferences.theme}
                onChange={(e) => updateTheme(e.target.value)}
            >
                <option value="light">Light</option>
                <option value="dark">Dark</option>
            </select>
        </div>
    );
}
```

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

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

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

    // 1. Esegue ad ogni render
    useEffect(() => {
        console.log('Eseguito ad ogni render');
    });

    // 2. Esegue solo al mount
    useEffect(() => {
        console.log('Eseguito solo al mount');
    }, []);

    // 3. Esegue quando count cambia
    useEffect(() => {
        console.log('Count è cambiato:', count);
    }, [count]);

    // 4. Con cleanup function
    useEffect(() => {
        console.log('Setup');
        return () => {
            console.log('Cleanup');
        };
    }, []);

    return <div>{count}</div>;
}
```

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

```
function UserProfile({ userId }) {
    const [user, setUser] = useState(null);
    const [loading, setLoading] = useState(true);

    // componentDidMount + componentDidUpdate (quando userId cambia)
    useEffect(() => {
        console.log('Fetching user:', userId);
        setLoading(true);

        fetch(`/api/users/${userId}`)
            .then(res => res.json())
            .then(data => {
                setUser(data);
                setLoading(false);
            });

        // componentWillUnmount / cleanup
        return () => {
            console.log('Cleanup per userId:', userId);
        };
    }, [userId]); // Re-esegue quando userId cambia

    if (loading) return <div>Loading...</div>;
    return <div>{user?.name}</div>;
}
```

5.3.3 useEffect: Data Fetching

Listing 5.7: Data fetching pattern

```
function PostsList() {
    const [posts, setPosts] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        let cancelled = false;

        const fetchPosts = async () => {
            try {
                setLoading(true);
                setError(null);

                const response = await fetch('/api/posts');
                if (!response.ok) {
                    throw new Error('Failed to fetch');
                }

                const data = await response.json();

                // Evita state update se componente è unmounted
                if (!cancelled) {
                    setPosts(data);
                    setLoading(false);
                }
            } catch (err) {
                if (!cancelled) {
                    setError(err.message);
                    setLoading(false);
                }
            }
        };

        fetchPosts();

        // Cleanup: segna come cancelled
        return () => {
            cancelled = true;
        };
    }, []);

    if (loading) return <div>Loading...</div>;
    if (error) return <div>Error: {error}</div>;

    return (
        <ul>
            {posts.map(post => (
                <li key={post.id}>{post.title}</li>
            ))}
        </ul>
    );
}
```

5.3.4 useEffect: Subscriptions

Listing 5.8: Event subscriptions

```
function WindowSize() {
    const [size, setSize] = useState({
        width: window.innerWidth,
        height: window.innerHeight
    });

    useEffect(() => {
        // Setup subscription
        const handleResize = () => {
            setSize({
                width: window.innerWidth,
                height: window.innerHeight
            });
        };

        window.addEventListener('resize', handleResize);

        // Cleanup subscription
        return () => {
            window.removeEventListener('resize', handleResize);
        };
    }, []); // Array vuoto = solo mount/unmount

    return (
        <div>
            Window size: {size.width} x {size.height}
        </div>
    );
}

// WebSocket subscription
function ChatRoom({ roomId }) {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        const ws = new WebSocket(`ws://api.example.com/chat/${roomId}`);

        ws.onmessage = (event) => {
            const message = JSON.parse(event.data);
            setMessages(prev => [...prev, message]);
        };

        ws.onerror = (error) => {
            console.error('WebSocket error:', error);
        };

        // Cleanup: chiudi connessione
        return () => {
            ws.close();
        };
    }, [roomId]); // Riconnetti quando roomId cambia

    return (
        <ul>
            {messages.map((msg, i) => (
                <li key={i}>{msg.text}</li>
            ))}
        </ul>
    );
}
```

5.3.5 useEffect: Timers

Listing 5.9: Timers e intervals

```
function Timer() {
    const [seconds, setSeconds] = useState(0);
    const [isRunning, setIsRunning] = useState(false);

    useEffect(() => {
        if (!isRunning) return;

        const interval = setInterval(() => {
            setSeconds(prev => prev + 1);
        }, 1000);

        // Cleanup: clear interval
        return () => clearInterval(interval);
    }, [isRunning]); // Re-esegue quando isRunning cambia

    return (
        <div>
            <p>Secondi: {seconds}</p>
            <button onClick={() => setIsRunning(!isRunning)}>
                {isRunning ? 'Pausa' : 'Avvia'}
            </button>
            <button onClick={() => setSeconds(0)}>Reset</button>
        </div>
    );
}

// Countdown timer
function Countdown({ initialSeconds }) {
    const [timeLeft, setTimeLeft] = useState(initialSeconds);

    useEffect(() => {
        if (timeLeft <= 0) return;

        const timeout = setTimeout(() => {
            setTimeLeft(timeLeft - 1);
        }, 1000);

        return () => clearTimeout(timeout);
    }, [timeLeft]);

    return (
        <div>
            {timeLeft > 0 ? (
                <p>Tempo rimanente: {timeLeft}s</p>
            ) : (
                <p>Tempo scaduto!</p>
            )}
        </div>
    );
}
```

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

```
import { createContext, useContext, useState } from 'react';

// 1. Crea Context
const ThemeContext = createContext();

// 2. Provider Component
export function ThemeProvider({ children }) {
    const [theme, setTheme] = useState('light');

    const toggleTheme = () => {
        setTheme(prev => prev === 'light' ? 'dark' : 'light');
    };

    const value = {
        theme,
        setTheme,
        toggleTheme
    };

    return (
        <ThemeContext.Provider value={value}>
            {children}
        </ThemeContext.Provider>
    );
}

// 3. Custom Hook per consumare Context
export function useTheme() {
    const context = useContext(ThemeContext);
    if (context === undefined) {
        throw new Error('useTheme must be used within ThemeProvider');
    }
    return context;
}

// 4. Uso in App
function App() {
    return (
        <ThemeProvider>
            <Navbar />
            <Content />
        </ThemeProvider>
    );
}

// 5. Consume Context in qualsiasi componente
function Navbar() {
    const { theme, toggleTheme } = useTheme();

    return (
        <nav className={`navbar-${theme}`}>
            <button onClick={toggleTheme}>
                Tema: {theme}
            </button>
        </nav>
    );
}
```

```

function Content() {
    const { theme } = useTheme();

    return (
        <div className={`content-${theme}`}>
            <p>Contenuto con tema {theme}</p>
        </div>
    );
}
```

5.4.2 Context Multipli

Listing 5.11: Multipli context providers

```
// contexts/AuthContext.js
const AuthContext = createContext();

export function AuthProvider({ children }) {
    const [user, setUser] = useState(null);

    const login = async (credentials) => {
        const userData = await api.login(credentials);
        setUser(userData);
    };

    const logout = () => {
        setUser(null);
    };

    return (
        <AuthContext.Provider value={{ user, login, logout }}>
            {children}
        </AuthContext.Provider>
    );
}

export const useAuth = () => useContext(AuthContext);

// contexts/SettingsContext.js
const SettingsContext = createContext();

export function SettingsProvider({ children }) {
    const [settings, setSettings] = useState({
        language: 'it',
        notifications: true
    });

    const updateSettings = (updates) => {
        setSettings(prev => ({ ...prev, ...updates }));
    };

    return (
        <SettingsContext.Provider value={{ settings, updateSettings }}>
            {children}
        </SettingsContext.Provider>
    );
}

export const useSettings = () => useContext(SettingsContext);

// App.jsx - Componi multipli providers
function App() {
    return (
        <AuthProvider>
            <SettingsProvider>
                <ThemeProvider>
                    <Router>
                        <Routes />
                    </Router>
                </ThemeProvider>
            </SettingsProvider>
        </AuthProvider>
    );
}

// Uso in componenti
function UserDashboard() {
    const { user, logout } = useAuth();
    const { settings } = useSettings();
    const { theme } = useTheme();

    return (
        <div className={theme}>
            <h1>Ciao, {user.name}</h1>
            <p>Lingua: {settings.language}</p>
            <button onClick={logout}>Logout</button>
        </div>
    );
}
```

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

```
import { useRef, useEffect } from 'react';

function FocusInput() {
    const inputRef = useRef(null);

    useEffect(() => {
        // Focus automatico al mount
        inputRef.current.focus();
    }, []);

    const handleFocus = () => {
        inputRef.current.focus();
    };

    return (
        <div>
            <input ref={inputRef} type="text" />
            <button onClick={handleFocus}>Focus Input</button>
        </div>
    );
}

// Scroll to element
function ScrollToSection() {
    const sectionRef = useRef(null);

    const scrollToSection = () => {
        sectionRef.current.scrollIntoView({ behavior: 'smooth' });
    };

    return (
        <div>
            <button onClick={scrollToSection}>Vai alla sezione</button>
            <div style={{ height: '1000px' }}>Contenuto...</div>
            <div ref={sectionRef}>
                <h2>Sezione Target</h2>
            </div>
        </div>
    );
}

// Media playback control
function VideoPlayer({ src }) {
    const videoRef = useRef(null);
    const [isPlaying, setIsPlaying] = useState(false);

    const togglePlay = () => {
        if (isPlaying) {
            videoRef.current.pause();
        } else {
            videoRef.current.play();
        }
        setIsPlaying(!isPlaying);
    };

    return (
        <div>
            <video ref={videoRef} src={src} />
            <button onClick={togglePlay}>
                {isPlaying ? 'Pausa' : 'Play'}
            </button>
        </div>
    );
}
```

5.5.2 useRef per Valori Mutabili

Listing 5.13: useRef per valori persistenti

```
function Timer() {
    const [seconds, setSeconds] = useState(0);
    const intervalRef = useRef(null);

    const start = () => {
        if (intervalRef.current) return; // Già in esecuzione

        intervalRef.current = setInterval(() => {
            setSeconds(prev => prev + 1);
        }, 1000);
    };

    const stop = () => {
        clearInterval(intervalRef.current);
        intervalRef.current = null;
    };

    const reset = () => {
        stop();
        setSeconds(0);
    };

    useEffect(() => {
        // Cleanup on unmount
        return () => {
            if (intervalRef.current) {
                clearInterval(intervalRef.current);
            }
        };
    }, []);

    return (
        <div>
            <p>Secondi: {seconds}</p>
            <button onClick={start}>Start</button>
            <button onClick={stop}>Stop</button>
            <button onClick={reset}>Reset</button>
        </div>
    );
}

// Tracciare valore precedente
function usePrevious(value) {
    const ref = useRef();

    useEffect(() => {
        ref.current = value;
    }, [value]);

    return ref.current;
}

// Uso
function Counter() {
    const [count, setCount] = useState(0);
    const prevCount = usePrevious(count);

    return (
        <div>
            <p>Attuale: {count}</p>
            <p>Precedente: {prevCount}</p>
            <button onClick={() => setCount(count + 1)}>
                Incrementa
            </button>
        </div>
    );
}

// Evitare re-render inutili
function SearchComponent() {
    const [query, setQuery] = useState('');
    const timeoutRef = useRef(null);

    const handleSearch = (value) => {
        setQuery(value);

        // Debounce: cancella timeout precedente
        clearTimeout(timeoutRef.current);

        timeoutRef.current = setTimeout(() => {
            console.log('Searching for:', value);
            // API call...
        }, 500);
    };

    return (
        <input
            value={query}
            onChange={(e) => handleSearch(e.target.value)}
        />
    );
}
```

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

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

function ExpensiveList({ items, filter }) {
    //     Senza useMemo: filtra ad ogni render
    const filteredItems = items.filter(item =>
        item.name.includes(filter)
    );

    //     Con useMemo: filtra solo quando items o filter cambiano
    const filteredItems = useMemo(() => {
        console.log('Filtering items...');
        return items.filter(item => item.name.includes(filter));
    }, [items, filter]);

    return (
        <ul>
            {filteredItems.map(item => (
                <li key={item.id}>{item.name}</li>
            ))}
        </ul>
    );
}

// Calcoli complessi
function DataAnalysis({ data }) {
    const statistics = useMemo(() => {
        console.log('Calculating statistics...');

        const sum = data.reduce((acc, val) => acc + val, 0);
        const avg = sum / data.length;
        const min = Math.min(...data);
        const max = Math.max(...data);

        return { sum, avg, min, max };
    }, [data]);

    return (
        <div>
            <p>Somma: {statistics.sum}</p>
            <p>Media: {statistics.avg}</p>
            <p>Min: {statistics.min}</p>
            <p>Max: {statistics.max}</p>
        </div>
    );
}
```

5.6.2 useMemo per Reference Equality

Listing 5.15: useMemo per stabilità riferimenti

```
function Parent() {
    const [count, setCount] = useState(0);
    const [name, setName] = useState('Mario');

    //      Nuovo oggetto ad ogni render
    const user = {
        name: name,
        age: 30
    };

    //      Stesso oggetto se name non cambia
    const user = useMemo(() => ({
        name: name,
        age: 30
    }), [name]);

    // Child non re-renderizza inutilmente quando count cambia
    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={() => setCount(count + 1)}>Inc</button>
            <ExpensiveChild user={user} />
        </div>
    );
}

const ExpensiveChild = React.memo(({ user }) => {
    console.log('ExpensiveChild render');
    return <div>{user.name}</div>;
});
```

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

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

function Parent() {
    const [count, setCount] = useState(0);
    const [name, setName] = useState('Mario');

    //      Nuova funzione ad ogni render
    const handleClick = () => {
        console.log('Clicked');
    };

    //      Stessa funzione tra i render
    const handleClick = useCallback(() => {
        console.log('Clicked');
    }, []); // Nessuna dipendenza

    //      Con dipendenze
    const handleNameChange = useCallback((newName) => {
        console.log('Old name:', name);
        setName(newName);
    }, [name]); // Ricreata solo quando name cambia

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={() => setCount(count + 1)}>Inc</button>
            <ExpensiveButton onClick={handleClick} />
        </div>
    );
}

const ExpensiveButton = React.memo(({ onClick }) => {
    console.log('ExpensiveButton render');
    return <button onClick={onClick}>Click me</button>;
});
```

5.7.2 useCallback vs useMemo

Listing 5.17: Differenza useCallback vs useMemo

```
function Component() {
    // useCallback: memoizza la FUNZIONE
    const handleClick = useCallback(() => {
        console.log('Clicked');
    }, []);

    // useMemo: memoizza il RISULTATO della funzione
    const value = useMemo(() => {
        return expensiveCalculation();
    }, []);

    // Equivalenze:
    // useCallback(fn, deps) === useMemo(() => fn, deps)

    const fn1 = useCallback(() => console.log('Hi'), []);
    const fn2 = useMemo(() => () => console.log('Hi'), []);
    // fn1 === fn2

    return <div />;
}
```

5.7.3 useCallback: Casi d'Uso

Listing 5.18: useCallback pattern comuni

```
function TodoList() {
    const [todos, setTodos] = useState([]);

    // Callback passata a child
    const handleToggle = useCallback((id) => {
        setTodos(prev => prev.map(todo =>
            todo.id === id
                ? { ...todo, completed: !todo.completed }
                : todo
        ));
    }, []); // Usa funzione updater, nessuna dipendenza

    const handleDelete = useCallback((id) => {
        setTodos(prev => prev.filter(todo => todo.id !== id));
    }, []);

    return (
        <div>
            {todos.map(todo => (
                <TodoItem
                    key={todo.id}
                    todo={todo}
                    onToggle={handleToggle}
                    onDelete={handleDelete}
                />
            ))}
        </div>
    );
}
```

```

const TodoItem = React.memo(({ todo, onToggle, onDelete }) => {
    console.log('TodoItem render:', todo.id);

    return (
        <div>
            <input
                type="checkbox"
                checked={todo.completed}
                onChange={() => onToggle(todo.id)}
            />
            <span>{todo.text}</span>
            <button onClick={() => onDelete(todo.id)}>Delete</button>
        </div>
    );
});
```

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

```
// hooks/useCounter.js
import { useState } from 'react';

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

    const increment = () => setCount(prev => prev + 1);
    const decrement = () => setCount(prev => prev - 1);
    const reset = () => setCount(initialValue);

    return { count, increment, decrement, reset };
}

// Uso
function Counter() {
    const { count, increment, decrement, reset } = useCounter(0);

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={decrement}>-</button>
            <button onClick={reset}>Reset</button>
            <button onClick={increment}>+</button>
        </div>
    );
}
```

5.8.2 useLocalStorage

Listing 5.20: Custom hook localStorage

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

function useLocalStorage(key, initialValue) {
    // State con lazy initialization
    const [storedValue, setStoredValue] = useState(() => {
        try {
            const item = window.localStorage.getItem(key);
            return item ? JSON.parse(item) : initialValue;
        } catch (error) {
            console.error(error);
            return initialValue;
        }
    });

    // Aggiorna localStorage quando il valore cambia
    useEffect(() => {
        try {
            window.localStorage.setItem(key, JSON.stringify(storedValue)
                );
        } catch (error) {
            console.error(error);
        }
    }, [key, storedValue]);

    return [storedValue, setStoredValue];
}

// Uso
function App() {
    const [name, setName] = useLocalStorage('name', 'Guest');
    const [theme, setTheme] = useLocalStorage('theme', 'light');

    return (
        <div>
            <input
                value={name}
                onChange={(e) => setName(e.target.value)}
            />
            <p>Nome salvato: {name}</p>

            <button onClick={() => setTheme(theme === 'light' ? 'dark' :
                'light')}>
                Tema: {theme}
            </button>
        </div>
    );
}
```

5.8.3 useFetch

Listing 5.21: Custom hook data fetching

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

function useFetch(url, options = {}) {
    const [data, setData] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    useEffect(() => {
        let cancelled = false;

        const fetchData = async () => {
            try {
                setLoading(true);
                setError(null);

                const response = await fetch(url, options);

                if (!response.ok) {
                    throw new Error(‘HTTP error! status: ${response.status}‘);
                }

                const json = await response.json();

                if (!cancelled) {
                    setData(json);
                    setLoading(false);
                }
            } catch (err) {
                if (!cancelled) {
                    setError(err.message);
                    setLoading(false);
                }
            }
        };

        fetchData();

        return () => {
            cancelled = true;
        };
    }, [url]);

    return { data, loading, error };
}

// Uso
function UserProfile({ userId }) {
    const { data: user, loading, error } = useFetch(‘/api/users/${userId}‘);

    if (loading) return <div>Loading...</div>;
    if (error) return <div>Error: {error}</div>;

    return (
        <div>
            <h1>{user.name}</h1>
            <p>{user.email}</p>
        </div>
    );
}
```

5.8.4 useDebounce

Listing 5.22: Custom hook debounce

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

function useDebounce(value, delay = 500) {
    const [debouncedValue, setDebouncedValue] = useState(value);

    useEffect(() => {
        const timer = setTimeout(() => {
            setDebouncedValue(value);
        }, delay);

        return () => {
            clearTimeout(timer);
        };
    }, [value, delay]);

    return debouncedValue;
}

// Uso
function SearchComponent() {
    const [searchTerm, setSearchTerm] = useState('');
    const debouncedSearchTerm = useDebounce(searchTerm, 500);

    useEffect(() => {
        if (debouncedSearchTerm) {
            console.log('Searching for:', debouncedSearchTerm);
            // API call...
        }
    }, [debouncedSearchTerm]);

    return (
        <input
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
            placeholder="Cerca..."
        />
    );
}
```

5.8.5 useToggle

Listing 5.23: Custom hook toggle

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

function useToggle(initialValue = false) {
    const [value, setValue] = useState(initialValue);

    const toggle = useCallback(() => {
        setValue(prev => !prev);
    }, []);

    const setTrue = useCallback(() => {
        setValue(true);
    }, []);

    const setFalse = useCallback(() => {
        setValue(false);
    }, []);

    return [value, toggle, setTrue, setFalse];
}

// Uso
function Modal() {
    const [isOpen, toggle, open, close] = useToggle(false);

    return (
        <div>
            <button onClick={open}>Apri Modal</button>

            {isOpen && (
                <div className="modal">
                    <h2>Modal</h2>
                    <button onClick={close}>Chiudi</button>
                    <button onClick={toggle}>Toggle</button>
                </div>
            )}
        </div>
    );
}
```

5.8.6 useWindowSize

Listing 5.24: Custom hook window size

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

function useWindowSize() {
    const [windowSize, setWindowSize] = useState({
        width: window.innerWidth,
        height: window.innerHeight
    });

    useEffect(() => {
        const handleResize = () => {
            setWindowSize({
                width: window.innerWidth,
                height: window.innerHeight
            });
        };

        window.addEventListener('resize', handleResize);
        return () => window.removeEventListener('resize', handleResize);
    }, []);

    return windowSize;
}

// Uso
function ResponsiveComponent() {
    const { width, height } = useWindowSize();

    return (
        <div>
            <p>Window size: {width} x {height}</p>
            {width < 768 ? (
                <MobileLayout />
            ) : (
                <DesktopLayout />
            )}
        </div>
    );
}
```

5.8.7 useClickOutside

Listing 5.25: Custom hook click outside

```
import { useEffect, useRef } from 'react';

function useClickOutside(callback) {
    const ref = useRef(null);

    useEffect(() => {
        const handleClick = (event) => {
            if (ref.current && !ref.current.contains(event.target)) {
                callback();
            }
        };

        document.addEventListener('mousedown', handleClick);
        return () => {
            document.removeEventListener('mousedown', handleClick);
        };
    }, [callback]);

    return ref;
}

// Uso
function Dropdown() {
    const [isOpen, setIsOpen] = useState(false);
    const dropdownRef = useClickOutside(() => setIsOpen(false));

    return (
        <div ref={dropdownRef}>
            <button onClick={() => setIsOpen(!isOpen)}>
                Menu
            </button>

            {isOpen && (
                <ul className="dropdown-menu">
                    <li>Option 1</li>
                    <li>Option 2</li>
                    <li>Option 3</li>
                </ul>
            )}
        </div>
    );
}
```

5.9 Hook Composition

Combina hooks multipli per funzionalità complesse.

Listing 5.26: Composizione di hooks

```
// Custom hook che usa altri custom hooks
function useForm(initialValues, validate) {
    const [values, setValues] = useState(initialValues);
    const [errors, setErrors] = useState({});
    const [touched, setTouched] = useState({});
    const [isSubmitting, setIsSubmitting] = useState(false);

    // Usa useCallback
    const handleChange = useCallback((e) => {
        const { name, value } = e.target;
        setValues(prev => ({ ...prev, [name]: value }));
    }, []);

    const handleBlur = useCallback((e) => {
        const { name } = e.target;
        setTouched(prev => ({ ...prev, [name]: true }));
    }, []);

    // Usa useEffect
    useEffect(() => {
        if (Object.keys(touched).length > 0) {
            const validationErrors = validate(values);
            setErrors(validationErrors);
        }
    }, [values, touched, validate]);

    const handleSubmit = useCallback(async (onSubmit) => {
        setIsSubmitting(true);
        const validationErrors = validate(values);

        if (Object.keys(validationErrors).length === 0) {
            await onSubmit(values);
        }

        setIsSubmitting(false);
    }, [values, validate]);

    return {
        values,
        errors,
        touched,
        isSubmitting,
        handleChange,
        handleBlur,
        handleSubmit
    };
}

```

```
// Uso
function LoginForm() {
    const validate = (values) => {
        const errors = {};
        if (!values.email) errors.email = 'Email richiesta';
        if (!values.password) errors.password = 'Password richiesta';
        return errors;
    };

    const form = useForm(
        { email: '', password: '' },
        validate
    );

    const handleSubmit = (e) => {
        e.preventDefault();
        form.handleSubmit(async (values) => {
            console.log('Login:', values);
            // API call...
        });
    };

    return (
        <form onSubmit={handleSubmit}>
            <input
                name="email"
                value={form.values.email}
                onChange={form.handleChange}
                onBlur={form.handleBlur}
            />
            {form.touched.email && form.errors.email && (
                <span>{form.errors.email}</span>
            )}

            <input
                type="password"
                name="password"
                value={form.values.password}
                onChange={form.handleChange}
                onBlur={form.handleBlur}
            />
            {form.touched.password && form.errors.password && (
                <span>{form.errors.password}</span>
            )}

            <button type="submit" disabled={form.isSubmitting}>
                Login
            </button>
        </form>
    );
}
```

5.10 Best Practices

5.10.1 1. Naming Convention

Listing 5.27: Naming dei custom hooks

```
//      Inizia sempre con "use"
function useCounter() {}
function useFetch() {}
function useLocalStorage() {}

//      Non usare "use" per funzioni normali
function useHelper() {}  // Se non è un hook, non chiamarlo "use..."
function formatDate() {} //      Meglio
```

5.10.2 2. Dependency Array

Listing 5.28: Gestione dependencies

```
function Component({ userId }) {
    const [user, setUser] = useState(null);

    //      Missing dependency
    useEffect(() => {
        fetchUser(userId).then(setUser);
    }, []); // userId dovrebbe essere nelle deps!

    //      Include tutte le dependencies
    useEffect(() => {
        fetchUser(userId).then(setUser);
    }, [userId]);

    //      Oppure usa ESLint plugin
    // eslint-plugin-react-hooks ti avviserà
}
```

5.10.3 3. Evita Premature Optimization

Listing 5.29: Non abusare di useMemo/useCallback

```
//      Overuse di useMemo
function Component() {
    const value = useMemo(() => 2 + 2, []); // Inutile!
    const name = useMemo(() => 'Mario', []); // Inutile!

    //      Usa solo per calcoli costosi
    const expensiveValue = useMemo(() => {
        return data.reduce((acc, item) => {
            // Calcolo complesso...
        }, 0);
    }, [data]);
}
```

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

```
//      DOM nativo (HTML)
<button onclick="handleClick()">Click me</button>

//      React (JSX)
<button onClick={handleClick}>Click me</button>

// Differenze principali:
// 1. camelCase invece di lowercase (onClick vs onclick)
// 2. Funzione invece di stringa ({handleClick} vs "handleClick()")
// 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

```
function Component() {
    const handleClick = (event) => {
        // event è un SyntheticEvent
        console.log(event);

        // Proprietà comuni
        console.log(event.type);             // "click"
        console.log(event.target);           // Elemento DOM che ha generato l'evento
        console.log(event.currentTarget);    // Elemento con l'handler
        console.log(event.timeStamp);        // Timestamp
        console.log(event.bubbles);          // true/false

        // Metodi
        event.preventDefault();  // Previeni comportamento default
        event.stopPropagation(); // Ferma propagazione

        // Accesso all'evento nativo
        const nativeEvent = event.nativeEvent;
        console.log(nativeEvent);
    };

    return <button onClick={handleClick}>Click</button>;
}
```

6.2.2 Event Pooling (React < 17)

Listing 6.3: Event pooling in React 16

```
// In React 16 e precedenti, gli eventi venivano "pooled"
function OldComponent() {
    const handleClick = (event) => {
        console.log(event.type); // "click"

        setTimeout(() => {
            //     event è null qui!
            console.log(event.type); // null
        }, 1000);

        //     Soluzione: persist()
        event.persist();
        setTimeout(() => {
            console.log(event.type); // "click"
        }, 1000);
    };

    return <button onClick={handleClick}>Click</button>;
}

// React 17+: Event pooling rimosso!
function NewComponent() {
    const handleClick = (event) => {
        setTimeout(() => {
            //     Funziona in React 17+
            console.log(event.type); // "click"
        }, 1000);
    };

    return <button onClick={handleClick}>Click</button>;
}
```

6.3 onClick: Eventi Click

onClick è l'evento più comune in React.

6.3.1 onClick Base

Listing 6.4: onClick esempi

```
function ClickExamples() {
    // 1. Inline arrow function
    return (
        <button onClick={() => console.log('Clicked!')}>
            Click 1
        </button>
    );

    // 2. Named function
    const handleClick = () => {
        console.log('Clicked!');
    };

    return <button onClick={handleClick}>Click 2</button>;

    // 3. Con parametri
    const handleClickWithParam = (id) => {
        console.log('Clicked item:', id);
    };

    return (
        <button onClick={() => handleClickWithParam(123)}>
            Click 3
        </button>
    );

    // 4. Con event object
    const handleClickWithEvent = (e) => {
        console.log('Clicked:', e.target);
    };

    return <button onClick={handleClickWithEvent}>Click 4</button>;

    // 5. Con event e parametri
    const handleClickBoth = (id, e) => {
        console.log('ID:', id, 'Event:', e);
    };

    return (
        <button onClick={(e) => handleClickBoth(123, e)}>
            Click 5
        </button>
    );
}
```

6.3.2 onClick Patterns

Listing 6.5: Pattern comuni onClick

```
function ClickPatterns() {
    const [count, setCount] = useState(0);
    const [items, setItems] = useState(['A', 'B', 'C']);

    // Pattern 1: Aggiornamento state
    const increment = () => {
        setCount(count + 1);
    };

    // Pattern 2: Toggle boolean
    const [isOpen, setIsOpen] = useState(false);
    const toggle = () => setIsOpen(!isOpen);

    // Pattern 3: Handler con parametro da lista
    const handleItemClick = (item) => {
        console.log('Clicked:', item);
    };

    // Pattern 4: Prevent default
    const handleLinkClick = (e) => {
        e.preventDefault();
        console.log('Link clicked but not followed');
    };

    // Pattern 5: Conditional handler
    const handleConditionalClick = () => {
        if (count < 10) {
            setCount(count + 1);
        } else {
            alert('Limite raggiunto!');
        }
    };

    return (
        <div>
            {/* Pattern 1 */}
            <button onClick={increment}>Count: {count}</button>

            {/* Pattern 2 */}
            <button onClick={toggle}>
                {isOpen ? 'Chiudi' : 'Apri'}
            </button>

            {/* Pattern 3 */}
            {items.map(item => (
                <button key={item} onClick={() => handleItemClick(item)
                    }>
                    {item}
                </button>
            ))}

            {/* Pattern 4 */}
            <a href="https://example.com" onClick={handleLinkClick}>
                Link
            </a>

            {/* Pattern 5 */}
            <button onClick={handleConditionalClick}>
```

```
                Incrementa (max 10)
            </button>
        </div>
    );
}
```

6.3.3 Gestione Click Multipli

Listing 6.6: Double click e timing

```
function MultiClickExample() {
    const [clicks, setClicks] = useState(0);
    const [lastClick, setLastClick] = useState(0);

    // Single click
    const handleClick = () => {
        setClicks(prev => prev + 1);
    };

    // Double click
    const handleDoubleClick = () => {
        console.log('Double clicked!');
        setClicks(0); // Reset
    };

    // Debounced click
    const timeoutRef = useRef(null);

    const handleDebouncedClick = () => {
        clearTimeout(timeoutRef.current);

        timeoutRef.current = setTimeout(() => {
            console.log('Clicked (debounced)');
        }, 300);
    };

    // Click con delay detection
    const handleClickWithDelay = () => {
        const now = Date.now();
        const timeSinceLastClick = now - lastClick;

        if (timeSinceLastClick < 300) {
            console.log('Quick click!');
        } else {
            console.log('Slow click!');
        }

        setLastClick(now);
    };

    return (
        <div>
            <button onClick={handleClick} onDoubleClick={
                handleDoubleClick}>
                Clicks: {clicks} (double click to reset)
            </button>

            <button onClick={handleDebouncedClick}>
                Debounced Click
            </button>

            <button onClick={handleClickWithDelay}>
                Click with timing
            </button>
        </div>
    );
}
```

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

```
function InputExamples() {
    const [text, setText] = useState('');
    const [number, setNumber] = useState(0);
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');

    return (
        <div>
            {/* Text input */}
            <input
                type="text"
                value={text}
                onChange={(e) => setText(e.target.value)}
                placeholder="Nome"
            />
            <p>Testo: {text}</p>

            {/* Number input */}
            <input
                type="number"
                value={number}
                onChange={(e) => setNumber(Number(e.target.value))}
                placeholder="Età"
            />
            <p>Numero: {number}</p>

            {/* Email input con validazione */}
            <input
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                placeholder="Email"
            />
            <p style={{ color: email.includes('@') ? 'green' : 'red' }}>
                Email: {email}
            </p>

            {/* Password input */}
            <input
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                placeholder="Password"
            />
            <p>Password length: {password.length}</p>
        </div>
    );
}
```

6.4.2 onChange con Select

Listing 6.8: onChange select

```
function SelectExamples() {
    const [country, setCountry] = useState('');
    const [language, setLanguage] = useState('it');
    const [categories, setCategories] = useState([]);

    return (
        <div>
            {/* Select singolo */}
            <select value={country} onChange={(e) => setCountry(e.target.value)}>
                <option value="">Seleziona paese...</option>
                <option value="IT">Italia</option>
                <option value="US">USA</option>
                <option value="UK">UK</option>
            </select>
            <p>Paese selezionato: {country}</p>

            {/* Select con default */}
            <select value={language} onChange={(e) => setLanguage(e.target.value)}>
                <option value="it">Italiano</option>
                <option value="en">English</option>
                <option value="es">Espa ol</option>
            </select>
            <p>Lingua: {language}</p>

            {/* Select multiplo */}
            <select
                multiple
                value={categories}
                onChange={(e) => {
                    const selected = Array.from(
                        e.target.selectedOptions,
                        option => option.value
                    );
                    setCategories(selected);
                }}
            >
                <option value="tech">Tech</option>
                <option value="sport">Sport</option>
                <option value="music">Musica</option>
                <option value="travel">Viaggi</option>
            </select>
            <p>Categorie: {categories.join(', ')}</p>
        </div>
    );
}
```

6.4.3 onChange con Checkbox e Radio

Listing 6.9: onChange checkbox/radio

```
function CheckboxRadioExamples() {
    const [isChecked, setIsChecked] = useState(false);
    const [gender, setGender] = useState('');
    const [interests, setInterests] = useState([]);

    // Checkbox singola
    const handleCheckboxChange = (e) => {
        setIsChecked(e.target.checked);
    };

    // Radio buttons
    const handleRadioChange = (e) => {
        setGender(e.target.value);
    };

    // Checkbox multiple
    const handleInterestChange = (interest) => {
        setInterests(prev =>
            prev.includes(interest)
                ? prev.filter(i => i !== interest)
                : [...prev, interest]
        );
    };

    return (
        <div>
            {/* Checkbox singola */}
            <label>
                <input
                    type="checkbox"
                    checked={isChecked}
                    onChange={handleCheckboxChange}
                />
                Accetto i termini
            </label>
            <p>Accettato: {isChecked ? 'Sì' : 'No'}</p>

            {/* Radio buttons */}
            <div>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="male"
                        checked={gender === 'male'}
                        onChange={handleRadioChange}
                    />
                    Maschio
                </label>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="female"
                        checked={gender === 'female'}
                        onChange={handleRadioChange}
                    />
                    Femmina
                </label>
            </div>
            <p>Genere: {gender}</p>

            {/* Checkbox multiple */}
            <div>
                {['Sport', 'Musica', 'Lettura', 'Viaggi'].map(interest
                   => (
                    <label key={interest}>
                        <input
                            type="checkbox"
                            checked={interests.includes(interest)}
                            onChange={() => handleInterestChange(
                               interest)}
                        />
                        {interest}
                    </label>
                ))}
            </div>
            <p>Interessi: {interests.join(', ')}</p>
        </div>
    );
}
```

6.4.4 onChange con Textarea

Listing 6.10: onChange textarea

```
function TextareaExample() {
    const [message, setMessage] = useState('');
    const maxLength = 200;

    const handleChange = (e) => {
        const value = e.target.value;

        // Limita lunghezza
        if (value.length <= maxLength) {
            setMessage(value);
        }
    };

    const remainingChars = maxLength - message.length;

    return (
        <div>
```

```
            <textarea
                value={message}
                onChange={handleChange}
                placeholder="Scrivi un messaggio..."
                rows={5}
                cols={50}
            />

            <p>
                Caratteri rimanenti: {remainingChars} / {maxLength}
            </p>

            <p>Righe: {message.split('\n').length}</p>

            <p style={{ color: remainingChars < 20 ? 'red' : 'black' }}>
                {remainingChars < 20 && 'Stai raggiungendo il limite!'}
            </p>
        </div>
    );
}
```

6.5 Altri Eventi Comuni

6.5.1 onSubmit: Form Submission

Listing 6.11: onSubmit form

```
function FormSubmitExample() {
    const [formData, setFormData] = useState({
        username: '',
        email: '',
        password: ''
    });

    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData(prev => ({
            ...prev,
            [name]: value
        }));
    };

    const handleSubmit = (e) => {
        e.preventDefault(); //        IMPORTANTE: Previeni reload pagina

        console.log('Form submitted:', formData);

        // Validazione
        if (!formData.username || !formData.email || !formData.password)
            {
            alert('Compila tutti i campi!');
            return;
        }

        // Invia dati
        fetch('/api/register', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(formData)
        })
        .then(response => response.json())
        .then(data => {
            console.log('Success:', data);
            // Reset form
            setFormData({ username: '', email: '', password: '' });
        });
    };

    return (
        <form onSubmit={handleSubmit}>
            <input
                name="username"
                value={formData.username}
                onChange={handleChange}
                placeholder="Username"
            />

            <input
                name="email"
                type="email"
                value={formData.email}
                onChange={handleChange}
                placeholder="Email"
            />

            <input
                name="password"
                type="password"
                value={formData.password}
                onChange={handleChange}
                placeholder="Password"
            />

            <button type="submit">Registrati</button>
        </form>
    );
}
```

6.5.2 onFocus e onBlur

Listing 6.12: onFocus e onBlur

```
function FocusBlurExample() {
    const [isFocused, setIsFocused] = useState(false);
    const [value, setValue] = useState('');
    const [touched, setTouched] = useState(false);

    const handleFocus = () => {
        setIsFocused(true);
    };

    const handleBlur = () => {
        setIsFocused(false);
        setTouched(true);
    };

    // Validazione solo dopo blur
    const error = touched && value.length < 3
        ? 'Minimo 3 caratteri'
        : '';

    return (
        <div>
            <input
                value={value}
                onChange={(e) => setValue(e.target.value)}
                onFocus={handleFocus}
                onBlur={handleBlur}
                placeholder="Nome"
                style={{
                    borderColor: isFocused ? 'blue' : error ? 'red' : '
                        gray'
                }}
            />

            <p>Focus: {isFocused ? 'Sì' : 'No'}</p>

            {error && <span style={{ color: 'red' }}>{error}</span>}
        </div>
    );
}
```

6.5.3 onKeyDown, onKeyUp, onKeyPress

Listing 6.13: Eventi keyboard

```
function KeyboardEvents() {
    const [input, setInput] = useState('');
    const [lastKey, setLastKey] = useState('');

    const handleKeyDown = (e) => {
        setLastKey(e.key);

        // Enter per submit
        if (e.key === 'Enter') {
            console.log('Enter pressed!');
        }

        // Escape per cancellare
        if (e.key === 'Escape') {
            setInput('');
        }

        // Ctrl + S per salvare
        if (e.ctrlKey && e.key === 's') {
            e.preventDefault();
            console.log('Saving...');
        }

        // Arrow keys
        if (e.key === 'ArrowUp') {
            console.log('Up arrow');
        }
    };

    const handleKeyPress = (e) => {
        // Solo caratteri alfanumerici
        if (!/[a-zA-Z0-9]/.test(e.key)) {
            e.preventDefault();
        }
    };

    return (
        <div>
            <input
                value={input}
                onChange={(e) => setInput(e.target.value)}
                onKeyDown={handleKeyDown}
                onKeyPress={handleKeyPress}
                placeholder="Prova i tasti..."
            />

            <p>Ultimo tasto: {lastKey}</p>
            <p>Input: {input}</p>

            <ul>
                <li>Enter: Submit</li>
                <li>Escape: Cancella</li>
                <li>Ctrl+S: Salva</li>
            </ul>
        </div>
    );
}
```

6.5.4 onMouseEnter e onMouseLeave

Listing 6.14: Eventi mouse hover

```
function HoverExample() {
    const [isHovering, setIsHovering] = useState(false);
    const [hoverCount, setHoverCount] = useState(0);
    const [position, setPosition] = useState({ x: 0, y: 0 });

    const handleMouseEnter = () => {
        setIsHovering(true);
        setHoverCount(prev => prev + 1);
    };

    const handleMouseLeave = () => {
        setIsHovering(false);
    };

    const handleMouseMove = (e) => {
        // Posizione relativa all'elemento
        const rect = e.currentTarget.getBoundingClientRect();
        setPosition({
            x: e.clientX - rect.left,
            y: e.clientY - rect.top
        });
    };

    return (
        <div>
            <div
                onMouseEnter={handleMouseEnter}
                onMouseLeave={handleMouseLeave}
                onMouseMove={handleMouseMove}
                style={{
                    width: '300px',
                    height: '200px',
                    backgroundColor: isHovering ? 'lightblue' : '
                       lightgray',
                    display: 'flex',
                    alignItems: 'center',
                    justifyContent: 'center',
                    cursor: 'pointer',
                    transition: 'background-color 0.3s'
                }}
            >
                <div>
                    <p>Hovering: {isHovering ? 'Sì' : 'No'}</p>
                    <p>Hover count: {hoverCount}</p>
                    <p>Position: ({position.x.toFixed(0)}, {position.y.
                       toFixed(0)})</p>
                </div>
            </div>
        </div>
    );
}
```

6.5.5 onScroll

Listing 6.15: Evento scroll

```
function ScrollExample() {
    const [scrollPosition, setScrollPosition] = useState(0);
    const [isTop, setIsTop] = useState(true);
    const scrollRef = useRef(null);

    const handleScroll = (e) => {
        const scrollTop = e.target.scrollTop;
        setScrollPosition(scrollTop);
        setIsTop(scrollTop === 0);
    };

    // Window scroll
    useEffect(() => {
        const handleWindowScroll = () => {
            setScrollPosition(window.scrollY);
        };

        window.addEventListener('scroll', handleWindowScroll);
        return () => window.removeEventListener('scroll',
           handleWindowScroll);
    }, []);

    return (
        <div>
            {/* Scrollable container */}
            <div
                ref={scrollRef}
                onScroll={handleScroll}
                style={{
                    height: '300px',
                    overflow: 'auto',
                    border: '1px solid black'
                }}
            >
                <div style={{ height: '1000px', padding: '20px' }}>
                    <p>Scroll position: {scrollPosition}px</p>
                    <p>At top: {isTop ? 'Sì' : 'No'}</p>
                    <p>Contenuto scrollabile...</p>
                </div>
            </div>
        </div>
    );
}
```

6.6 Event Delegation e Bubbling

6.6.1 Event Bubbling

Listing 6.16: Event bubbling

```
function BubblingExample() {
    const handleParentClick = (e) => {
        console.log('Parent clicked');
        console.log('Target:', e.target.tagName);         // Elemento
            cliccato
        console.log('CurrentTarget:', e.currentTarget.tagName); //
            Elemento con handler
    };

    const handleChildClick = (e) => {
        console.log('Child clicked');

        // Ferma la propagazione
        // e.stopPropagation();
    };

    return (
        <div
            onClick={handleParentClick}
            style={{ padding: '20px', backgroundColor: 'lightblue' }}
        >
            Parent
            <button onClick={handleChildClick}>
                Child (clicca qui)
            </button>
        </div>
    );
    // Se clicchi il button, vedi:
```

```
    // "Child clicked"
    // "Parent clicked" (bubbling!)
}
```

6.6.2 Event Delegation

Listing 6.17: Event delegation pattern

```
function TodoList() {
    const [todos, setTodos] = useState([
        { id: 1, text: 'Todo 1' },
        { id: 2, text: 'Todo 2' },
        { id: 3, text: 'Todo 3' }
    ]);

    //      Handler per ogni elemento (non scalabile)
    const BadApproach = () => (
        <ul>
            {todos.map(todo => (
                <li key={todo.id}>
                    {todo.text}
                    <button onClick={() => deleteTodo(todo.id)}>Delete</
                        button>
                </li>
            ))}
        </ul>
    );

    //      Event delegation (un solo handler)
    const handleListClick = (e) => {
        // Verifica se è un button delete
        if (e.target.classList.contains('delete-btn')) {
            const id = Number(e.target.dataset.id);
            deleteTodo(id);
        }
    };

    const deleteTodo = (id) => {
        setTodos(todos.filter(todo => todo.id !== id));
    };

    return (
        <ul onClick={handleListClick}>
            {todos.map(todo => (
                <li key={todo.id}>
                    {todo.text}
                    <button
                        className="delete-btn"
                        data-id={todo.id}
                    >
                        Delete
                    </button>
                </li>
            ))}
        </ul>
    );
}
```

6.7 Custom Events

6.7.1 Eventi Personalizzati

Listing 6.18: Custom events pattern

```
function CustomEventExample() {
    const handleCustomEvent = (data) => {
        console.log('Custom event:', data);
    };

    return (
        <div>
            <CustomButton onCustomClick={handleCustomEvent} />
        </div>
    );
}

function CustomButton({ onCustomClick }) {
    const handleClick = (e) => {
        // Raccogli dati aggiuntivi
        const data = {
            timestamp: Date.now(),
            position: {
                x: e.clientX,
                y: e.clientY
            },
            button: e.button
        };

        // Chiama callback con dati custom
        onCustomClick?.(data);
    };

    return <button onClick={handleClick}>Click me</button>;
}
```

6.8 Performance e Best Practices

6.8.1 1. Evita Inline Functions nei Render

Listing 6.19: Ottimizzazione event handlers

```
function TodoItem({ todo, onToggle, onDelete }) {
    //     MALE: Crea nuova funzione ad ogni render
    return (
        <li>
            <input
                type="checkbox"
                onChange={() => onToggle(todo.id)}
            />
            <button onClick={() => onDelete(todo.id)}>Delete</button>
        </li>
    );
}

//     MEGLIO: useCallback
```

```
function TodoList() {
    const [todos, setTodos] = useState([...]);

    const handleToggle = useCallback((id) => {
        setTodos(prev => prev.map(todo =>
            todo.id === id ? { ...todo, completed: !todo.completed } :
                todo
        ));
    }, []);

    const handleDelete = useCallback((id) => {
        setTodos(prev => prev.filter(todo => todo.id !== id));
    }, []);

    return (
        <ul>
            {todos.map(todo => (
                <TodoItem
                    key={todo.id}
                    todo={todo}
                    onToggle={handleToggle}
                    onDelete={handleDelete}
                />
            ))}
        </ul>
    );
}
```

6.8.2 2. Debouncing e Throttling

Listing 6.20: Debounce e throttle

```
// Debounce: Esegui dopo che l'utente smette di digitare
function SearchWithDebounce() {
    const [query, setQuery] = useState('');
    const [results, setResults] = useState([]);

    useEffect(() => {
        const timer = setTimeout(() => {
            if (query) {
                fetch(`/api/search?q=${query}`)
                    .then(res => res.json())
                    .then(setResults);
            }
        }, 500); // Aspetta 500ms dopo l'ultimo input

        return () => clearTimeout(timer);
    }, [query]);

    return (
        <div>
            <input
                value={query}
                onChange={(e) => setQuery(e.target.value)}
                placeholder="Cerca..."
            />
            <ul>
```

```
                {results.map(result => (
                    <li key={result.id}>{result.name}</li>
                ))}
            </ul>
        </div>
    );
}

// Throttle: Esegui al massimo una volta ogni X ms
function ScrollProgress() {
    const [progress, setProgress] = useState(0);
    const throttleRef = useRef(null);

    useEffect(() => {
        const handleScroll = () => {
            if (throttleRef.current) return;

            throttleRef.current = setTimeout(() => {
                const winScroll = document.documentElement.scrollTop;
                const height = document.documentElement.scrollHeight -
                               document.documentElement.clientHeight;
                const scrolled = (winScroll / height) * 100;

                setProgress(scrolled);
                throttleRef.current = null;
            }, 100); // Max una volta ogni 100ms
        };

        window.addEventListener('scroll', handleScroll);
        return () => window.removeEventListener('scroll', handleScroll);
    }, []);

    return (
        <div style={{
            position: 'fixed',
            top: 0,
            left: 0,
            width: `${progress}%`,
            height: '4px',
            backgroundColor: 'blue'
        }} />
    );
}
```

6.8.3 3. Passive Event Listeners

Listing 6.21: Passive listeners per performance

```
function PassiveListenerExample() {
    useEffect(() => {
        const handleTouchMove = (e) => {
            // Non chiama preventDefault()
            console.log('Touch move');
        };

        // Passive listener per scroll performance
        document.addEventListener(
            'touchmove',
            handleTouchMove,
            { passive: true }
        );

        return () => {
            document.removeEventListener('touchmove', handleTouchMove);
        };
    }, []);

    return <div>Swipe qui</div>;
}
```

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

```
//     Uncontrolled: Il DOM gestisce lo stato
function UncontrolledForm() {
    const nameRef = useRef();

    const handleSubmit = (e) => {
        e.preventDefault();
        console.log('Name:', nameRef.current.value);
    };

    return (
        <form onSubmit={handleSubmit}>
            <input ref={nameRef} defaultValue="Mario" />
            <button type="submit">Submit</button>
        </form>
    );
}

//     Controlled: React gestisce lo stato
function ControlledForm() {
    const [name, setName] = useState('Mario');

    const handleSubmit = (e) => {
        e.preventDefault();
        console.log('Name:', name);
    };

    return (
        <form onSubmit={handleSubmit}>
            <input
                value={name}
                onChange={(e) => setName(e.target.value)}
            />
            <button type="submit">Submit</button>
        </form>
    );
}
```

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

```
function RegistrationForm() {
    const [formData, setFormData] = useState({
        username: '',
        email: '',
        password: '',
        confirmPassword: '',
        age: '',
        country: '',
        gender: '',
        terms: false,
        newsletter: false
    });

    // Handler generico per tutti gli input
    const handleChange = (e) => {
        const { name, value, type, checked } = e.target;

        setFormData(prev => ({
            ...prev,
            [name]: type === 'checkbox' ? checked : value
        }));
    };

    const handleSubmit = (e) => {
        e.preventDefault();
        console.log('Form data:', formData);
    };

    return (
        <form onSubmit={handleSubmit}>
            {/* Text Input */}
            <div>
                <label htmlFor="username">Username:</label>
                <input
                    id="username"
                    type="text"
                    name="username"
                    value={formData.username}
                    onChange={handleChange}
                    required
                />
            </div>

            {/* Email Input */}
            <div>
                <label htmlFor="email">Email:</label>
                <input
                    id="email"
                    type="email"
                    name="email"
                    value={formData.email}
                    onChange={handleChange}
                    required
                />
            </div>

            {/* Password Input */}
            <div>
                <label htmlFor="password">Password:</label>
                <input
                    id="password"
                    type="password"
                    name="password"
                    value={formData.password}
                    onChange={handleChange}
                    required
                />
            </div>

            {/* Confirm Password */}
            <div>
                <label htmlFor="confirmPassword">Conferma Password:</label>
                <input
                    id="confirmPassword"
                    type="password"
                    name="confirmPassword"
                    value={formData.confirmPassword}
                    onChange={handleChange}
                    required
                />
            </div>

            {/* Number Input */}
            <div>
                <label htmlFor="age">Età:</label>
                <input
                    id="age"
                    type="number"
                    name="age"
                    value={formData.age}
                    onChange={handleChange}
                    min="18"
                    max="120"
                />
            </div>

            {/* Select */}
            <div>
                <label htmlFor="country">Paese:</label>
                <select
                    id="country"
                    name="country"
                    value={formData.country}
                    onChange={handleChange}
                >
                    <option value="">Seleziona...</option>
                    <option value="IT">Italia</option>
                    <option value="US">USA</option>
                    <option value="UK">UK</option>
                    <option value="FR">Francia</option>
                </select>
            </div>

            {/* Radio Buttons */}
            <div>
                <label>Genere:</label>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="male"
                        checked={formData.gender === 'male'}
                        onChange={handleChange}
                    />
                    Maschio
                </label>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="female"
                        checked={formData.gender === 'female'}
                        onChange={handleChange}
                    />
                    Femmina
                </label>
                <label>
                    <input
                        type="radio"
                        name="gender"
                        value="other"
                        checked={formData.gender === 'other'}
                        onChange={handleChange}
                    />
                    Altro
                </label>
            </div>

            {/* Checkboxes */}
            <div>
                <label>
                    <input
```

```
                        type="checkbox"
                        name="terms"
                        checked={formData.terms}
                        onChange={handleChange}
                    />
                    Accetto i termini e condizioni
                </label>
            </div>

            <div>
                <label>
                    <input
                        type="checkbox"
                        name="newsletter"
                        checked={formData.newsletter}
                        onChange={handleChange}
                    />
                    Iscriviti alla newsletter
                </label>
            </div>

            {/* Submit Button */}
            <button type="submit">Registrati</button>

            {/* Debug */}
            <pre>{JSON.stringify(formData, null, 2)}</pre>
        </form>
    );
}
```

7.3 Validazione Form

7.3.1 Validazione Base

Listing 7.3: Validazione manuale

```
function ValidatedForm() {
    const [formData, setFormData] = useState({
        username: '',
        email: '',
        password: '',
        confirmPassword: ''
    });

    const [errors, setErrors] = useState({});
    const [touched, setTouched] = useState({});

    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData(prev => ({ ...prev, [name]: value }));

        // Rimuovi errore quando l'utente inizia a digitare
        if (errors[name]) {
            setErrors(prev => ({ ...prev, [name]: '' }));
        }
    };

    const handleBlur = (e) => {
        const { name } = e.target;
        setTouched(prev => ({ ...prev, [name]: true }));
        validateField(name, formData[name]);
    };

    const validateField = (name, value) => {
        let error = '';

        switch (name) {
            case 'username':
                if (!value) {
                    error = 'Username è richiesto';
                } else if (value.length < 3) {
                    error = 'Username deve essere almeno 3 caratteri';
                } else if (!/^[a-zA-Z0-9_]+$/.test(value)) {
                    error = 'Username può contenere solo lettere, numeri e underscore';
                }
                break;

            case 'email':
                if (!value) {
                    error = 'Email è richiesta';
                } else if (!/\S+@\S+\.\S+/.test(value)) {
                    error = 'Email non valida';
                }
                break;

            case 'password':
                if (!value) {
                    error = 'Password è richiesta';
                } else if (value.length < 8) {
                    error = 'Password deve essere almeno 8 caratteri';
                } else if (!/(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/.test(value)) {
                    error = 'Password deve contenere maiuscola, minuscola e numero';
                }
                break;

            case 'confirmPassword':
                if (!value) {
                    error = 'Conferma password è richiesta';
                } else if (value !== formData.password) {
                    error = 'Le password non corrispondono';
                }
                break;
        }

        setErrors(prev => ({ ...prev, [name]: error }));
        return error;
    };

    const validateForm = () => {
        const newErrors = {};

        Object.keys(formData).forEach(key => {
            const error = validateField(key, formData[key]);
            if (error) newErrors[key] = error;
        });

        setErrors(newErrors);
        return Object.keys(newErrors).length === 0;
    };

    const handleSubmit = (e) => {
        e.preventDefault();

        // Mark all fields as touched
        const allTouched = Object.keys(formData).reduce(
            (acc, key) => ({ ...acc, [key]: true }),
            {}
        );
        setTouched(allTouched);

        if (validateForm()) {
            console.log('Form valido:', formData);
            // Submit...
        } else {
            console.log('Form non valido');
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            <div>
                <label htmlFor="username">Username:</label>
                <input
                    id="username"
                    type="text"
                    name="username"
                    value={formData.username}
                    onChange={handleChange}
                    onBlur={handleBlur}
                    className={touched.username && errors.username ? '
                        error' : ''}
                />
                {touched.username && errors.username && (
                    <span className="error-message">{errors.username}</
                        span>
                )}
            </div>

            <div>
                <label htmlFor="email">Email:</label>
                <input
                    id="email"
                    type="email"
                    name="email"
                    value={formData.email}
                    onChange={handleChange}
                    onBlur={handleBlur}
                    className={touched.email && errors.email ? 'error' :
                        ''}
                />
                {touched.email && errors.email && (
                    <span className="error-message">{errors.email}</span>
                )}
            </div>

            <div>
                <label htmlFor="password">Password:</label>
                <input
                    id="password"
                    type="password"
                    name="password"
                    value={formData.password}
                    onChange={handleChange}
                    onBlur={handleBlur}
                    className={touched.password && errors.password ? 'error' : ''}
                />
                {touched.password && errors.password && (
                    <span className="error-message">{errors.password}</span>
                )}
            </div>

            <div>
                <label htmlFor="confirmPassword">Conferma Password:</label>
                <input
                    id="confirmPassword"
                    type="password"
                    name="confirmPassword"
                    value={formData.confirmPassword}
                    onChange={handleChange}
                    onBlur={handleBlur}
                    className={touched.confirmPassword && errors.confirmPassword ? 'error' : ''}
                />
                {touched.confirmPassword && errors.confirmPassword && (
                    <span className="error-message">{errors.confirmPassword}</span>
                )}
            </div>

            <button type="submit">Registrati</button>
        </form>
    );
}
```

7.3.2 Custom Hook per Validazione

Listing 7.4: useForm custom hook

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

    const handleChange = (e) => {
        const { name, value, type, checked } = e.target;
        const newValue = type === 'checkbox' ? checked : value;

        setValues(prev => ({ ...prev, [name]: newValue }));

        // Rimuovi errore
        if (errors[name]) {
            setErrors(prev => ({ ...prev, [name]: '' }));
        }
    };

    const handleBlur = (e) => {
        const { name } = e.target;
        setTouched(prev => ({ ...prev, [name]: true }));
    };

    const handleSubmit = async (onSubmit) => {
        // Mark all as touched
        const allTouched = Object.keys(values).reduce(
            (acc, key) => ({ ...acc, [key]: true }),
            {}
        );
        setTouched(allTouched);

        // Validate
        const validationErrors = validate(values);
        setErrors(validationErrors);

        if (Object.keys(validationErrors).length === 0) {
            setIsSubmitting(true);
            try {
                await onSubmit(values);
            } finally {
                setIsSubmitting(false);
            }
        }
    };

    const resetForm = () => {
        setValues(initialValues);
        setErrors({});
        setTouched({});
        setIsSubmitting(false);
    };

    return {
        values,
        errors,
        touched,
        isSubmitting,
        handleChange,
        handleBlur,
        handleSubmit,
        resetForm
    };
}
```

```

// Uso
function LoginForm() {
    const validate = (values) => {
        const errors = {};

        if (!values.email) {
            errors.email = 'Email richiesta';
        } else if (!/\S+@\S+\.\S+/.test(values.email)) {
            errors.email = 'Email non valida';
        }

        if (!values.password) {
            errors.password = 'Password richiesta';
        } else if (values.password.length < 6) {
            errors.password = 'Password troppo corta';
        }

        return errors;
    };

    const form = useForm(
        { email: '', password: '' },
        validate
    );

    const onSubmit = async (values) => {
        console.log('Submitting:', values);
        await new Promise(resolve => setTimeout(resolve, 1000));
        console.log('Submitted!');
        form.resetForm();
    };

    return (
        <form onSubmit={(e) => {
            e.preventDefault();
            form.handleSubmit(onSubmit);
        }}>
            <div>
                <input
                    name="email"
                    type="email"
                    value={form.values.email}
                    onChange={form.handleChange}
                    onBlur={form.handleBlur}
                    placeholder="Email"
                />
                {form.touched.email && form.errors.email && (
                    <span>{form.errors.email}</span>
                )}
            </div>

            <div>
                <input
                    name="password"
                    type="password"
                    value={form.values.password}
                    onChange={form.handleChange}
                    onBlur={form.handleBlur}
                    placeholder="Password"
                />
                {form.touched.password && form.errors.password && (
                    <span>{form.errors.password}</span>
                )}
            </div>

            <button type="submit" disabled={form.isSubmitting}>
                {form.isSubmitting ? 'Accedendo...' : 'Accedi'}
            </button>
        </form>
    );
}
```

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

```
npm install react-hook-form
```

7.4.2 Form Base con React Hook Form

Listing 7.6: React Hook Form esempio base

```
import { useForm } from 'react-hook-form';

function BasicForm() {
    const {
        register,
        handleSubmit,
        formState: { errors, isSubmitting },
        reset
    } = useForm({
        defaultValues: {
            username: '',
            email: '',
            password: ''
        }
    });

    const onSubmit = async (data) => {
        console.log('Form data:', data);
        await new Promise(resolve => setTimeout(resolve, 1000));
        console.log('Submitted!');
        reset();
    };

    return (
        <form onSubmit={handleSubmit(onSubmit)}>
            {/* Username */}
            <div>
                <label htmlFor="username">Username:</label>
                <input
                    id="username"
                    {...register('username', {
                        required: 'Username è richiesto',
                        minLength: {
                            value: 3,
                            message: 'Minimo 3 caratteri'
                        },
                        pattern: {
                            value: /^[a-zA-Z0-9_]+$/,
                            message: 'Solo lettere, numeri e underscore'
                        }
                    })}
                />
                {errors.username && (
                    <span className="error">{errors.username.message}</
                        span>
                )}
            </div>

            {/* Email */}
            <div>
                <label htmlFor="email">Email:</label>
                <input
                    id="email"
                    type="email"
                    {...register('email', {
                        required: 'Email è richiesta',
                        pattern: {
                            value: /\S+@\S+\.\S+/,
                            message: 'Email non valida'
                        }
                    })}
                />
                {errors.email && (
                    <span className="error">{errors.email.message}</span
                        >
                )}
            </div>

            {/* Password */}
            <div>
                <label htmlFor="password">Password:</label>
                <input
                    id="password"
                    type="password"
                    {...register('password', {
                        required: 'Password è richiesta',
                        minLength: {
                            value: 8,
                            message: 'Minimo 8 caratteri'
                        },
                        validate: {
                            hasUpperCase: (value) =>
                                /[A-Z]/.test(value) || 'Deve contenere
                                    maiuscola',
                            hasLowerCase: (value) =>
                                /[a-z]/.test(value) || 'Deve contenere minuscola',
                            hasNumber: (value) =>
                                /\d/.test(value) || 'Deve contenere un numero'
                        }
                    })}
                />
                {errors.password && (
                    <span className="error">{errors.password.message}</span>
                )}
            </div>

            <button type="submit" disabled={isSubmitting}>
                {isSubmitting ? 'Invio...' : 'Invia'}
            </button>
        </form>
    );
}
```

7.4.3 Validazione Avanzata React Hook Form

Listing 7.7: Validazione custom

```
import { useForm } from 'react-hook-form';

function AdvancedForm() {
    const { register, handleSubmit, watch, formState: { errors } } = useForm();

    // Watch per validazione dipendente
    const password = watch('password');

    const onSubmit = (data) => {
        console.log(data);
    };

    return (
        <form onSubmit={handleSubmit(onSubmit)}>
            {/* Password */}
            <input
                type="password"
                {...register('password', {
                    required: 'Password richiesta',
                    minLength: { value: 8, message: 'Minimo 8 caratteri' }
                })}
            />
            {errors.password && <span>{errors.password.message}</span>}

            {/* Confirm Password - dipende da password */}
            <input
                type="password"
                {...register('confirmPassword', {
                    required: 'Conferma password richiesta',
                    validate: (value) =>
                        value === password || 'Le password non corrispondono'
                })}
            />
            {errors.confirmPassword && (
                <span>{errors.confirmPassword.message}</span>
            )}

            {/* Age - validazione numerica */}
            <input
                type="number"
                {...register('age', {
                    required: 'Età richiesta',
                    min: { value: 18, message: 'Devi avere almeno 18 anni' },
                    max: { value: 120, message: 'Età non valida' },
                    valueAsNumber: true
                })}
            />
            {errors.age && <span>{errors.age.message}</span>}

            {/* Terms - checkbox */}
            <label>
                <input
                    type="checkbox"
                    {...register('terms', {
                        required: 'Devi accettare i termini'
                    })}
                />
                Accetto i termini
            </label>
            {errors.terms && <span>{errors.terms.message}</span>}

            <button type="submit">Invia</button>
        </form>
    );
}
```

7.4.4 Form Arrays con React Hook Form

Listing 7.8: Dynamic form fields

```
import { useForm, useFieldArray } from 'react-hook-form';

function DynamicForm() {
    const { register, control, handleSubmit } = useForm({
        defaultValues: {
            users: [{ name: '', email: '' }]
        }
    });

    const { fields, append, remove } = useFieldArray({
        control,
        name: 'users'
    });

    const onSubmit = (data) => {
        console.log('Users:', data.users);
    };

    return (
        <form onSubmit={handleSubmit(onSubmit)}>
            {fields.map((field, index) => (
                <div key={field.id}>
                    <input
                        {...register(`users.${index}.name`, {
                            required: 'Nome richiesto'
                        })}
                        placeholder="Nome"
                    />

                    <input
                        {...register(`users.${index}.email`, {
                            required: 'Email richiesta',
                            pattern: {
                                value: /\S+@\S+\.\S+/,
                                message: 'Email non valida'
                            }
                        })}
                        placeholder="Email"
                    />

                    <button type="button" onClick={() => remove(index)}>
                        Rimuovi
                    </button>
                </div>
            ))}

            <button
                type="button"
                onClick={() => append({ name: '', email: '' })}
            >
                Aggiungi Utente
            </button>

            <button type="submit">Invia</button>
        </form>
    );
}
```

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

```
npm install formik yup
```

7.5.2 Form Base con Formik

Listing 7.10: Formik esempio base

```
import { Formik, Form, Field, ErrorMessage } from 'formik';
import * as Yup from 'yup';

// Schema validazione con Yup
const validationSchema = Yup.object({
    username: Yup.string()
        .min(3, 'Minimo 3 caratteri')
        .required('Username richiesto'),
    email: Yup.string()
        .email('Email non valida')
        .required('Email richiesta'),
    password: Yup.string()
        .min(8, 'Minimo 8 caratteri')
        .matches(/[A-Z]/, 'Deve contenere maiuscola')
        .matches(/[a-z]/, 'Deve contenere minuscola')
        .matches(/\d/, 'Deve contenere numero')
        .required('Password richiesta'),
    confirmPassword: Yup.string()
        .oneOf([Yup.ref('password')], 'Le password non corrispondono')
        .required('Conferma password richiesta'),
    age: Yup.number()
        .min(18, 'Devi avere almeno 18 anni')
        .max(120, 'Età non valida')
        .required('Età richiesta'),
    terms: Yup.boolean()
        .oneOf([true], 'Devi accettare i termini')
});

function FormikExample() {
    const initialValues = {
        username: '',
        email: '',
        password: '',
        confirmPassword: '',
        age: '',
        terms: false
    };

    const handleSubmit = async (values, { setSubmitting, resetForm }) =>
        {
        console.log('Form values:', values);
        await new Promise(resolve => setTimeout(resolve, 1000));
        console.log('Submitted!');
        resetForm();
        setSubmitting(false);
    };

    return (
        <Formik
            initialValues={initialValues}
            validationSchema={validationSchema}
            onSubmit={handleSubmit}
        >
            {({ isSubmitting, errors, touched }) => (
                <Form>
                    {/* Username */}
                    <div>
```

```
                        <label htmlFor="username">Username:</label>
                        <Field
                            id="username"
                            name="username"
                            type="text"
                        />
                        <ErrorMessage name="username" component="span"
                           className="error" />
                    </div>

                    {/* Email */}
                    <div>
                        <label htmlFor="email">Email:</label>
                        <Field
                            id="email"
                            name="email"
                            type="email"
                        />
                        <ErrorMessage name="email" component="span"
                           className="error" />
                    </div>

                    {/* Password */}
                    <div>
                        <label htmlFor="password">Password:</label>
                        <Field
                            id="password"
                            name="password"
                            type="password"
                        />
                        <ErrorMessage name="password" component="span"
                           className="error" />
                    </div>

                    {/* Confirm Password */}
                    <div>
                        <label htmlFor="confirmPassword">Conferma
                           Password:</label>
                        <Field
                            id="confirmPassword"
                            name="confirmPassword"
                            type="password"
                        />
                        <ErrorMessage name="confirmPassword" component="
                           span" className="error" />
                    </div>

                    {/* Age */}
                    <div>
                        <label htmlFor="age">Età:</label>
                        <Field
                            id="age"
                            name="age"
                            type="number"
                        />
                        <ErrorMessage name="age" component="span"
                           className="error" />
                    </div>

                    {/* Terms Checkbox */}
                    <div>
                        <label>
                            <Field
                                type="checkbox"
                                name="terms"
                            />
                            Accetto i termini e condizioni
                        </label>
                        <ErrorMessage name="terms" component="span"
                           className="error" />
                    </div>

                    <button type="submit" disabled={isSubmitting}>
                        {isSubmitting ? 'Invio...' : 'Invia'}
                    </button>
                </Form>
            )}
        </Formik>
    );
}
```

7.5.3 Formik con Custom Components

Listing 7.11: Custom input components con Formik

```
import { Formik, Form, Field, ErrorMessage } from 'formik';

// Custom Input Component
function TextInput({ label, ...props }) {
    return (
        <div className="form-field">
            <label htmlFor={props.id || props.name}>{label}</label>
            <Field {...props} />
            <ErrorMessage name={props.name} component="div" className="
               error" />
        </div>
    );
}

// Custom Select Component
function Select({ label, options, ...props }) {
    return (
        <div className="form-field">
            <label htmlFor={props.id || props.name}>{label}</label>
            <Field as="select" {...props}>
                <option value="">Seleziona...</option>
                {options.map(option => (
                    <option key={option.value} value={option.value}>
                        {option.label}
                    </option>
                ))}
            </Field>
            <ErrorMessage name={props.name} component="div" className="
               error" />
        </div>
    );
}

// Custom Checkbox Component
function Checkbox({ children, ...props }) {
    return (
        <div className="form-field">
            <label className="checkbox-label">
                <Field type="checkbox" {...props} />
                {children}
            </label>
            <ErrorMessage name={props.name} component="div" className="
              error" />
        </div>
    );
}

// Uso
function CustomComponentsForm() {
    return (
        <Formik
            initialValues={{
                name: '',
                email: '',
                country: '',
                terms: false
            }}
            onSubmit={(values) => console.log(values)}
        >
            <Form>
                <TextInput
                    label="Nome"
                    name="name"
                    type="text"
                />

                <TextInput
                    label="Email"
                    name="email"
                    type="email"
                />

                <Select
                    label="Paese"
                    name="country"
                    options={[
                        { value: 'IT', label: 'Italia' },
                        { value: 'US', label: 'USA' },
                        { value: 'UK', label: 'UK' }
                    ]}
                />

                <Checkbox name="terms">
                    Accetto i termini e condizioni
                </Checkbox>

                <button type="submit">Invia</button>
            </Form>
        </Formik>
    );
}
```

7.6 Form Patterns Avanzati

7.6.1 Multi-Step Form

Listing 7.12: Form multi-step

```
function MultiStepForm() {
    const [step, setStep] = useState(1);
    const [formData, setFormData] = useState({
        // Step 1
        personalInfo: {
            firstName: '',
            lastName: '',
            email: ''
        },
        // Step 2
        address: {
            street: '',
            city: '',
            country: ''
        },
        // Step 3
        account: {
            username: '',
            password: ''
        }
    });

    const nextStep = () => setStep(prev => prev + 1);
    const prevStep = () => setStep(prev => prev - 1);

    const updateFormData = (section, data) => {
        setFormData(prev => ({
            ...prev,
            [section]: { ...prev[section], ...data }
        }));
    };

    const handleSubmit = () => {
        console.log('Final form data:', formData);
    };

    return (
        <div className="multi-step-form">
            {/* Progress Indicator */}
            <div className="progress">
                <div className={`step ${step >= 1 ? 'active' : ''}`}>1</
                   div>
                <div className={`step ${step >= 2 ? 'active' : ''}`}>2</
                   div>
                <div className={`step ${step >= 3 ? 'active' : ''}`}>3</
                   div>
            </div>

            {/* Step 1: Personal Info */}
            {step === 1 && (
                <Step1
                    data={formData.personalInfo}
                    onNext={(data) => {
                        updateFormData('personalInfo', data);
                        nextStep();
                    }}
                />
            )}

            {/* Step 2: Address */}
            {step === 2 && (
                <Step2
                    data={formData.address}
                    onNext={(data) => {
                        updateFormData('address', data);
                        nextStep();
                    }}
                    onPrev={prevStep}
                />
            )}

            {/* Step 3: Account */}
            {step === 3 && (
                <Step3
                    data={formData.account}
                    onSubmit={(data) => {
                        updateFormData('account', data);
                        handleSubmit();
                    }}
                    onPrev={prevStep}
                />
            )}
        </div>
    );
}

// Step 1 Component
function Step1({ data, onNext }) {
    const [formData, setFormData] = useState(data);

    const handleSubmit = (e) => {
        e.preventDefault();
        onNext(formData);
    };

    return (
        <form onSubmit={handleSubmit}>
            <h2>Informazioni Personali</h2>
            <input
                value={formData.firstName}
                onChange={(e) => setFormData({ ...formData, firstName: e
                    .target.value })}
                placeholder="Nome"
                required
            />
```

```
                <input
                    value={formData.lastName}
                    onChange={(e) => setFormData({ ...formData, lastName: e.
                        target.value })}
                    placeholder="Cognome"
                    required
                />
                <input
                    type="email"
                    value={formData.email}
                    onChange={(e) => setFormData({ ...formData, email: e.
                        target.value })}
                    placeholder="Email"
                    required
                />
                <button type="submit">Avanti</button>
            </form>
        );
}

// Step 2 e Step 3 simili...
```

7.6.2 Form con File Upload

Listing 7.13: File upload form

```
function FileUploadForm() {
    const [file, setFile] = useState(null);
    const [preview, setPreview] = useState(null);
    const [uploading, setUploading] = useState(false);
    const [progress, setProgress] = useState(0);

    const handleFileChange = (e) => {
        const selectedFile = e.target.files[0];

        if (selectedFile) {
            setFile(selectedFile);

            // Preview per immagini
            if (selectedFile.type.startsWith('image/')) {
                const reader = new FileReader();
                reader.onloadend = () => {
                    setPreview(reader.result);
                };
                reader.readAsDataURL(selectedFile);
            }
        }
    };

    const handleSubmit = async (e) => {
        e.preventDefault();

        if (!file) {
            alert('Seleziona un file');
            return;
        }

        setUploading(true);

        const formData = new FormData();
        formData.append('file', file);
        formData.append('description', 'File upload');

        try {
            const xhr = new XMLHttpRequest();

            // Progress tracking
            xhr.upload.addEventListener('progress', (e) => {
                if (e.lengthComputable) {
                    const percentComplete = (e.loaded / e.total) * 100;
                    setProgress(percentComplete);
                }
            });

            xhr.addEventListener('load', () => {
                if (xhr.status === 200) {
                    console.log('Upload completato!');
                    setFile(null);
                    setPreview(null);
                    setProgress(0);
                }
            });

            xhr.open('POST', '/api/upload');
            xhr.send(formData);
        } catch (error) {
            console.error('Errore upload:', error);
        } finally {
            setUploading(false);
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            <div>
                <input
                    type="file"
                    onChange={handleFileChange}
                    accept="image/*"
                />
            </div>

            {preview && (
                <div>
                    <h3>Anteprima:</h3>
                    <img src={preview} alt="Preview" style={{ maxWidth:
                        '300px' }} />
                </div>
            )}

            {file && (
                <div>
                    <p>File: {file.name}</p>
                    <p>Dimensione: {(file.size / 1024).toFixed(2)} KB</p
                        >
                </div>
            )}

            {uploading && (
                <div>
                    <progress value={progress} max="100" />
                    <p>{progress.toFixed(0)}%</p>
                </div>
            )}

            <button type="submit" disabled={!file || uploading}>
                {uploading ? 'Caricamento...' : 'Carica'}
            </button>
        </form>
    );
}
```

7.6.3 Form con Async Validation

Listing 7.14: Validazione asincrona

```
function AsyncValidationForm() {
    const [username, setUsername] = useState('');
    const [checking, setChecking] = useState(false);
    const [isAvailable, setIsAvailable] = useState(null);
    const debounceRef = useRef(null);

    // Check username availability
    const checkUsername = async (value) => {
        if (value.length < 3) {
            setIsAvailable(null);
            return;
        }

        setChecking(true);

        try {
            // Simula API call
            await new Promise(resolve => setTimeout(resolve, 1000));
            const response = await fetch(`/api/check-username?username=$
               {value}`);
            const data = await response.json();

            setIsAvailable(data.available);
        } catch (error) {
            console.error('Error checking username:', error);
        } finally {
            setChecking(false);
        }
    };

    const handleUsernameChange = (e) => {
        const value = e.target.value;
        setUsername(value);
        setIsAvailable(null);

        // Debounce check
        clearTimeout(debounceRef.current);
        debounceRef.current = setTimeout(() => {
            checkUsername(value);
        }, 500);
    };

    const handleSubmit = (e) => {
        e.preventDefault();

        if (isAvailable) {
            console.log('Submitting username:', username);
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            <div>
                <label>Username:</label>
                <input
                    value={username}
                    onChange={handleUsernameChange}
                    placeholder="Scegli username"
                />

                {checking && <span>Controllo disponibilità...</span>}

                {!checking && isAvailable === true && (
                    <span style={{ color: 'green' }}>    Disponibile</
                        span>
                )}

                {!checking && isAvailable === false && (
                    <span style={{ color: 'red' }}>    Non disponibile</
                        span>
                )}
            </div>

            <button type="submit" disabled={!isAvailable}>
                Registra
            </button>
        </form>
    );
}
```

7.7 Best Practices

7.7.1 1. Accessibilità

Listing 7.15: Form accessibili

```
function AccessibleForm() {
    return (
        <form>
            {/*     Label associata a input */}
            <label htmlFor="email">Email:</label>
            <input id="email" type="email" name="email" />

            {/*     Aria attributes */}
            <input
                id="username"
                aria-label="Username"
                aria-required="true"
                aria-invalid={errors.username ? 'true' : 'false'}
                aria-describedby="username-error"
            />
            <span id="username-error" role="alert">
                {errors.username}
            </span>

            {/*     Fieldset per raggruppamento */}
            <fieldset>
                <legend>Informazioni di contatto</legend>
                <input type="tel" name="phone" />
                <input type="email" name="email" />
            </fieldset>
        </form>
    );
}
```

7.7.2 2. Performance

Listing 7.16: Ottimizzazione performance

```
//     Debounce validation
const useDebounce = (value, delay) => {
    const [debouncedValue, setDebouncedValue] = useState(value);

    useEffect(() => {
        const timer = setTimeout(() => setDebouncedValue(value), delay);
        return () => clearTimeout(timer);
    }, [value, delay]);

    return debouncedValue;
};

//     Memoizza callbacks
const handleSubmit = useCallback(async (data) => {
    await submitForm(data);
}, []);

//     Controlled components solo quando necessario
// Se non hai bisogno di validazione real-time, usa uncontrolled
```

7.7.3 3. Security

Listing 7.17: Sicurezza form

```
function SecureForm() {
    const handleSubmit = async (data) => {
        //     Sanitize input prima di inviare
        const sanitized = {
            username: DOMPurify.sanitize(data.username),
            email: data.email.toLowerCase().trim()
        };

        //     HTTPS sempre
        await fetch('https://api.example.com/submit', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                //     CSRF token
                'X-CSRF-Token': getCsrfToken()
            },
            body: JSON.stringify(sanitized)
        });
    };

    return <form onSubmit={handleSubmit}>...</form>;
}
```

7.8 Conclusione

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

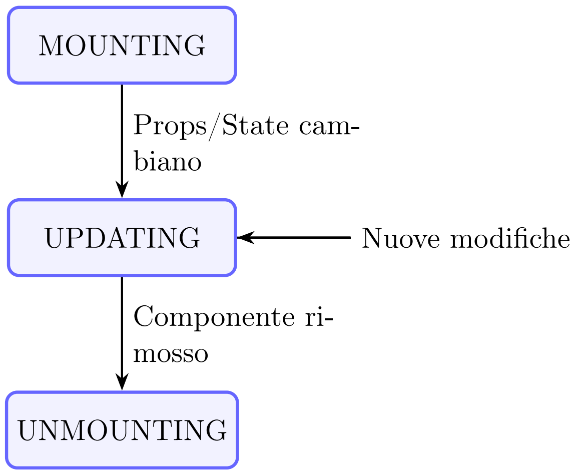
Ogni componente React passa attraverso tre fasi principali:

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

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

function MyComponent() {
  useEffect(() => {
    // Codice da eseguire (effetto)

    return () => {
      // Codice di cleanup (opzionale)
    };
  }, [dependencies]); // Array di dipendenze
}
```

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. Nessun array: esegue ad ogni render
useEffect(() => {
  console.log('Esegue ad ogni render');
});

// 2. Array vuoto: esegue solo al mount
useEffect(() => {
  console.log('Esegue solo una volta al mount');
}, []);

// 3. Con dipendenze: esegue quando cambiano le dipendenze
useEffect(() => {
  console.log('Esegue quando userId cambia');
}, [userId]);
```

8.3.2 Esempio Pratico: Document Title

Listing 8.3: Aggiornamento dinamico del titolo della pagina

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

function DocumentTitleUpdater() {
  const [count, setCount] = useState(0);
  const [name, setName] = useState('Utente');

  // Effetto che dipende da count
  useEffect(() => {
    document.title = `Hai cliccato ${count} volte`;
  }, [count]); // Si ri-esegue solo quando count cambia

  // Effetto che dipende da name
  useEffect(() => {
    console.log(`L'utente ora si chiama: ${name}`);
  }, [name]); // Si ri-esegue solo quando name cambia

  return (
    <div>
      <h2>Ciao, {name}!</h2>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Il tuo nome"
      />

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

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

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

function UserProfile({ userId }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    // Questa funzione viene eseguita solo al mount
    // (o quando userId cambia)

    setLoading(true);
    setError(null);

    fetch(`https://api.example.com/users/${userId}`)
      .then(response => {
        if (!response.ok) {
          throw new Error('Errore nel caricamento');
        }
        return response.json();
      })
      .then(data => {
        setUser(data);
        setLoading(false);
      })
      .catch(err => {
        setError(err.message);
        setLoading(false);
      });
  }, [userId]); // Ri-esegue quando userId cambia

  if (loading) return <p>Caricamento...</p>;
  if (error) return <p>Errore: {error}</p>;
  if (!user) return null;

  return (
    <div className="user-profile">
      <h2>{user.name}</h2>
      <p>Email: {user.email}</p>
      <p>Username: {user.username}</p>
    </div>
  );
}
```

8.4.3 Pattern Avanzato: Async/Await nel useEffect

Listing 8.5: Uso di async/await in useEffect

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

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Non possiamo rendere useEffect async direttamente
    // Quindi creiamo una funzione async interna

    const fetchUsers = async () => {
      try {
        setLoading(true);
        const response = await fetch('https://api.example.com/users');

        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }

        const data = await response.json();
        setUsers(data);
      } catch (error) {
        console.error('Errore nel fetch:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchUsers();
  }, []); // Array vuoto = esegue solo al mount

  if (loading) {
    return <div>Caricamento utenti...</div>;
  }

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

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

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

function Timer() {
  const [seconds, setSeconds] = useState(0);
  const [isRunning, setIsRunning] = useState(false);

  useEffect(() => {
    let intervalId = null;

    if (isRunning) {
      // Setup: avvia il timer
      intervalId = setInterval(() => {
        setSeconds(prev => prev + 1);
      }, 1000);

      console.log('Timer avviato');
    }

    // Cleanup: pulisce il timer
    return () => {
      if (intervalId) {
        clearInterval(intervalId);
        console.log('Timer pulito');
      }
    };
  }, [isRunning]); // Ri-esegue quando isRunning cambia

  const toggle = () => setIsRunning(!isRunning);
  const reset = () => {
    setIsRunning(false);
    setSeconds(0);
  };

  return (
    <div className="timer">
      <h2>Timer: {seconds}s</h2>
      <button onClick={toggle}>
        {isRunning ? 'Pausa' : 'Avvia'}
      </button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}
```

8.5.3 Esempio: Event Listener con Cleanup

Listing 8.7: Gestione di event listener del browser

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

function WindowSize() {
  const [windowSize, setWindowSize] = useState({
    width: window.innerWidth,
    height: window.innerHeight
  });

  useEffect(() => {
    // Handler dell'evento
    const handleResize = () => {
      setWindowSize({
        width: window.innerWidth,
        height: window.innerHeight
      });
    };

    // Setup: aggiungi event listener
    window.addEventListener('resize', handleResize);
    console.log('Event listener aggiunto');

    // Cleanup: rimuovi event listener
    return () => {
      window.removeEventListener('resize', handleResize);
      console.log('Event listener rimosso');
    };
  }, []); // Array vuoto = setup al mount, cleanup all'unmount

  return (
    <div>
      <h2>Dimensioni Finestra</h2>
      <p>Larghezza: {windowSize.width}px</p>
      <p>Altezza: {windowSize.height}px</p>
    </div>
  );
}
```

8.5.4 Esempio: WebSocket con Cleanup

Listing 8.8: Connessione WebSocket con pulizia corretta

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

function ChatRoom({ roomId }) {
  const [messages, setMessages] = useState([]);
  const [connectionStatus, setConnectionStatus] = useState('disconnected
     ');

  useEffect(() => {
    // Setup: crea connessione WebSocket
    const ws = new WebSocket(`wss://chat.example.com/room/${roomId}`);

    ws.onopen = () => {
      console.log('Connesso alla chat room:', roomId);
      setConnectionStatus('connected');
    };

    ws.onmessage = (event) => {
      const message = JSON.parse(event.data);
      setMessages(prev => [...prev, message]);
    };

    ws.onerror = (error) => {
      console.error('WebSocket error:', error);
      setConnectionStatus('error');
    };

    ws.onclose = () => {
      console.log('Disconnesso dalla chat room');
      setConnectionStatus('disconnected');
    };

    // Cleanup: chiudi connessione quando il componente si smonta
    // o quando roomId cambia
    return () => {
      console.log('Chiusura connessione WebSocket...');
      ws.close();
    };
  }, [roomId]); // Ri-connetti quando cambia la room

  return (
    <div className="chat-room">
      <div className="status">
        Status: <span className={connectionStatus}>{connectionStatus}</
            span>
      </div>
      <div className="messages">
        {messages.map((msg, index) => (
          <div key={index} className="message">
            <strong>{msg.user}:</strong> {msg.text}
          </div>
        ))}
      </div>
    </div>
  );
}
```

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

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

function SearchResults({ searchTerm }) {
  const [results, setResults] = useState([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    // Questo effetto si ri-esegue ogni volta che searchTerm cambia
    if (!searchTerm) {
      setResults([]);
      return;
    }

    setLoading(true);

    // Simula chiamata API
    const timeoutId = setTimeout(() => {
      const mockResults = [
        `Risultato 1 per "${searchTerm}"`,
        `Risultato 2 per "${searchTerm}"`,
        `Risultato 3 per "${searchTerm}"`
      ];
      setResults(mockResults);
      setLoading(false);
    }, 500);

    // Cleanup: annulla la ricerca precedente se searchTerm cambia
    return () => {
      clearTimeout(timeoutId);
    };
  }, [searchTerm]);

  if (loading) return <div>Ricerca in corso...</div>;

  return (
    <ul>
      {results.map((result, index) => (
        <li key={index}>{result}</li>
      ))}
    </ul>
  );
}

// Componente padre con input
function SearchApp() {
  const [searchTerm, setSearchTerm] = useState('');

  return (
    <div>
      <input
        type="text"
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        placeholder="Cerca..."
      />
      <SearchResults searchTerm={searchTerm} />
    </div>
  );
}
```

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

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

function DebouncedSearch() {
  const [searchTerm, setSearchTerm] = useState('');
  const [debouncedTerm, setDebouncedTerm] = useState('');

  // Effetto per il debouncing
  useEffect(() => {
    // Imposta un timer per aggiornare debouncedTerm dopo 500ms
    const timerId = setTimeout(() => {
      setDebouncedTerm(searchTerm);
    }, 500);

    // Cleanup: cancella il timer se searchTerm cambia
    // prima che i 500ms siano trascorsi
    return () => {
      clearTimeout(timerId);
    };
  }, [searchTerm]);

  // Effetto per la ricerca (si attiva solo quando debouncedTerm cambia)
  useEffect(() => {
    if (debouncedTerm) {
      console.log('Eseguo ricerca per:', debouncedTerm);
      // Qui faremmo la chiamata API
    }
  }, [debouncedTerm]);

  return (
    <div>
      <input
        type="text"
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        placeholder="Cerca (con debounce)..."
      />
      <p>Termine di ricerca: {searchTerm}</p>
      <p>Ricerca attiva per: {debouncedTerm}</p>
    </div>
  );
}
```

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

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

function UserData({ userId }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    // Crea un AbortController per questa richiesta
    const controller = new AbortController();
    const signal = controller.signal;

    const fetchUser = async () => {
      try {
        setLoading(true);
        setError(null);

        const response = await fetch(
          `https://api.example.com/users/${userId}`,
          { signal } // Passa il signal alla fetch
        );

        if (!response.ok) {
          throw new Error('Errore nel caricamento');
        }

        const data = await response.json();

        // Controlla se il componente è ancora montato
        if (!signal.aborted) {
          setUser(data);
        }
      } catch (err) {
        // Ignora errori di abort
        if (err.name !== 'AbortError') {
          setError(err.message);
        }
      } finally {
        if (!signal.aborted) {
          setLoading(false);
        }
      }
    };

    fetchUser();

    // Cleanup: annulla la richiesta se il componente si smonta
    // o se userId cambia
    return () => {
      controller.abort();
    };
  }, [userId]);

  if (loading) return <div>Caricamento...</div>;
  if (error) return <div>Errore: {error}</div>;
  if (!user) return null;

  return (
    <div>
      <h2>{user.name}</h2>
      <p>{user.email}</p>
    </div>
  );
}
```

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

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

// Hook che esegue una funzione solo al mount
function useMount(callback) {
  useEffect(() => {
    callback();
  }, []); // Array vuoto = esegue solo al mount
}

// Hook che esegue una funzione solo all'unmount
function useUnmount(callback) {
  useEffect(() => {
    return callback; // Ritorna la funzione di cleanup
  }, []); // Array vuoto = cleanup solo all'unmount
}

// Hook che esegue codice al mount e all'unmount
function useMountUnmount(onMount, onUnmount) {
  useEffect(() => {
    onMount();
    return onUnmount;
  }, []);
}

// Esempio di utilizzo
function ExampleComponent() {
  useMount(() => {
    console.log('Componente montato!');
  });

  useUnmount(() => {
    console.log('Componente smontato!');
  });

  return <div>Esempio</div>;
}
```

8.7.3 Hook useUpdateEffect

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

Listing 8.13: Custom hook useUpdateEffect

```
import { useEffect, useRef, useState } from 'react';

function useUpdateEffect(effect, dependencies) {
  const isFirstRender = useRef(true);

  useEffect(() => {
    if (isFirstRender.current) {
      isFirstRender.current = false;
      return;
    }

    return effect();
  }, dependencies);
}
```

```

// Esempio di utilizzo
function Counter() {
  const [count, setCount] = useState(0);

  // Questo NON si esegue al primo render
  useUpdateEffect(() => {
    console.log('Count è stato aggiornato a:', count);
  }, [count]);

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

8.7.4 Hook useInterval

Un hook personalizzato per gestire intervalli in modo sicuro.

Listing 8.14: Custom hook useInterval

```
import { useEffect, useRef, useState } from 'react';

function useInterval(callback, delay) {
  const savedCallback = useRef();

  // Salva la callback più recente
  useEffect(() => {
    savedCallback.current = callback;
  }, [callback]);

  // Setup dell'intervallo
  useEffect(() => {
    if (delay === null) return;

    const tick = () => {
      savedCallback.current();
    };

    const id = setInterval(tick, delay);

    // Cleanup
    return () => clearInterval(id);
  }, [delay]);
}

// Esempio di utilizzo
function Clock() {
  const [time, setTime] = useState(new Date());

  useInterval(() => {
    setTime(new Date());
  }, 1000); // Aggiorna ogni secondo

  return (
    <div>
      <h2>{time.toLocaleTimeString()}</h2>
    </div>
  );
}
```

8.8 Errori Comuni e Best Practices

8.8.1 Errore: Dipendenze Mancanti

Listing 8.15: Esempio di dipendenze mancanti

```
// SBAGLIATO: count non è nelle dipendenze
function BadExample() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      console.log(count); // Usa count ma non è nelle dipendenze!
    }, 1000);

    return () => clearInterval(interval);
  }, []); // Manca count

  return <button onClick={() => setCount(count + 1)}>Click</button>;
}

// CORRETTO: Usa la forma funzionale di setState
function GoodExample() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setCount(prev => {
        console.log(prev); // Accedi al valore tramite callback
        return prev;
      });
    }, 1000);

    return () => clearInterval(interval);
  }, []); // Corretto: non dipende da count

  return <button onClick={() => setCount(count + 1)}>Click</button>;
}
```

8.8.2 Errore: Memory Leak

Listing 8.16: Come evitare memory leak

```
// SBAGLIATO: Nessun cleanup
function BadComponent() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(res => res.json())
      .then(data => setData(data)); // Potrebbe eseguire su componente
         smontato
  }, []);

  return <div>{data?.value}</div>;
}

// CORRETTO: Con flag di cleanup
function GoodComponent() {
  const [data, setData] = useState(null);

  useEffect(() => {
    let isMounted = true;

    fetch('https://api.example.com/data')
      .then(res => res.json())
      .then(data => {
        if (isMounted) { // Controlla se il componente è ancora montato
          setData(data);
        }
      });

    return () => {
      isMounted = false; // Cleanup
    };
  }, []);

  return <div>{data?.value}</div>;
}
```

8.8.3 Best Practice: Separazione degli Effetti

Listing 8.17: Separare effetti con dipendenze diverse

```
// SBAGLIATO: Un unico effetto con troppe responsabilità
function BadComponent({ userId, theme }) {
  useEffect(() => {
    // Fetch user data
    fetchUser(userId);

    // Update theme
    document.body.className = theme;

    // Setup analytics
    analytics.track('page_view');
  }, [userId, theme]); // Entrambe le operazioni si ri-eseguono
}

// CORRETTO: Effetti separati
function GoodComponent({ userId, theme }) {
  // Effetto per il fetch dell'utente
  useEffect(() => {
    fetchUser(userId);
  }, [userId]);

  // Effetto per il tema
  useEffect(() => {
    document.body.className = theme;
  }, [theme]);

  // Effetto per analytics (solo al mount)
  useEffect(() => {
    analytics.track('page_view');
  }, []);
}
```

8.9 Casi d'Uso Pratici Completi

8.9.1 Sistema di Notifiche con Timeout

Listing 8.18: Sistema completo di notifiche

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

function Notification({ message, duration = 3000, onClose }) {
  useEffect(() => {
    // Imposta timer per chiudere automaticamente
    const timer = setTimeout(() => {
      onClose();
    }, duration);

    // Cleanup: cancella timer se il componente si smonta prima
    return () => {
      clearTimeout(timer);
    };
  }, [duration, onClose]);

  return (
    <div className="notification">
      <p>{message}</p>
      <button onClick={onClose}>X</button>
    </div>
  );
}

function NotificationSystem() {
  const [notifications, setNotifications] = useState([]);

  const addNotification = (message) => {
    const id = Date.now();
    setNotifications(prev => [...prev, { id, message }]);
  };

  const removeNotification = (id) => {
    setNotifications(prev => prev.filter(n => n.id !== id));
  };

  return (
    <div>
      <button onClick={() => addNotification('Nuova notifica!')}>
        Mostra Notifica
      </button>

      <div className="notifications-container">
```

```
        {notifications.map(notification => (
          <Notification
            key={notification.id}
            message={notification.message}
            onClose={() => removeNotification(notification.id)}
          />
        ))}
      </div>
    </div>
  );
}
```

8.9.2 Auto-Save con Debounce

Listing 8.19: Auto-save di un form con debounce

```
import { useState, useEffect, useRef } from 'react';

function AutoSaveForm() {
  const [formData, setFormData] = useState({
    title: '',
    content: ''
  });
  const [saveStatus, setSaveStatus] = useState('saved'); // saved,
     saving, error
  const [lastSaved, setLastSaved] = useState(null);

  // Debounce e auto-save
  useEffect(() => {
    // Non salvare se i campi sono vuoti
    if (!formData.title && !formData.content) {
      return;
    }

    setSaveStatus('saving');

    // Debounce di 2 secondi
    const saveTimer = setTimeout(async () => {
      try {
        // Simula chiamata API
        await new Promise(resolve => setTimeout(resolve, 500));

        console.log('Dati salvati:', formData);

        setSaveStatus('saved');
        setLastSaved(new Date());
      } catch (error) {
        setSaveStatus('error');
        console.error('Errore nel salvataggio:', error);
      }
    }, 2000);

    // Cleanup: cancella il timer se formData cambia prima di 2 secondi
    return () => {
      clearTimeout(saveTimer);
    };
  }, [formData]);

  const handleChange = (field, value) => {
    setFormData(prev => ({ ...prev, [field]: value }));
  };

  return (
    <div className="auto-save-form">
      <div className="status-bar">
        {saveStatus === 'saving' && <span>Salvataggio in corso...</span
            >}
        {saveStatus === 'saved' && (
          <span>
            Salvato {lastSaved && `alle ${lastSaved.toLocaleTimeString()
               }`}
          </span>
        )}
        {saveStatus === 'error' && <span className="error">Errore
            salvataggio</span>}
      </div>

      <input
        type="text"
        value={formData.title}
        onChange={(e) => handleChange('title', e.target.value)}
        placeholder="Titolo"
      />

      <textarea
        value={formData.content}
        onChange={(e) => handleChange('content', e.target.value)}
        placeholder="Contenuto"
        rows={10}
      />
    </div>
  );
}
```

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

```
# Installazione con npm
npm install react-router-dom

# Installazione con yarn
yarn add react-router-dom

# Installazione con pnpm
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

```
// main.jsx o index.jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

9.2.2 Componente App con Routes

Listing 9.3: App.jsx con routes di base

```
import { Routes, Route, Link } from 'react-router-dom';
import Home from './pages/Home';
import About from './pages/About';
import Contact from './pages/Contact';
import NotFound from './pages/NotFound';

function App() {
  return (
    <div className="app">
      {/* Navigation */}
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          <li><Link to="/about">About</Link></li>
          <li><Link to="/contact">Contact</Link></li>
        </ul>
      </nav>

      {/* Routes */}
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </div>
  );
}

export default App;
```

9.2.3 Componenti Pagina di Base

Listing 9.4: Esempi di componenti pagina

```
// pages/Home.jsx
function Home() {
  return (
    <div>
      <h1>Home Page</h1>
      <p>Benvenuto nella home page!</p>
    </div>
  );
}

export default Home;

// pages/About.jsx
function About() {
  return (
    <div>
      <h1>About Page</h1>
      <p>Informazioni sull'applicazione</p>
    </div>
  );
}

export default About;

// pages/NotFound.jsx
import { Link } from 'react-router-dom';

function NotFound() {
  return (
    <div>
      <h1>404 - Pagina Non Trovata</h1>
      <p>La pagina che stai cercando non esiste.</p>
      <Link to="/">Torna alla Home</Link>
    </div>
  );
}

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

```
import { Link } from 'react-router-dom';

function Navigation() {
  return (
    <nav>
      {/* Link semplice */}
      <Link to="/">Home</Link>

      {/* Link con classi CSS */}
      <Link to="/about" className="nav-link">About</Link>

      {/* Link con inline style */}
      <Link to="/contact" style={{ color: 'blue' }}>Contact</Link>

      {/* Link con state (passa dati alla route) */}
      <Link to="/profile" state={{ from: 'navigation' }}>
        Profile
      </Link>
    </nav>
  );
}
```

9.3.2 NavLink Component

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

Listing 9.6: NavLink per navigazione attiva

```
import { NavLink } from 'react-router-dom';

function Navigation() {
  return (
    <nav>
      {/* NavLink con classe attiva automatica */}
      <NavLink
        to="/"
        className={({ isActive }) => isActive ? 'active' : ''}
      >
        Home
      </NavLink>

      {/* NavLink con stile attivo */}
      <NavLink
        to="/about"
        style={({ isActive }) => ({
          color: isActive ? 'red' : 'blue',
          fontWeight: isActive ? 'bold' : 'normal'
        })}
      >
        About
      </NavLink>

      {/* NavLink con end (esatto match del path) */}
      <NavLink
        to="/"
        end
        className={({ isActive }) => isActive ? 'active' : ''}
      >
        Home (exact)
      </NavLink>
    </nav>
  );
}

// CSS
// .active {
//   color: red;
//   font-weight: bold;
//   border-bottom: 2px solid red;
// }
```

9.3.3 Navigazione Programmatica

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

Listing 9.7: useNavigate per navigazione programmatica

```
import { useNavigate } from 'react-router-dom';

function LoginForm() {
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();

    // Simula login
    const success = await login(username, password);

    if (success) {
      // Naviga alla dashboard dopo login
      navigate('/dashboard');

      // Naviga con replace (non aggiunge alla history)
      // navigate('/dashboard', { replace: true });

      // Naviga passando state
      // navigate('/dashboard', { state: { user: userData } });
    }
  };

  // Torna indietro
  const goBack = () => {
    navigate(-1); // Come browser back button
  };

  // Vai avanti
  const goForward = () => {
    navigate(1);
  };

  return (
    <form onSubmit={handleSubmit}>
      {/* form fields */}
      <button type="submit">Login</button>
      <button type="button" onClick={goBack}>Indietro</button>
    </form>
  );
}
```

9.4 Parametri URL

9.4.1 Route Parameters

I parametri URL permettono di creare route dinamiche.

Listing 9.8: Route con parametri

```
import { Routes, Route } from 'react-router-dom';
import UserProfile from './pages/UserProfile';
import BlogPost from './pages/BlogPost';

function App() {
  return (
    <Routes>
      {/* Parametro singolo */}
      <Route path="/users/:userId" element={<UserProfile />} />

      {/* Parametri multipli */}
      <Route path="/blog/:category/:postId" element={<BlogPost />} />

      {/* Parametro opzionale (con ?) */}
      <Route path="/products/:id?" element={<Products />} />
    </Routes>
  );
}
```

9.4.2 useParams Hook

Accedere ai parametri URL con `useParams`.

Listing 9.9: Accesso ai parametri con useParams

```
import { useParams } from 'react-router-dom';
import { useState, useEffect } from 'react';

function UserProfile() {
  const { userId } = useParams(); // Estrai parametro dall'URL
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchUser = async () => {
      setLoading(true);
      try {
        const response = await fetch(`/api/users/${userId}`);
        const data = await response.json();
        setUser(data);
      } catch (error) {
        console.error('Errore:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchUser();
  }, [userId]); // Ri-esegui quando userId cambia

  if (loading) return <div>Caricamento...</div>;
  if (!user) return <div>Utente non trovato</div>;

  return (
    <div>
      <h1>{user.name}</h1>
      <p>Email: {user.email}</p>
      <p>User ID: {userId}</p>
    </div>
  );
}
```

9.4.3 Esempio: Blog con Parametri Multipli

Listing 9.10: Blog post con categoria e ID

```
import { useParams, Link } from 'react-router-dom';

function BlogPost() {
  const { category, postId } = useParams();
  const [post, setPost] = useState(null);

  useEffect(() => {
    // Fetch post based on category and postId
    fetch(`/api/blog/${category}/${postId}`)
      .then(res => res.json())
      .then(setPost);
  }, [category, postId]);

  if (!post) return <div>Loading...</div>;

  return (
    <article>
      <div className="breadcrumb">
        <Link to="/blog">Blog</Link> /
        <Link to={`/blog/${category}`}>{category}</Link> /
        {post.title}
      </div>

      <h1>{post.title}</h1>
      <p className="category">Categoria: {category}</p>
      <div className="content">{post.content}</div>
    </article>
  );
}
```

9.5 Query Parameters

9.5.1 useSearchParams Hook

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

Listing 9.11: Gestione query parameters

```
import { useSearchParams } from 'react-router-dom';

function ProductList() {
  const [searchParams, setSearchParams] = useSearchParams();

  // Leggi query params
  const page = searchParams.get('page') || '1';
  const sort = searchParams.get('sort') || 'name';
  const category = searchParams.get('category');

  // Aggiorna query params
  const updatePage = (newPage) => {
    setSearchParams({ page: newPage, sort, category });
  };

  const updateSort = (newSort) => {
    setSearchParams({ page, sort: newSort, category });
  };

  // Rimuovi un query param
  const clearCategory = () => {
    const params = new URLSearchParams(searchParams);
    params.delete('category');
    setSearchParams(params);
  };

  return (
    <div>
      <h1>Prodotti</h1>

      {/* Filtri */}
      <div>
        <button onClick={() => updateSort('name')}>Ordina per Nome</
            button>
        <button onClick={() => updateSort('price')}>Ordina per Prezzo</
            button>
      </div>

      {/* Paginazione */}
      <div>
        <button onClick={() => updatePage(parseInt(page) - 1)}>
          Precedente
        </button>
        <span>Pagina {page}</span>
        <button onClick={() => updatePage(parseInt(page) + 1)}>
          Successivo
        </button>
      </div>

      {/* Mostra filtri attivi */}
      <div>
        Ordinamento: {sort}
        {category && (
          <span>
            , Categoria: {category}
            <button onClick={clearCategory}>X</button>
          </span>
        )}
      </div>
    </div>
  );
}
```

9.5.2 Esempio Completo: Ricerca e Filtri

Listing 9.12: Sistema di ricerca con filtri URL

```
import { useSearchParams, useNavigate } from 'react-router-dom';
import { useState, useEffect } from 'react';

function ProductSearch() {
  const [searchParams, setSearchParams] = useSearchParams();
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(false);

  // Estrai parametri dalla URL
  const query = searchParams.get('q') || '';
  const category = searchParams.get('category') || 'all';
  const minPrice = searchParams.get('minPrice') || '';
  const maxPrice = searchParams.get('maxPrice') || '';

  // Form state locale
  const [searchInput, setSearchInput] = useState(query);

  // Fetch prodotti quando cambiano i parametri
  useEffect(() => {
    const fetchProducts = async () => {
      setLoading(true);
      const params = new URLSearchParams(searchParams);
      const response = await fetch(`/api/products?${params}`);
      const data = await response.json();
      setProducts(data);
      setLoading(false);
    };

    fetchProducts();
  }, [searchParams]);

  // Handler per il form di ricerca
  const handleSearch = (e) => {
    e.preventDefault();
    setSearchParams(prev => {
      const params = new URLSearchParams(prev);
      if (searchInput) {
        params.set('q', searchInput);
      } else {
        params.delete('q');
      }
      return params;
    });
  };

  // Handler per cambio categoria
  const handleCategoryChange = (newCategory) => {
    setSearchParams(prev => {
      const params = new URLSearchParams(prev);
      params.set('category', newCategory);
      return params;
    });
  };

  // Handler per filtro prezzo
  const handlePriceFilter = (min, max) => {
    setSearchParams(prev => {
      const params = new URLSearchParams(prev);
      if (min) params.set('minPrice', min);
      else params.delete('minPrice');
      if (max) params.set('maxPrice', max);
      else params.delete('maxPrice');
      return params;
    });
  };

  // Reset tutti i filtri
  const resetFilters = () => {
    setSearchParams({});
    setSearchInput('');
  };

  return (
    <div className="product-search">
      {/* Search Form */}
      <form onSubmit={handleSearch}>
        <input
          type="text"
          value={searchInput}
          onChange={(e) => setSearchInput(e.target.value)}
          placeholder="Cerca prodotti..."
        />
        <button type="submit">Cerca</button>
      </form>

      {/* Filters */}
      <div className="filters">
        <select
          value={category}
          onChange={(e) => handleCategoryChange(e.target.value)}
        >
          <option value="all">Tutte le categorie</option>
          <option value="electronics">Elettronica</option>
          <option value="clothing">Abbigliamento</option>
          <option value="books">Libri</option>
        </select>

        <input
          type="number"
          placeholder="Prezzo min"
          value={minPrice}
          onChange={(e) => handlePriceFilter(e.target.value, maxPrice)}
        />

        <input
          type="number"
          placeholder="Prezzo max"
          value={maxPrice}
          onChange={(e) => handlePriceFilter(minPrice, e.target.value)}
        />

        <button onClick={resetFilters}>Reset Filtri</button>
      </div>

      {/* Active Filters Display */}
      <div className="active-filters">
        {query && <span>Ricerca: "{query}"</span>}
        {category !== 'all' && <span>Categoria: {category}</span>}
        {(minPrice || maxPrice) && (
          <span>
            Prezzo: {minPrice || '0'} - {maxPrice || '   '}
          </span>
        )}
      </div>

      {/* Results */}
      {loading ? (
        <div>Caricamento...</div>
      ) : (
        <div className="products">
          {products.map(product => (
            <div key={product.id} className="product">
              <h3>{product.name}</h3>
              <p>{product.price}   </p>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}
```

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

```
import { Outlet, Link } from 'react-router-dom';

// Layout principale con sidebar
function DashboardLayout() {
  return (
    <div className="dashboard">
      <aside className="sidebar">
        <h2>Dashboard</h2>
        <nav>
          <Link to="/dashboard">Overview</Link>
          <Link to="/dashboard/stats">Statistiche</Link>
          <Link to="/dashboard/settings">Impostazioni</Link>
          <Link to="/dashboard/users">Utenti</Link>
        </nav>
      </aside>

      <main className="content">
        {/* Qui vengono renderizzate le route figlie */}
        <Outlet />
      </main>
    </div>
  );
}

export default DashboardLayout;
```

9.6.2 Configurazione Route Nidificate

Listing 9.14: Setup di route nidificate

```
import { Routes, Route } from 'react-router-dom';
import DashboardLayout from './layouts/DashboardLayout';
import DashboardHome from './pages/dashboard/Home';
import DashboardStats from './pages/dashboard/Stats';
import DashboardSettings from './pages/dashboard/Settings';
import DashboardUsers from './pages/dashboard/Users';

function App() {
  return (
    <Routes>
      {/* Route principale con nested routes */}
      <Route path="/dashboard" element={<DashboardLayout />}>
        {/* Index route (default) */}
        <Route index element={<DashboardHome />} />

        {/* Nested routes */}
        <Route path="stats" element={<DashboardStats />} />
        <Route path="settings" element={<DashboardSettings />} />
        <Route path="users" element={<DashboardUsers />} />
      </Route>
    </Routes>
  );
}
```

9.6.3 Route Nidificate Multilivello

Listing 9.15: Nesting profondo con più livelli

```
import { Routes, Route } from 'react-router-dom';

function App() {
  return (
    <Routes>
      <Route path="/" element={<RootLayout />}>
        <Route index element={<Home />} />

        {/* Blog con nested routes */}
        <Route path="blog" element={<BlogLayout />}>
          <Route index element={<BlogHome />} />
          <Route path=":category" element={<CategoryLayout />}>
            <Route index element={<CategoryHome />} />
            <Route path=":postId" element={<BlogPost />} />
          </Route>
        </Route>

        {/* Admin con nested routes */}
        <Route path="admin" element={<AdminLayout />}>
          <Route index element={<AdminDashboard />} />
          <Route path="users" element={<UsersLayout />}>
            <Route index element={<UsersList />} />
            <Route path=":userId" element={<UserDetail />} />
            <Route path=":userId/edit" element={<UserEdit />} />
          </Route>
        </Route>
      </Route>
    </Routes>
  );
}
```

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

```
import { Navigate, useLocation } from 'react-router-dom';
import { useAuth } from './hooks/useAuth'; // Custom hook per auth

function ProtectedRoute({ children }) {
  const { isAuthenticated, loading } = useAuth();
  const location = useLocation();

  if (loading) {
    return <div>Caricamento...</div>;
  }

  if (!isAuthenticated) {
    // Redirect al login, salvando la location corrente
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  return children;
}

export default ProtectedRoute;
```

9.7.2 Utilizzo delle Protected Routes

Listing 9.17: Setup di route protette

```
import { Routes, Route } from 'react-router-dom';
import ProtectedRoute from './components/ProtectedRoute';

function App() {
  return (
    <Routes>
      {/* Route pubbliche */}
      <Route path="/" element={<Home />} />
      <Route path="/login" element={<Login />} />
      <Route path="/register" element={<Register />} />

      {/* Route protette */}
      <Route
        path="/dashboard"
        element={
          <ProtectedRoute>
            <Dashboard />
          </ProtectedRoute>
        }
      />

      <Route
        path="/profile"
        element={
          <ProtectedRoute>
            <Profile />
          </ProtectedRoute>
        }
      />

      {/* Nested protected routes */}
      <Route
        path="/admin"
        element={
          <ProtectedRoute>
            <AdminLayout />
          </ProtectedRoute>
        }
      >
        <Route index element={<AdminDashboard />} />
        <Route path="users" element={<AdminUsers />} />
      </Route>
    </Routes>
  );
}
```

9.7.3 Hook useAuth Personalizzato

Listing 9.18: Custom hook per autenticazione

```
import { createContext, useContext, useState, useEffect } from 'react';

const AuthContext = createContext(null);

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Controlla se c'è un utente loggato (es. da localStorage)
    const checkAuth = async () => {
      try {
        const token = localStorage.getItem('token');
        if (token) {
          const response = await fetch('/api/auth/me', {
            headers: { Authorization: `Bearer ${token}` }
          });
          if (response.ok) {
            const userData = await response.json();
            setUser(userData);
          }
        }
      } catch (error) {
        console.error('Auth check failed:', error);
      } finally {
        setLoading(false);
      }
    };

    checkAuth();
  }, []);

  const login = async (email, password) => {
    const response = await fetch('/api/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password })
    });

    if (response.ok) {
      const { user, token } = await response.json();
      localStorage.setItem('token', token);
      setUser(user);
      return true;
    }
    return false;
  };

  const logout = () => {
    localStorage.removeItem('token');
    setUser(null);
  };

  return (
    <AuthContext.Provider
      value={{
        user,
        isAuthenticated: !!user,
        loading,
        login,
        logout
      }}
    >
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth deve essere usato dentro AuthProvider');
  }
  return context;
}
```

9.7.4 Route con Ruoli (Role-Based)

Listing 9.19: Route protette per ruolo

```
import { Navigate } from 'react-router-dom';
import { useAuth } from './hooks/useAuth';

function RoleProtectedRoute({ children, allowedRoles }) {
  const { user, isAuthenticated, loading } = useAuth();

  if (loading) {
    return <div>Caricamento...</div>;
  }

  if (!isAuthenticated) {
    return <Navigate to="/login" replace />;
  }

  if (allowedRoles && !allowedRoles.includes(user.role)) {
    return <Navigate to="/unauthorized" replace />;
  }

  return children;
}

// Utilizzo
function App() {
  return (
    <Routes>
      {/* Solo admin */}
      <Route
        path="/admin"
        element={
          <RoleProtectedRoute allowedRoles={['admin']}>
            <AdminPanel />
          </RoleProtectedRoute>
        }
      />

      {/* Admin e moderatori */}
      <Route
        path="/moderation"
        element={
          <RoleProtectedRoute allowedRoles={['admin', 'moderator']}>
            <ModerationPanel />
          </RoleProtectedRoute>
        }
      />

      {/* Tutti gli utenti autenticati */}
      <Route
        path="/dashboard"
        element={
          <RoleProtectedRoute>
```

```
            <Dashboard />
          </RoleProtectedRoute>
        }
      />
    </Routes>
  );
}
```

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

```
import { lazy, Suspense } from 'react';
import { Routes, Route } from 'react-router-dom';

// Lazy import
const Home = lazy(() => import('./pages/Home'));
const About = lazy(() => import('./pages/About'));
const Dashboard = lazy(() => import('./pages/Dashboard'));
const Profile = lazy(() => import('./pages/Profile'));

function App() {
  return (
    <Suspense fallback={<div>Caricamento...</div>}>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/profile" element={<Profile />} />
      </Routes>
    </Suspense>
  );
}
```

9.8.2 Loading Component Personalizzato

Listing 9.21: Componente loading personalizzato

```
function LoadingSpinner() {
  return (
    <div className="loading-container">
      <div className="spinner"></div>
      <p>Caricamento in corso...</p>
    </div>
  );
}

// CSS per spinner
// .spinner {
//   border: 4px solid #f3f3f3;
//   border-top: 4px solid #3498db;
//   border-radius: 50%;
//   width: 40px;
//   height: 40px;
//   animation: spin 1s linear infinite;
// }
//
// @keyframes spin {
//   0% { transform: rotate(0deg); }
//   100% { transform: rotate(360deg); }
// }

function App() {
  return (
    <Suspense fallback={<LoadingSpinner />}>
      <Routes>
        {/* routes */}
      </Routes>
    </Suspense>
  );
}
```

9.8.3 Lazy Loading con Error Boundary

Listing 9.22: Gestione errori nel lazy loading

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

class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false, error: null };
  }

  static getDerivedStateFromError(error) {
    return { hasError: true, error };
  }

  componentDidCatch(error, errorInfo) {
    console.error('Error loading component:', error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="error-container">
          <h2>Errore nel caricamento</h2>
          <p>{this.state.error?.message}</p>
          <button onClick={() => window.location.reload()}>
            Ricarica Pagina
          </button>
        </div>
      );
    }

    return this.props.children;
  }
}
```

```

// Utilizzo
function App() {
  return (
    <ErrorBoundary>
      <Suspense fallback={<LoadingSpinner />}>
        <Routes>
          <Route path="/" element={<LazyHome />} />
          <Route path="/dashboard" element={<LazyDashboard />} />
        </Routes>
      </Suspense>
    </ErrorBoundary>
  );
}
```

9.8.4 Preloading delle Route

Listing 9.23: Precaricamento route al hover

```
import { lazy } from 'react';
import { Link } from 'react-router-dom';

// Lazy imports
const Dashboard = lazy(() => import('./pages/Dashboard'));

// Funzione per preload
const preloadDashboard = () => {
  import('./pages/Dashboard');
};

function Navigation() {
  return (
    <nav>
      <Link to="/">Home</Link>

      {/* Preload al mouseenter */}
      <Link
        to="/dashboard"
        onMouseEnter={preloadDashboard}
        onFocus={preloadDashboard}
      >
        Dashboard
      </Link>
    </nav>
  );
}
```

9.9 useLocation Hook

`useLocation` fornisce accesso alla location corrente.

Listing 9.24: Utilizzo di useLocation

```
import { useLocation, useNavigate } from 'react-router-dom';
import { useEffect } from 'react';

function Analytics() {
  const location = useLocation();

  useEffect(() => {
    // Track page view
    console.log('Page view:', location.pathname);
    // analytics.track('page_view', { path: location.pathname });
  }, [location]);

  return null;
}

function LoginPage() {
  const location = useLocation();
  const navigate = useNavigate();

  const handleLogin = async (credentials) => {
    const success = await login(credentials);

    if (success) {
      // Redirect alla pagina da cui l'utente veniva
      const from = location.state?.from?.pathname || '/dashboard';
      navigate(from, { replace: true });
    }
  };

  return (
    <div>
      <h1>Login</h1>
      {location.state?.from && (
        <p>Devi effettuare il login per accedere a questa pagina</p>
      )}
      {/* form */}
    </div>
  );
}
```

9.10 Applicazione Completa

9.10.1 Struttura del Progetto

Listing 9.25: Struttura directory consigliata

```
src/
          components/
                common/
                      LoadingSpinner.jsx
                      ErrorBoundary.jsx
                      Navigation.jsx
                auth/
                      ProtectedRoute.jsx
                      RoleProtectedRoute.jsx
                layout/
                    RootLayout.jsx
                    DashboardLayout.jsx
                    AdminLayout.jsx
          pages/
                Home.jsx
                About.jsx
                Login.jsx
                Register.jsx
                NotFound.jsx
                dashboard/
                      Dashboard.jsx
                      Stats.jsx
                      Settings.jsx
                admin/
                    AdminPanel.jsx
                    Users.jsx
          hooks/
                useAuth.js
          context/
                AuthContext.jsx
          App.jsx
          main.jsx
```

9.10.2 App.jsx Completa

Listing 9.26: Configurazione completa dell'app

```
import { lazy, Suspense } from 'react';
import { Routes, Route } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import ErrorBoundary from './components/common/ErrorBoundary';
import LoadingSpinner from './components/common/LoadingSpinner';
import ProtectedRoute from './components/auth/ProtectedRoute';
import RoleProtectedRoute from './components/auth/RoleProtectedRoute';
import RootLayout from './components/layout/RootLayout';
import DashboardLayout from './components/layout/DashboardLayout';
import AdminLayout from './components/layout/AdminLayout';

// Lazy loaded pages
const Home = lazy(() => import('./pages/Home'));
const About = lazy(() => import('./pages/About'));
const Login = lazy(() => import('./pages/Login'));
const Register = lazy(() => import('./pages/Register'));
const Dashboard = lazy(() => import('./pages/dashboard/Dashboard'));
const Stats = lazy(() => import('./pages/dashboard/Stats'));
const Settings = lazy(() => import('./pages/dashboard/Settings'));
const AdminPanel = lazy(() => import('./pages/admin/AdminPanel'));
const AdminUsers = lazy(() => import('./pages/admin/Users'));
const NotFound = lazy(() => import('./pages/NotFound'));

function App() {
  return (
    <AuthProvider>
      <ErrorBoundary>
        <Suspense fallback={<LoadingSpinner />}>
          <Routes>
            {/* Root layout */}
            <Route path="/" element={<RootLayout />}>
              {/* Public routes */}
              <Route index element={<Home />} />
              <Route path="about" element={<About />} />
              <Route path="login" element={<Login />} />
              <Route path="register" element={<Register />} />

              {/* Protected dashboard routes */}
              <Route
                path="dashboard"
                element={
                  <ProtectedRoute>
                    <DashboardLayout />
                  </ProtectedRoute>
                }
              >
                <Route index element={<Dashboard />} />
                <Route path="stats" element={<Stats />} />
                <Route path="settings" element={<Settings />} />
              </Route>

              {/* Admin routes (role-based) */}
              <Route
                path="admin"
                element={
                  <RoleProtectedRoute allowedRoles={['admin']}>
                    <AdminLayout />
                  </RoleProtectedRoute>
                }
              >
                <Route index element={<AdminPanel />} />
                <Route path="users" element={<AdminUsers />} />
              </Route>

              {/* 404 */}
              <Route path="*" element={<NotFound />} />
            </Route>
          </Routes>
        </Suspense>
      </ErrorBoundary>
    </AuthProvider>
  );
}

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

```
// Componente Root
function App() {
  const [user, setUser] = useState(null);

  return (
    <Layout user={user} setUser={setUser}>
      <Dashboard user={user} setUser={setUser} />
    </Layout>
  );
}

// Layout deve passare props anche se non le usa
function Layout({ user, setUser, children }) {
  return (
    <div>
      <Header user={user} setUser={setUser} />
      <main>{children}</main>
    </div>
  );
}

// Header finalmente usa le props
function Header({ user, setUser }) {
  return (
    <header>
      {user ? (
        <UserMenu user={user} onLogout={() => setUser(null)} />
      ) : (
        <LoginButton />
      )}
    </header>
  );
}
```

```

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

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

```
import { createContext, useContext, useState } from 'react';

// 1. Crea il Context
const ThemeContext = createContext(null);

// 2. Provider Component
export function ThemeProvider({ children }) {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme(prev => prev === 'light' ? 'dark' : 'light');
  };

  const value = {
    theme,
    toggleTheme
  };

  return (
    <ThemeContext.Provider value={value}>
      {children}
    </ThemeContext.Provider>
  );
}

// 3. Custom Hook per usare il context
export function useTheme() {
  const context = useContext(ThemeContext);

  if (context === null) {
    throw new Error('useTheme deve essere usato dentro ThemeProvider');
  }

  return context;
}
```

10.2.2 Utilizzo del Context

Listing 10.3: Usare il context nell'app

```
// main.jsx
import { ThemeProvider } from './context/ThemeContext';

ReactDOM.createRoot(document.getElementById('root')).render(
  <ThemeProvider>
    <App />
  </ThemeProvider>
);

// Componente che usa il tema
function Header() {
  const { theme, toggleTheme } = useTheme();

  return (
    <header className={theme}>
      <h1>My App</h1>
      <button onClick={toggleTheme}>
        Toggle Theme (Current: {theme})
      </button>
    </header>
  );
}

// Altro componente che usa il tema
function Card({ children }) {
  const { theme } = useTheme();

  return (
    <div className={`card card-${theme}`}>
      {children}
    </div>
  );
}
```

10.2.3 Context per Autenticazione

Listing 10.4: Auth Context completo

```
import { createContext, useContext, useState, useEffect } from 'react';

const AuthContext = createContext(null);

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  // Controlla auth al mount
  useEffect(() => {
    const checkAuth = async () => {
      const token = localStorage.getItem('authToken');

      if (token) {
        try {
          const response = await fetch('/api/auth/me', {
            headers: { Authorization: `Bearer ${token}` }
          });

          if (response.ok) {
            const userData = await response.json();
            setUser(userData);
          } else {
            localStorage.removeItem('authToken');
          }
        } catch (error) {
          console.error('Auth check failed:', error);
        }
      }

      setLoading(false);
    };

    checkAuth();
  }, []);

  const login = async (email, password) => {
    try {
      const response = await fetch('/api/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, password })
      });

      if (!response.ok) {
        throw new Error('Login failed');
      }

      const { user, token } = await response.json();
      localStorage.setItem('authToken', token);
      setUser(user);
      return { success: true };
    } catch (error) {
      return { success: false, error: error.message };
    }
  };

  const logout = () => {
    localStorage.removeItem('authToken');
    setUser(null);
  };

  const register = async (userData) => {
    try {
      const response = await fetch('/api/auth/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(userData)
      });

      if (!response.ok) {
        throw new Error('Registration failed');
      }

      const { user, token } = await response.json();
      localStorage.setItem('authToken', token);
      setUser(user);
      return { success: true };
    } catch (error) {
      return { success: false, error: error.message };
    }
  };

  const value = {
    user,
    loading,
    isAuthenticated: !!user,
    login,
    logout,
    register
  };

  return (
    <AuthContext.Provider value={value}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  const context = useContext(AuthContext);

  if (context === null) {
    throw new Error('useAuth deve essere usato dentro AuthProvider');
  }

  return context;
}
```

10.2.4 Context con Reducer

Per logica complessa, combina Context con useReducer.

Listing 10.5: Context con useReducer per carrello

```
import { createContext, useContext, useReducer } from 'react';

const CartContext = createContext(null);

// Reducer
function cartReducer(state, action) {
  switch (action.type) {
    case 'ADD_ITEM':
      const existingIndex = state.items.findIndex(
        item => item.id === action.payload.id
      );

      if (existingIndex >= 0) {
        const newItems = [...state.items];
        newItems[existingIndex].quantity += 1;
        return { ...state, items: newItems };
      }

      return {
        ...state,
        items: [...state.items, { ...action.payload, quantity: 1 }]
      };

    case 'REMOVE_ITEM':
      return {
        ...state,
        items: state.items.filter(item => item.id !== action.payload)
      };

    case 'UPDATE_QUANTITY':
      return {
        ...state,
        items: state.items.map(item =>
          item.id === action.payload.id
            ? { ...item, quantity: action.payload.quantity }
            : item
        )
      };

    case 'CLEAR_CART':
      return { ...state, items: [] };

    default:
      return state;
  }
}

// Provider
export function CartProvider({ children }) {
  const [state, dispatch] = useReducer(cartReducer, { items: [] });

  // Selectors
  const totalItems = state.items.reduce(
    (sum, item) => sum + item.quantity,
    0
  );

  const totalPrice = state.items.reduce(
    (sum, item) => sum + (item.price * item.quantity),
    0
  );

  // Actions
  const addItem = (item) => {
    dispatch({ type: 'ADD_ITEM', payload: item });
  };

  const removeItem = (itemId) => {
    dispatch({ type: 'REMOVE_ITEM', payload: itemId });
  };

  const updateQuantity = (itemId, quantity) => {
    if (quantity <= 0) {
      removeItem(itemId);
    } else {
      dispatch({
        type: 'UPDATE_QUANTITY',
        payload: { id: itemId, quantity }
      });
    }
  };

  const clearCart = () => {
    dispatch({ type: 'CLEAR_CART' });
  };

  const value = {
    items: state.items,
    totalItems,
    totalPrice,
    addItem,
    removeItem,
    updateQuantity,
    clearCart
  };

  return (
    <CartContext.Provider value={value}>
      {children}
    </CartContext.Provider>
  );
}

export function useCart() {
  const context = useContext(CartContext);

  if (context === null) {
    throw new Error('useCart deve essere usato dentro CartProvider');
  }

  return context;
}
```

10.2.5 Utilizzo del Cart Context

Listing 10.6: Componenti che usano il carrello

```
// Componente Product
function ProductCard({ product }) {
  const { addItem } = useCart();

  return (
    <div className="product-card">
      <h3>{product.name}</h3>
      <p>{product.price}   </p>
      <button onClick={() => addItem(product)}>
        Aggiungi al Carrello
      </button>
    </div>
  );
}

// Componente CartIcon
function CartIcon() {
  const { totalItems, totalPrice } = useCart();

  return (
    <div className="cart-icon">
      <ShoppingCartIcon />
      <span className="badge">{totalItems}</span>
      <span className="price">{totalPrice.toFixed(2)}   </span>
    </div>
  );
}

// Componente Cart
function Cart() {
  const { items, updateQuantity, removeItem, clearCart } = useCart();

  return (
    <div className="cart">
      <h2>Carrello</h2>

      {items.length === 0 ? (
        <p>Il carrello è vuoto</p>
      ) : (
        <>
          {items.map(item => (
            <div key={item.id} className="cart-item">
              <h4>{item.name}</h4>
              <p>{item.price}   </p>

              <div className="quantity-controls">
                <button onClick={() => updateQuantity(item.id, item.
                   quantity - 1)}>
                  -
                </button>
                <span>{item.quantity}</span>
                <button onClick={() => updateQuantity(item.id, item.
                   quantity + 1)}>
                  +
                </button>
              </div>

              <button onClick={() => removeItem(item.id)}>
                Rimuovi
              </button>
            </div>
          ))}

          <button onClick={clearCart}>Svuota Carrello</button>
        </>
      )}
    </div>
  );
}
```

10.2.6 Performance: Splitting Context

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

Listing 10.7: Separazione context per performance

```
// SBAGLIATO: Un unico context con troppi dati
const AppContext = createContext(null);

function AppProvider({ children }) {
  const [user, setUser] = useState(null);
  const [theme, setTheme] = useState('light');
  const [notifications, setNotifications] = useState([]);
  const [settings, setSettings] = useState({});

  // Ogni cambiamento ri-renderizza tutti i consumer!
  return (
    <AppContext.Provider value={{
      user, setUser,
      theme, setTheme,
      notifications, setNotifications,
      settings, setSettings
    }}>
      {children}
    </AppContext.Provider>
  );
}

// CORRETTO: Context separati
// UserContext.jsx
const UserContext = createContext(null);
export function UserProvider({ children }) {
  const [user, setUser] = useState(null);
  return (
    <UserContext.Provider value={{ user, setUser }}>
      {children}
    </UserContext.Provider>
  );
}

// ThemeContext.jsx
const ThemeContext = createContext(null);
export function ThemeProvider({ children }) {
  const [theme, setTheme] = useState('light');
  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

// App.jsx - Composizione providers
function App() {
  return (
    <UserProvider>
      <ThemeProvider>
        <NotificationProvider>
          <SettingsProvider>
            <AppContent />
          </SettingsProvider>
        </NotificationProvider>
      </ThemeProvider>
    </UserProvider>
  );
}
```

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

```
npm install @reduxjs/toolkit react-redux

# o con yarn
yarn add @reduxjs/toolkit react-redux
```

10.3.2 Setup Store

Listing 10.9: Configurazione base dello store

```
// store/store.js
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './slices/counterSlice';
import authReducer from './slices/authSlice';
import cartReducer from './slices/cartSlice';

export const store = configureStore({
  reducer: {
    counter: counterReducer,
    auth: authReducer,
    cart: cartReducer
  }
});
```

10.3.3 Provider Setup

Listing 10.10: Wrap app con Provider

```jsx
// main.jsx
import { Provider } from 'react-redux';
import { store } from './store/store';

ReactDOM.createRoot(document.getElementById('root')).render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

10.3.4 Creazione di uno Slice

Listing 10.11: Counter slice esempio

```js
// store/slices/counterSlice.js
import { createSlice } from '@reduxjs/toolkit';

const counterSlice = createSlice({
  name: 'counter',

  initialState: {
    value: 0
  },

  reducers: {
    increment: (state) => {
      state.value += 1; // Immer permette mutazioni dirette
    },

    decrement: (state) => {
      state.value -= 1;
    },

    incrementByAmount: (state, action) => {
      state.value += action.payload;
    },

    reset: (state) => {
      state.value = 0;
    }
  }
});

// Export actions
export const { increment, decrement, incrementByAmount, reset } =
  counterSlice.actions;

// Export reducer
export default counterSlice.reducer;

// Selectors
export const selectCount = (state) => state.counter.value;
```

10.3.5 Utilizzo in Componenti

Listing 10.12: Usare Redux in componenti

```
import { useSelector, useDispatch } from 'react-redux';
import {
  increment,
  decrement,
  incrementByAmount,
  reset,
  selectCount
} from './store/slices/counterSlice';

function Counter() {
  const count = useSelector(selectCount);
  const dispatch = useDispatch();

  return (
    <div>
      <h2>Count: {count}</h2>

      <button onClick={() => dispatch(increment())}>+</button>
      <button onClick={() => dispatch(decrement())}>-</button>
      <button onClick={() => dispatch(incrementByAmount(5))}>+5</button>
      <button onClick={() => dispatch(reset())}>Reset</button>
    </div>
  );
}
```

10.3.6 Slice Complesso: Autenticazione

Listing 10.13: Auth slice con async thunks

```
// store/slices/authSlice.js
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

// Async thunk per login
export const loginUser = createAsyncThunk(
  'auth/login',
  async ({ email, password }, { rejectWithValue }) => {
    try {
      const response = await fetch('/api/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, password })
      });

      if (!response.ok) {
        throw new Error('Login failed');
      }

      const data = await response.json();
      localStorage.setItem('token', data.token);
      return data;
    } catch (error) {
      return rejectWithValue(error.message);
    }
  }
```

```
);

// Async thunk per fetch user
export const fetchUser = createAsyncThunk(
  'auth/fetchUser',
  async (_, { rejectWithValue }) => {
    try {
      const token = localStorage.getItem('token');

      if (!token) {
        throw new Error('No token');
      }

      const response = await fetch('/api/auth/me', {
        headers: { Authorization: `Bearer ${token}` }
      });

      if (!response.ok) {
        throw new Error('Failed to fetch user');
      }

      return await response.json();
    } catch (error) {
      return rejectWithValue(error.message);
    }
  }
);

const authSlice = createSlice({
  name: 'auth',

  initialState: {
    user: null,
    token: localStorage.getItem('token'),
    loading: false,
    error: null
  },

  reducers: {
    logout: (state) => {
      state.user = null;
      state.token = null;
      state.error = null;
      localStorage.removeItem('token');
    },

    clearError: (state) => {
      state.error = null;
    }
  },

  extraReducers: (builder) => {
    builder
      // Login
      .addCase(loginUser.pending, (state) => {
        state.loading = true;
        state.error = null;
      })
      .addCase(loginUser.fulfilled, (state, action) => {
        state.loading = false;
        state.user = action.payload.user;
        state.token = action.payload.token;
        state.error = null;
      })
      .addCase(loginUser.rejected, (state, action) => {
        state.loading = false;
        state.error = action.payload;
      })

      // Fetch User
      .addCase(fetchUser.pending, (state) => {
        state.loading = true;
      })
      .addCase(fetchUser.fulfilled, (state, action) => {
        state.loading = false;
        state.user = action.payload;
      })
      .addCase(fetchUser.rejected, (state, action) => {
        state.loading = false;
        state.token = null;
        state.error = action.payload;
        localStorage.removeItem('token');
      });
  }
});

export const { logout, clearError } = authSlice.actions;
export default authSlice.reducer;

// Selectors
export const selectUser = (state) => state.auth.user;
export const selectIsAuthenticated = (state) => !!state.auth.user;
export const selectAuthLoading = (state) => state.auth.loading;
export const selectAuthError = (state) => state.auth.error;
```

10.3.7 Utilizzo Auth Slice

Listing 10.14: Login component con Redux

```
import { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { useNavigate } from 'react-router-dom';
import {
  loginUser,
  selectAuthLoading,
  selectAuthError,
  clearError
} from './store/slices/authSlice';

function LoginPage() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const dispatch = useDispatch();
  const navigate = useNavigate();
  const loading = useSelector(selectAuthLoading);
  const error = useSelector(selectAuthError);

  const handleSubmit = async (e) => {
    e.preventDefault();

    const result = await dispatch(loginUser({ email, password }));

    if (loginUser.fulfilled.match(result)) {
      navigate('/dashboard');
    }
  };

  useEffect(() => {
    // Pulisci errori quando il componente si smonta
    return () => {
      dispatch(clearError());
    };
  }, [dispatch]);

  return (
    <div>
      <h1>Login</h1>

      {error && (
        <div className="error">
          {error}
          <button onClick={() => dispatch(clearError())}>X</button>
        </div>
      )}

      <form onSubmit={handleSubmit}>
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          placeholder="Email"
          required
        />

        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Password"
          required
        />

        <button type="submit" disabled={loading}>
          {loading ? 'Loading...' : 'Login'}
        </button>
      </form>
    </div>
  );
}
```

10.3.8 Cart Slice Completo

Listing 10.15: Carrello e-commerce con Redux

```
// store/slices/cartSlice.js
import { createSlice } from '@reduxjs/toolkit';

const cartSlice = createSlice({
  name: 'cart',

  initialState: {
    items: [],
    isOpen: false
  },

  reducers: {
    addItem: (state, action) => {
      const existingItem = state.items.find(
        item => item.id === action.payload.id
      );

      if (existingItem) {
        existingItem.quantity += 1;
      } else {
        state.items.push({ ...action.payload, quantity: 1 });
      }
    },

    removeItem: (state, action) => {
      state.items = state.items.filter(
        item => item.id !== action.payload
      );
    },

    updateQuantity: (state, action) => {
      const { id, quantity } = action.payload;
      const item = state.items.find(item => item.id === id);

      if (item) {
        if (quantity <= 0) {
          state.items = state.items.filter(item => item.id !== id);
        } else {
          item.quantity = quantity;
        }
      }
    },

    clearCart: (state) => {
      state.items = [];
    },

    toggleCart: (state) => {
      state.isOpen = !state.isOpen;
    },

    closeCart: (state) => {
      state.isOpen = false;
    }
  }
});

export const {
  addItem,
  removeItem,
  updateQuantity,
  clearCart,
  toggleCart,
  closeCart
} = cartSlice.actions;

export default cartSlice.reducer;

// Selectors con memoization
export const selectCartItems = (state) => state.cart.items;
export const selectIsCartOpen = (state) => state.cart.isOpen;

export const selectTotalItems = (state) =>
  state.cart.items.reduce((sum, item) => sum + item.quantity, 0);

export const selectTotalPrice = (state) =>
  state.cart.items.reduce(
    (sum, item) => sum + (item.price * item.quantity),
    0
  );
```

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

```
// store/api/apiSlice.js
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'
    ;

export const apiSlice = createApi({
  reducerPath: 'api',

  baseQuery: fetchBaseQuery({
    baseUrl: '/api',
    prepareHeaders: (headers, { getState }) => {
      const token = getState().auth.token;
      if (token) {
        headers.set('authorization', `Bearer ${token}`);
      }
      return headers;
    }
  }),

  tagTypes: ['User', 'Product', 'Order'],

  endpoints: (builder) => ({
    // Get all products
    getProducts: builder.query({
      query: () => '/products',
      providesTags: ['Product']
    }),

    // Get single product
    getProduct: builder.query({
      query: (id) => `/products/${id}`,
      providesTags: (result, error, id) => [{ type: 'Product', id }]
    }),

    // Create product
    createProduct: builder.mutation({
      query: (newProduct) => ({
        url: '/products',
        method: 'POST',
        body: newProduct
      }),
      invalidatesTags: ['Product']
    }),

    // Update product
    updateProduct: builder.mutation({
      query: ({ id, ...updates }) => ({
        url: `/products/${id}`,
        method: 'PUT',
        body: updates
      }),
      invalidatesTags: (result, error, { id }) => [{ type: 'Product', id
          }]
    }),

    // Delete product
    deleteProduct: builder.mutation({
      query: (id) => ({
        url: `/products/${id}`,
        method: 'DELETE'
      }),
      invalidatesTags: ['Product']
    }),

    // Get user orders
    getUserOrders: builder.query({
      query: () => '/orders',
      providesTags: ['Order']
    })
  })
});

export const {
  useGetProductsQuery,
  useGetProductQuery,
  useCreateProductMutation,
  useUpdateProductMutation,
  useDeleteProductMutation,
  useGetUserOrdersQuery
} = apiSlice;
```

10.3.10 Store con RTK Query

Listing 10.17: Store configuration con RTK Query

```
// store/store.js
import { configureStore } from '@reduxjs/toolkit';
import { apiSlice } from './api/apiSlice';
import authReducer from './slices/authSlice';
import cartReducer from './slices/cartSlice';

export const store = configureStore({
  reducer: {
    [apiSlice.reducerPath]: apiSlice.reducer,
    auth: authReducer,
    cart: cartReducer
  },

  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(apiSlice.middleware)
});
```

10.3.11 Utilizzo RTK Query

Listing 10.18: Componente con RTK Query

```
import {
  useGetProductsQuery,
  useCreateProductMutation,
  useDeleteProductMutation
} from './store/api/apiSlice';

function ProductList() {
  const {
    data: products,
    isLoading,
    isError,
    error,
    refetch
  } = useGetProductsQuery();

  const [createProduct, { isLoading: isCreating }] =
    useCreateProductMutation();

  const [deleteProduct, { isLoading: isDeleting }] =
    useDeleteProductMutation();

  const handleCreate = async () => {
    try {
      await createProduct({
        name: 'New Product',
        price: 99.99
      }).unwrap();
      // Successo
    } catch (error) {
      console.error('Failed to create:', error);
    }
  };

  const handleDelete = async (id) => {
    try {
      await deleteProduct(id).unwrap();
    } catch (error) {
      console.error('Failed to delete:', error);
    }
  };

  if (isLoading) return <div>Loading...</div>;
  if (isError) return <div>Error: {error.message}</div>;

  return (
    <div>
      <h1>Products</h1>

      <button onClick={handleCreate} disabled={isCreating}>
        {isCreating ? 'Creating...' : 'Add Product'}
      </button>

      <button onClick={refetch}>Refresh</button>

      <ul>
        {products.map(product => (
          <li key={product.id}>
            {product.name} - {product.price}
            <button
              onClick={() => handleDelete(product.id)}
              disabled={isDeleting}
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

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

```
npm install zustand

# o con yarn
yarn add zustand
```

10.4.2 Store Base

Listing 10.20: Creazione di uno store Zustand

```
// store/useCounterStore.js
```

```
import { create } from 'zustand';

const useCounterStore = create((set) => ({
  // State
  count: 0,

  // Actions
  increment: () => set((state) => ({ count: state.count + 1 })),
  decrement: () => set((state) => ({ count: state.count - 1 })),
  incrementByAmount: (amount) =>
    set((state) => ({ count: state.count + amount })),
  reset: () => set({ count: 0 })
}));

export default useCounterStore;
```

10.4.3 Utilizzo in Componenti

Listing 10.21: Usare Zustand in componenti

```
import useCounterStore from './store/useCounterStore';

function Counter() {
  // Seleziona solo ciò che serve
  const count = useCounterStore((state) => state.count);
  const increment = useCounterStore((state) => state.increment);
  const decrement = useCounterStore((state) => state.decrement);
  const reset = useCounterStore((state) => state.reset);

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}
```

10.4.4 Auth Store con Zustand

Listing 10.22: Store autenticazione completo

```
// store/useAuthStore.js
import { create } from 'zustand';
import { persist } from 'zustand/middleware';

const useAuthStore = create(
  persist(
    (set, get) => ({
      // State
      user: null,
      token: null,
      loading: false,
      error: null,

      // Actions
      login: async (email, password) => {
        set({ loading: true, error: null });

        try {
          const response = await fetch('/api/auth/login', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ email, password })
          });

          if (!response.ok) {
            throw new Error('Login failed');
          }

          const data = await response.json();

          set({
            user: data.user,
            token: data.token,
            loading: false,
            error: null
          });

          return { success: true };
        } catch (error) {
          set({
            loading: false,
            error: error.message
          });
          return { success: false, error: error.message };
        }
      },

      logout: () => {
        set({
          user: null,
          token: null,
          error: null
        });
      },

      clearError: () => set({ error: null })
    }),
    {
      name: 'auth-storage',
      partialize: (state) => ({
        token: state.token
      })
    }
  )
);

export default useAuthStore;
```

10.4.5 Middleware in Zustand

Listing 10.23: Uso di middleware: persist e devtools

```
import { create } from 'zustand';
import { persist, devtools } from 'zustand/middleware';

const useStore = create(
  devtools(
    persist(
      (set, get) => ({
        bears: 0,
        increasePopulation: () =>
          set((state) => ({ bears: state.bears + 1 }))
      }),
      {
        name: 'animals-storage',
      }
    ),
    { name: 'AnimalStore' }
  )
);
```

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

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

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => {
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        return response.json();
      })
      .then(data => {
        setUsers(data);
        setLoading(false);
      })
      .catch(error => {
        setError(error.message);
        setLoading(false);
      });
  }, []);

  if (loading) return <div>Caricamento...</div>;
  if (error) return <div>Errore: {error}</div>;

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

11.2.2 Async/Await con Fetch

Listing 11.2: Fetch con async/await

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

function Posts() {
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchPosts = async () => {
      try {
        setLoading(true);

        const response = await fetch(
          'https://jsonplaceholder.typicode.com/posts'
        );

        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }

        const data = await response.json();
        setPosts(data);
      } catch (error) {
        setError(error.message);
      } finally {
        setLoading(false);
      }
    };

    fetchPosts();
  }, []);

  if (loading) return <div>Caricamento...</div>;
  if (error) return <div>Errore: {error}</div>;

  return (
    <div>
```

```
      {posts.map(post => (
        <article key={post.id}>
          <h3>{post.title}</h3>
          <p>{post.body}</p>
        </article>
      ))}
    </div>
  );
}
```

11.2.3 POST Request

Listing 11.3: Fetch POST per creare risorsa

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

function CreatePost() {
  const [title, setTitle] = useState('');
  const [body, setBody] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const [success, setSuccess] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      setLoading(true);
      setError(null);
      setSuccess(false);

      const response = await fetch(
        'https://jsonplaceholder.typicode.com/posts',
        {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({
            title,
            body,
            userId: 1
          })
        }
      );

      if (!response.ok) {
        throw new Error('Failed to create post');
      }

      const data = await response.json();
      console.log('Created post:', data);

      setSuccess(true);
      setTitle('');
      setBody('');
    } catch (error) {
      setError(error.message);
    } finally {
      setLoading(false);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>Crea Post</h2>

      {error && <div className="error">{error}</div>}
      {success && <div className="success">Post creato con successo!</
         div>}

      <input
        type="text"
        value={title}
        onChange={(e) => setTitle(e.target.value)}
        placeholder="Titolo"
        required
      />

      <textarea
        value={body}
        onChange={(e) => setBody(e.target.value)}
        placeholder="Contenuto"
        required
      />

      <button type="submit" disabled={loading}>
        {loading ? 'Creazione...' : 'Crea Post'}
      </button>
    </form>
  );
}
```

11.2.4 PUT e DELETE Requests

Listing 11.4: Update e Delete con Fetch

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

function PostManager({ postId }) {
  const [loading, setLoading] = useState(false);

  // UPDATE
  const updatePost = async (updates) => {
    try {
      setLoading(true);

      const response = await fetch(
        `https://jsonplaceholder.typicode.com/posts/${postId}`,
        {
          method: 'PUT',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(updates)
        }
      );

      if (!response.ok) {
        throw new Error('Failed to update post');
      }

      const data = await response.json();
      console.log('Updated post:', data);
    } catch (error) {
      console.error('Update error:', error);
    } finally {
      setLoading(false);
    }
  };

  // PATCH (partial update)
  const patchPost = async (updates) => {
    try {
      setLoading(true);

      const response = await fetch(
        `https://jsonplaceholder.typicode.com/posts/${postId}`,
        {
          method: 'PATCH',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(updates)
        }
      );

      const data = await response.json();
      console.log('Patched post:', data);
    } catch (error) {
      console.error('Patch error:', error);
    } finally {
      setLoading(false);
    }
  };

  // DELETE
  const deletePost = async () => {
    if (!confirm('Sei sicuro di voler eliminare questo post?')) {
      return;
    }

    try {
      setLoading(true);

      const response = await fetch(
        `https://jsonplaceholder.typicode.com/posts/${postId}`,
        {
          method: 'DELETE'
        }
      );

      if (!response.ok) {
        throw new Error('Failed to delete post');
      }

      console.log('Post deleted');
    } catch (error) {
      console.error('Delete error:', error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div>
      <button onClick={() => updatePost({ title: 'Nuovo titolo' })}>
        Update
      </button>
      <button onClick={() => patchPost({ title: 'Titolo modificato' })}>
        Patch
      </button>
      <button onClick={deletePost} disabled={loading}>
        Delete
      </button>
    </div>
  );
}
```

11.2.5 Gestione Autenticazione con Headers

Listing 11.5: Richieste autenticate con token

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

function ProtectedData() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchProtectedData = async () => {
      try {
        // Recupera token dal localStorage
        const token = localStorage.getItem('authToken');

        if (!token) {
          throw new Error('No authentication token found');
        }

        const response = await fetch('https://api.example.com/protected'
           , {
          headers: {
            'Authorization': `Bearer ${token}`,
            'Content-Type': 'application/json'
          }
        });

        if (response.status === 401) {
          // Token scaduto o invalido
          throw new Error('Unauthorized - Please login again');
        }

        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }

        const data = await response.json();
        setData(data);
      } catch (error) {
        setError(error.message);
      } finally {
        setLoading(false);
      }
    };

    fetchProtectedData();
  }, []);

  if (loading) return <div>Caricamento...</div>;
  if (error) return <div>Errore: {error}</div>;

  return <div>{JSON.stringify(data, null, 2)}</div>;
}
```

11.2.6 AbortController per Cancellazione

Listing 11.6: Cancellazione richieste con AbortController

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

function SearchResults({ searchTerm }) {
  const [results, setResults] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  useEffect(() => {
    if (!searchTerm) {
      setResults([]);
      return;
    }

    // Crea AbortController
    const controller = new AbortController();
    const signal = controller.signal;

    const fetchResults = async () => {
      try {
        setLoading(true);
        setError(null);

        const response = await fetch(
          `https://api.example.com/search?q=${searchTerm}`,
          { signal } // Passa signal alla fetch
        );

        if (!response.ok) {
          throw new Error('Search failed');
        }

        const data = await response.json();

        // Controlla se non abortito
        if (!signal.aborted) {
          setResults(data);
        }
      } catch (error) {
        // Ignora errori di abort
        if (error.name !== 'AbortError') {
          setError(error.message);
        }
      } finally {
        if (!signal.aborted) {
          setLoading(false);
        }
      }
    };

    fetchResults();

    // Cleanup: abort fetch quando searchTerm cambia
    return () => {
      controller.abort();
    };
  }, [searchTerm]);

  if (loading) return <div>Ricerca in corso...</div>;
  if (error) return <div>Errore: {error}</div>;

  return (
    <ul>
      {results.map(result => (
        <li key={result.id}>{result.name}</li>
      ))}
    </ul>
  );
}
```

11.3 Custom Hook useFetch

Creare un custom hook riutilizzabile per fetch.

Listing 11.7: Custom hook useFetch

```
// hooks/useFetch.js
import { useState, useEffect } from 'react';

function useFetch(url, options = {}) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const controller = new AbortController();

    const fetchData = async () => {
      try {
        setLoading(true);
        setError(null);

        const response = await fetch(url, {
          ...options,
          signal: controller.signal
        });

        if (!response.ok) {
          throw new Error(`HTTP error! status: ${response.status}`);
        }

        const json = await response.json();
        setData(json);
      } catch (error) {
        if (error.name !== 'AbortError') {
          setError(error.message);
        }
      } finally {
        if (!controller.signal.aborted) {
          setLoading(false);
        }
      }
    };

    fetchData();

    return () => {
      controller.abort();
    };
  }, [url]); // Dipende solo da url

  return { data, loading, error };
}

export default useFetch;

// Utilizzo
function Users() {
  const { data: users, loading, error } = useFetch(
    'https://jsonplaceholder.typicode.com/users'
  );

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

11.4 Axios

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

11.4.1 Installazione

Listing 11.8: Installazione Axios

```
npm install axios

# o con yarn
yarn add axios
```

11.4.2 GET Request con Axios

Listing 11.9: Axios GET request

```
import { useState, useEffect } from 'react';
import axios from 'axios';

function Users() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        setLoading(true);

        // Axios gestisce automaticamente JSON
        const response = await axios.get(
          'https://jsonplaceholder.typicode.com/users'
        );

        // I dati sono direttamente in response.data
        setUsers(response.data);
      } catch (error) {
        // Axios fornisce error.response per errori HTTP
        if (error.response) {
          setError(`Error: ${error.response.status}`);
        } else if (error.request) {
          setError('No response from server');
        } else {
          setError(error.message);
        }
      } finally {
        setLoading(false);
      }
    };

    fetchUsers();
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>{error}</div>;

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

11.4.3 POST, PUT, DELETE con Axios

Listing 11.10: CRUD operations con Axios

```
import axios from 'axios';

const API_URL = 'https://jsonplaceholder.typicode.com/posts';

// CREATE
async function createPost(postData) {
  try {
    const response = await axios.post(API_URL, postData);
    return { success: true, data: response.data };
  } catch (error) {
    return { success: false, error: error.message };
  }
}

// READ
async function getPost(id) {
  try {
    const response = await axios.get(`${API_URL}/${id}`);
    return { success: true, data: response.data };
  } catch (error) {
    return { success: false, error: error.message };
  }
}

// UPDATE (full)
async function updatePost(id, postData) {
  try {
    const response = await axios.put(`${API_URL}/${id}`, postData);
    return { success: true, data: response.data };
  } catch (error) {
    return { success: false, error: error.message };
  }
}

// UPDATE (partial)
async function patchPost(id, updates) {
  try {
    const response = await axios.patch(`${API_URL}/${id}`, updates);
    return { success: true, data: response.data };
  } catch (error) {
    return { success: false, error: error.message };
  }
}

```

```
// DELETE
async function deletePost(id) {
  try {
    await axios.delete(`${API_URL}/${id}`);
    return { success: true };
  } catch (error) {
    return { success: false, error: error.message };
  }
}

// Utilizzo in componente
function PostEditor({ postId }) {
  const handleUpdate = async () => {
    const result = await updatePost(postId, {
      title: 'Titolo aggiornato',
      body: 'Corpo aggiornato'
    });

    if (result.success) {
      console.log('Post aggiornato:', result.data);
    } else {
      console.error('Errore:', result.error);
    }
  };

  const handleDelete = async () => {
    const result = await deletePost(postId);

    if (result.success) {
      console.log('Post eliminato');
    } else {
      console.error('Errore:', result.error);
    }
  };

  return (
    <div>
      <button onClick={handleUpdate}>Aggiorna</button>
      <button onClick={handleDelete}>Elimina</button>
    </div>
  );
}
```

11.4.4 Axios Instance e Interceptors

Listing 11.11: Configurazione Axios instance

```
// api/axios.js
import axios from 'axios';

// Crea instance personalizzata
const axiosInstance = axios.create({
  baseURL: 'https://api.example.com',
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json'
  }
```

```
});

// Request interceptor
axiosInstance.interceptors.request.use(
  (config) => {
    // Aggiungi token ad ogni richiesta
    const token = localStorage.getItem('authToken');

    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }

    console.log('Request:', config.method.toUpperCase(), config.url);
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

// Response interceptor
axiosInstance.interceptors.response.use(
  (response) => {
    console.log('Response:', response.status, response.config.url);
    return response;
  },
  (error) => {
    // Gestione errori globale
    if (error.response) {
      switch (error.response.status) {
        case 401:
          // Unauthorized - redirect al login
          localStorage.removeItem('authToken');
          window.location.href = '/login';
          break;

        case 403:
          // Forbidden
          console.error('Access forbidden');
          break;

        case 404:
          console.error('Resource not found');
          break;

        case 500:
          console.error('Server error');
          break;

        default:
          console.error('Request error:', error.response.status);
      }
    } else if (error.request) {
      console.error('No response from server');
    } else {
      console.error('Error:', error.message);
    }

    return Promise.reject(error);
  }
);

export default axiosInstance;

// Utilizzo
import api from './api/axios';

function MyComponent() {
  const fetchData = async () => {
    try {
      // L'interceptor aggiunge automaticamente il token
      const response = await api.get('/protected-route');
      console.log(response.data);
    } catch (error) {
      // L'error interceptor ha già loggato l'errore
      console.error('Failed to fetch data');
    }
  };

  return <button onClick={fetchData}>Fetch Data</button>;
}
```

11.4.5 Cancellazione con Axios

Listing 11.12: Cancel token in Axios

```
import { useState, useEffect } from 'react';
import axios from 'axios';

function SearchComponent({ query }) {
  const [results, setResults] = useState([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    // Crea cancel token
    const source = axios.CancelToken.source();

    const fetchResults = async () => {
      try {
        setLoading(true);

        const response = await axios.get(
          `https://api.example.com/search?q=${query}`,
          {
            cancelToken: source.token
          }
        );

        setResults(response.data);
      } catch (error) {
        if (axios.isCancel(error)) {
          console.log('Request canceled:', error.message);
        } else {
          console.error('Error:', error);
        }
      } finally {
        setLoading(false);
      }
    };

    if (query) {
      fetchResults();
    }

    // Cleanup: cancella richiesta
    return () => {
      source.cancel('Operation canceled by user');
    };
  }, [query]);

  return (
    <div>
      {loading && <div>Searching...</div>}
      {results.map(result => (
        <div key={result.id}>{result.name}</div>
      ))}
    </div>
  );
}
```

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

```
npm install @tanstack/react-query

# o con yarn
yarn add @tanstack/react-query
```

11.5.2 Setup Base

Listing 11.14: QueryClient setup

```
// main.jsx
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
    ;
import { ReactQueryDevtools } from '@tanstack/react-query-devtools';

// Crea query client
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 1000 * 60 * 5, // 5 minuti
      cacheTime: 1000 * 60 * 10, // 10 minuti
      refetchOnWindowFocus: false,
      retry: 1
    }
  }
});

ReactDOM.createRoot(document.getElementById('root')).render(
  <QueryClientProvider client={queryClient}>
    <App />
    <ReactQueryDevtools initialIsOpen={false} />
  </QueryClientProvider>
);
```

11.5.3 useQuery Hook

Listing 11.15: Fetch data con useQuery

```
import { useQuery } from '@tanstack/react-query';
import axios from 'axios';

// Funzione di fetch
const fetchUsers = async () => {
  const { data } = await axios.get(
    'https://jsonplaceholder.typicode.com/users'
  );
  return data;
};

function Users() {
  const {
    data: users,
    isLoading,
    isError,
    error,
    refetch,
    isFetching
  } = useQuery({
    queryKey: ['users'],
    queryFn: fetchUsers
  });

  if (isLoading) return <div>Caricamento iniziale...</div>;
  if (isError) return <div>Errore: {error.message}</div>;

  return (
    <div>
      <button onClick={() => refetch()}>
        {isFetching ? 'Aggiornamento...' : 'Aggiorna'}
      </button>

      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

11.5.4 Query con Parametri

Listing 11.16: useQuery con parametri dinamici

```
import { useQuery } from '@tanstack/react-query';
import axios from 'axios';

const fetchUser = async (userId) => {
  const { data } = await axios.get(
    `https://jsonplaceholder.typicode.com/users/${userId}`
  );
  return data;
};

function UserProfile({ userId }) {
  const {
    data: user,
    isLoading,
    isError,
    error
  } = useQuery({
    queryKey: ['user', userId], // Include userId nella key
    queryFn: () => fetchUser(userId),
    enabled: !!userId // Esegue solo se userId esiste
  });

  if (!userId) return <div>Seleziona un utente</div>;
  if (isLoading) return <div>Caricamento...</div>;
  if (isError) return <div>Errore: {error.message}</div>;

  return (
    <div>
      <h2>{user.name}</h2>
      <p>Email: {user.email}</p>
      <p>Website: {user.website}</p>
    </div>
  );
}
```

11.5.5 useMutation Hook

Listing 11.17: Mutations per CREATE, UPDATE, DELETE

```
import { useMutation, useQueryClient } from '@tanstack/react-query';
import axios from 'axios';

const createPost = async (newPost) => {
  const { data } = await axios.post(
    'https://jsonplaceholder.typicode.com/posts',
    newPost
  );
  return data;
};

function CreatePost() {
  const queryClient = useQueryClient();

  const mutation = useMutation({
    mutationFn: createPost,
    onSuccess: (data) => {
      // Invalida e refetch
      queryClient.invalidateQueries({ queryKey: ['posts'] });

      // Oppure aggiungi manualmente alla cache
      // queryClient.setQueryData(['posts'], (old) => [...old, data]);

      console.log('Post creato:', data);
    },
    onError: (error) => {
      console.error('Errore creazione:', error);
    }
  });

  const handleSubmit = (e) => {
    e.preventDefault();

    mutation.mutate({
      title: 'Nuovo Post',
      body: 'Contenuto del post',
      userId: 1
    });
  };

  return (
    <form onSubmit={handleSubmit}>
      <button type="submit" disabled={mutation.isPending}>
        {mutation.isPending ? 'Creazione...' : 'Crea Post'}
      </button>

      {mutation.isError && (
        <div>Errore: {mutation.error.message}</div>
      )}

      {mutation.isSuccess && (
        <div>Post creato con successo!</div>
      )}
    </form>
  );
}
```

11.5.6 Esempio Completo: CRUD con React Query

Listing 11.18: CRUD completo con React Query

```
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-
    query';
import axios from 'axios';
import { useState } from 'react';

const API_URL = 'https://jsonplaceholder.typicode.com/posts';

// API functions
const fetchPosts = async () => {
  const { data } = await axios.get(API_URL);
  return data;
```

```
};

const createPost = async (newPost) => {
  const { data } = await axios.post(API_URL, newPost);
  return data;
};

const updatePost = async ({ id, updates }) => {
  const { data } = await axios.put(`${API_URL}/${id}`, updates);
  return data;
};

const deletePost = async (id) => {
  await axios.delete(`${API_URL}/${id}`);
  return id;
};

function PostsManager() {
  const queryClient = useQueryClient();
  const [editingId, setEditingId] = useState(null);

  // GET - Fetch all posts
  const {
    data: posts,
    isLoading,
    isError,
    error
  } = useQuery({
    queryKey: ['posts'],
    queryFn: fetchPosts
  });

  // CREATE
  const createMutation = useMutation({
    mutationFn: createPost,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['posts'] });
    }
  });

  // UPDATE
  const updateMutation = useMutation({
    mutationFn: updatePost,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['posts'] });
      setEditingId(null);
    }
  });

  // DELETE
  const deleteMutation = useMutation({
    mutationFn: deletePost,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['posts'] });
    }
  });

  const handleCreate = () => {
    createMutation.mutate({
      title: 'Nuovo Post',
      body: 'Contenuto',
      userId: 1
    });
  };

  const handleUpdate = (id) => {
    updateMutation.mutate({
      id,
      updates: {
        title: 'Titolo Aggiornato',
        body: 'Corpo Aggiornato'
      }
    });
  };

  const handleDelete = (id) => {
    if (confirm('Eliminare questo post?')) {
      deleteMutation.mutate(id);
    }
  };

  if (isLoading) return <div>Caricamento...</div>;
  if (isError) return <div>Errore: {error.message}</div>;

  return (
    <div>
      <h1>Posts Manager</h1>

      <button onClick={handleCreate} disabled={createMutation.isPending
         }>
        {createMutation.isPending ? 'Creazione...' : 'Crea Nuovo Post'}
      </button>

      <div className="posts">
        {posts.map(post => (
          <div key={post.id} className="post">
            <h3>{post.title}</h3>
            <p>{post.body}</p>

            <button
              onClick={() => handleUpdate(post.id)}
              disabled={updateMutation.isPending}
            >
              Update
            </button>

            <button
              onClick={() => handleDelete(post.id)}
              disabled={deleteMutation.isPending}
            >
              Delete
            </button>
          </div>
        ))}
      </div>
    </div>
  );
}
```

11.5.7 Infinite Queries

Listing 11.19: Infinite scrolling con useInfiniteQuery

```
import { useInfiniteQuery } from '@tanstack/react-query';
import axios from 'axios';
import { useEffect, useRef } from 'react';

const fetchPosts = async ({ pageParam = 1 }) => {
  const { data } = await axios.get(
    `https://jsonplaceholder.typicode.com/posts?_page=${pageParam}&
       _limit=10`
  );
  return {
    posts: data,
    nextPage: pageParam + 1,
    hasMore: data.length > 0
  };
};

function InfinitePostList() {
  const {
    data,
    fetchNextPage,
    hasNextPage,
    isFetchingNextPage,
    isLoading,
    isError,
    error
  } = useInfiniteQuery({
    queryKey: ['posts', 'infinite'],
    queryFn: fetchPosts,
    getNextPageParam: (lastPage) =>
      lastPage.hasMore ? lastPage.nextPage : undefined,
    initialPageParam: 1
  });

  const observerTarget = useRef(null);

  useEffect(() => {
    const observer = new IntersectionObserver(
      entries => {
        if (entries[0].isIntersecting && hasNextPage) {
          fetchNextPage();
        }
      },
      { threshold: 1 }
    );

    if (observerTarget.current) {
      observer.observe(observerTarget.current);
    }

    return () => observer.disconnect();
  }, [hasNextPage, fetchNextPage]);

  if (isLoading) return <div>Caricamento...</div>;
  if (isError) return <div>Errore: {error.message}</div>;

  return (
    <div>
      <h1>Infinite Scroll Posts</h1>

      {data.pages.map((page, pageIndex) => (
        <div key={pageIndex}>
          {page.posts.map(post => (
            <article key={post.id} className="post">
              <h3>{post.title}</h3>
              <p>{post.body}</p>
            </article>
          ))}
        </div>
      ))}

      <div ref={observerTarget} className="observer">
        {isFetchingNextPage && <div>Caricamento altri post...</div>}
        {!hasNextPage && <div>Fine dei post</div>}
      </div>
    </div>
  );
}
```

11.5.8 Optimistic Updates

Listing 11.20: Aggiornamenti ottimistici

```
import { useMutation, useQueryClient } from '@tanstack/react-query';
import axios from 'axios';

function TodoList() {
  const queryClient = useQueryClient();

  const toggleTodoMutation = useMutation({
    mutationFn: async ({ id, completed }) => {
      const { data } = await axios.patch(
        `https://api.example.com/todos/${id}`,
        { completed: !completed }
      );
      return data;
    },

    // Optimistic update
    onMutate: async ({ id, completed }) => {
      // Cancella queries in uscita
      await queryClient.cancelQueries({ queryKey: ['todos'] });

      // Snapshot del valore precedente
      const previousTodos = queryClient.getQueryData(['todos']);

      // Aggiorna ottimisticamente
      queryClient.setQueryData(['todos'], (old) =>
        old.map(todo =>
          todo.id === id
            ? { ...todo, completed: !completed }
            : todo
        )
      );

      // Ritorna context con snapshot
      return { previousTodos };
    },

    // Se la mutation fallisce, rollback
    onError: (err, variables, context) => {
      queryClient.setQueryData(['todos'], context.previousTodos);
    },

    // Sempre refetch dopo error o success
    onSettled: () => {
      queryClient.invalidateQueries({ queryKey: ['todos'] });
    }
  });

  return (
    // JSX component
    <div>Todos</div>
  );
}
```

11.6 Error Handling Avanzato

11.6.1 Error Boundary per API Errors

Listing 11.21: Error boundary per gestione errori

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

class ApiErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false, error: null };
  }

  static getDerivedStateFromError(error) {
    return { hasError: true, error };
  }

  componentDidCatch(error, errorInfo) {
    console.error('API Error:', error, errorInfo);

    // Log to error reporting service
    // logErrorToService(error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="error-container">
```

```
          <h2>Qualcosa è andato storto</h2>
          <p>{this.state.error?.message}</p>
          <button onClick={() => this.setState({ hasError: false })}>
            Riprova
          </button>
        </div>
      );
    }

    return this.props.children;
  }
}

// Utilizzo
function App() {
  return (
    <ApiErrorBoundary>
      <DataFetchingComponent />
    </ApiErrorBoundary>
  );
}
```

11.6.2 Toast Notifications per Errori

Listing 11.22: Sistema di notifiche per errori API

```
import { createContext, useContext, useState } from ’react’;

const ToastContext = createContext(null);

export function ToastProvider({ children }) {
  const [toasts, setToasts] = useState([]);

  const addToast = (message, type = ’info’) => {
    const id = Date.now();
    setToasts(prev => [...prev, { id, message, type }]);

    // Auto-remove dopo 5 secondi
    setTimeout(() => {
      removeToast(id);
    }, 5000);
  };

  const removeToast = (id) => {
    setToasts(prev => prev.filter(toast => toast.id !== id));
  };

  return (
    <ToastContext.Provider value={{ addToast, removeToast }}>
      {children}

      <div className="toast-container">
        {toasts.map(toast => (
          <div key={toast.id} className={‘toast toast-${toast.type}‘}>
            {toast.message}
            <button onClick={() => removeToast(toast.id)}>  </button>
          </div>
        ))}
      </div>
    </ToastContext.Provider>
  );
}

export function useToast() {
  return useContext(ToastContext);
}

// Utilizzo con API
function MyComponent() {
  const { addToast } = useToast();

  const fetchData = async () => {
    try {
      const response = await fetch('https://api.example.com/data');

      if (!response.ok) {
        throw new Error('Failed to fetch data');
      }

      const data = await response.json();
      addToast('Dati caricati con successo!', 'success');
    } catch (error) {
      addToast(error.message, 'error');
    }
  };

  return <button onClick={fetchData}>Fetch Data</button>;
}
```

11.6.3 Retry Logic

Listing 11.23: Logica di retry personalizzata

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

async function fetchWithRetry(url, options = {}, maxRetries = 3) {
  let lastError;

  for (let i = 0; i <= maxRetries; i++) {
    try {
      const response = await fetch(url, options);

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }

      return await response.json();
    } catch (error) {
      lastError = error;

      // Non ritentare se è l'ultimo tentativo
      if (i === maxRetries) {
        break;
      }

      // Exponential backoff: attendi 1s, 2s, 4s, etc.
      const delay = Math.pow(2, i) * 1000;
      console.log(`Retry ${i + 1}/${maxRetries} after ${delay}ms`);

      await new Promise(resolve => setTimeout(resolve, delay));
    }
  }

  throw lastError;
}

// Utilizzo
function DataComponent() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const loadData = async () => {
    try {
      setLoading(true);
      setError(null);

      const result = await fetchWithRetry(
        'https://api.example.com/data',
        {},
        3 // max 3 tentativi
      );

      setData(result);
    } catch (error) {
      setError(`Failed after retries: ${error.message}`);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div>
      <button onClick={loadData} disabled={loading}>
        {loading ? 'Loading...' : 'Load Data'}
      </button>
      {error && <div className="error">{error}</div>}
      {data && <pre>{JSON.stringify(data, null, 2)}</pre>}
    </div>
  );
}
```

11.7 Loading States Avanzati

11.7.1 Skeleton Screens

Listing 11.24: Skeleton loading component

```
function Skeleton({ width, height, className = '' }) {
  return (
    <div
```

```
      className={`skeleton ${className}`}
      style={{
        width,
        height,
        background: 'linear-gradient(90deg, #f0f0f0 25%, #e0e0e0 50%, #
            f0f0f0 75%)',
        backgroundSize: '200% 100%',
        animation: 'loading 1.5s ease-in-out infinite',
        borderRadius: '4px'
      }}
    />
  );
}

// CSS
// @keyframes loading {
//   0% {
//     background-position: 200% 0;
//   }
//   100% {
//     background-position: -200% 0;
//   }
// }

function UserCardSkeleton() {
  return (
    <div className="user-card">
      <Skeleton width={64} height={64} className="avatar" />
      <div>
        <Skeleton width={200} height={20} />
        <Skeleton width={150} height={16} />
      </div>
    </div>
  );
}

// Utilizzo
function UserList() {
  const { data: users, isLoading } = useQuery({
    queryKey: ['users'],
    queryFn: fetchUsers
  });

  if (isLoading) {
    return (
      <div>
        {[...Array(5)].map((_, i) => (
          <UserCardSkeleton key={i} />
        ))}
      </div>
    );
  }

  return (
    <div>
      {users.map(user => (
        <UserCard key={user.id} user={user} />
      ))}
    </div>
  );
}
```

11.7.2 Progress Indicator

Listing 11.25: Progress bar per upload

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

function FileUpload() {
  const [file, setFile] = useState(null);
  const [progress, setProgress] = useState(0);
  const [uploading, setUploading] = useState(false);

  const handleUpload = async () => {
    if (!file) return;

    const formData = new FormData();
    formData.append('file', file);

    try {
      setUploading(true);

      await axios.post('https://api.example.com/upload', formData, {
        headers: {
          'Content-Type': 'multipart/form-data'
        },
        onUploadProgress: (progressEvent) => {
          const percentCompleted = Math.round(
            (progressEvent.loaded * 100) / progressEvent.total
          );
          setProgress(percentCompleted);
        }
      });

      console.log('Upload completato!');
    } catch (error) {
      console.error('Upload fallito:', error);
    } finally {
      setUploading(false);
      setProgress(0);
    }
  };

  return (
    <div>
      <input
        type="file"
        onChange={(e) => setFile(e.target.files[0])}
      />

      <button onClick={handleUpload} disabled={!file || uploading}>
        {uploading ? 'Uploading...' : 'Upload'}
      </button>

      {uploading && (
        <div className="progress-bar">
          <div
            className="progress-fill"
            style={{ width: `${progress}%` }}
          >
            {progress}%
          </div>
        </div>
      )}
    </div>
  );
}
```

11.8 Caching e Revalidation

11.8.1 React Query Cache Config

Listing 11.26: Configurazione cache avanzata

```
import { QueryClient } from '@tanstack/react-query';

const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      // Dati considerati fresh per 5 minuti
      staleTime: 1000 * 60 * 5,

      // Dati mantenuti in cache per 10 minuti
      cacheTime: 1000 * 60 * 10,

      // Refetch quando la finestra riceve focus
      refetchOnWindowFocus: true,

      // Refetch quando si riconnette
      refetchOnReconnect: true,

      // Refetch on mount se dati stale
      refetchOnMount: true,

      // Retry fallite requests
      retry: 2,

      // Retry delay (exponential backoff)
      retryDelay: (attemptIndex) =>
        Math.min(1000 * 2 ** attemptIndex, 30000)
    },
    mutations: {
      // Retry mutations
      retry: 1
    }
  }
});
```

11.8.2 Manual Cache Updates

Listing 11.27: Aggiornamento manuale della cache

```
import { useQueryClient } from '@tanstack/react-query';

function UserProfile({ userId }) {
  const queryClient = useQueryClient();

  const updateUserName = (newName) => {
    // Update cache direttamente
    queryClient.setQueryData(['user', userId], (oldData) => ({
      ...oldData,
      name: newName
    }));
  };

  const invalidateUser = () => {
    // Invalida cache (forza refetch)
    queryClient.invalidateQueries({ queryKey: ['user', userId] });
  };

  const prefetchUser = async (nextUserId) => {
    // Prefetch dati per navigazione anticipata
    await queryClient.prefetchQuery({
      queryKey: ['user', nextUserId],
      queryFn: () => fetchUser(nextUserId)
    });
  };

  return <div>User Profile</div>;
}
```

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

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

// Componente senza memo (ri-renderizza sempre)
function ExpensiveComponent({ data }) {
  console.log('Rendering ExpensiveComponent');

  // Calcoli costosi...
  const processedData = data.map(item => {
    // elaborazione complessa
    return item * 2;
  });

  return (
    <div>
      {processedData.map((item, i) => (
        <div key={i}>{item}</div>
      ))}
    </div>
  );
}

// Componente con memo (ri-renderizza solo se props cambiano)
const MemoizedExpensiveComponent = memo(ExpensiveComponent);

// Utilizzo
function Parent() {
  const [count, setCount] = useState(0);
  const [data, setData] = useState([1, 2, 3, 4, 5]);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Count: {count}</button
        >

      {/* Questo NON ri-renderizza quando count cambia */}
      <MemoizedExpensiveComponent data={data} />
    </div>
  );
}
```

12.2.2 Custom Comparison Function

Listing 12.2: React.memo con comparatore personalizzato

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

function UserCard({ user, onEdit }) {
  console.log('Rendering UserCard for', user.name);

  return (
    <div className="user-card">
      <h3>{user.name}</h3>
      <p>{user.email}</p>
      <button onClick={() => onEdit(user.id)}>Edit</button>
    </div>
  );
}

// Custom comparison: ri-renderizza solo se user.id o user.name cambiano
const MemoizedUserCard = memo(UserCard, (prevProps, nextProps) => {
  // Ritorna true se props sono uguali (NON re-render)
  // Ritorna false se props sono diverse (re-render)
  return (
    prevProps.user.id === nextProps.user.id &&
    prevProps.user.name === nextProps.user.name
  );
});

export default MemoizedUserCard;
```

12.2.3 Quando NON usare React.memo

Listing 12.3: Casi in cui React.memo non aiuta

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

// SBAGLIATO: Component che cambia sempre
const BadMemo = memo(({ timestamp }) => {
  // timestamp cambia sempre, memo inutile
  return <div>Time: {timestamp}</div>;
});

// SBAGLIATO: Props non primitive sempre nuove
function Parent() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Increment</button>

      {/* Oggetto creato ogni render, memo inutile */}
      <MemoizedChild config={{ theme: 'dark' }} />

      {/* Funzione creata ogni render, memo inutile */}
      <MemoizedChild onClick={() => console.log('click')} />
    </div>
  );
}

// CORRETTO: Usa useCallback e useMemo
function ParentFixed() {
  const [count, setCount] = useState(0);

  const config = useMemo(() => ({ theme: 'dark' }), []);
  const handleClick = useCallback(() => console.log('click'), []);

  return (
    <div>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <MemoizedChild config={config} onClick={handleClick} />
    </div>
  );
}
```

12.3 useMemo Hook

`useMemo` memoizza il risultato di un calcolo costoso.

12.3.1 Utilizzo Base

Listing 12.4: useMemo per calcoli costosi

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

function ProductList({ products }) {
  const [filter, setFilter] = useState('');

  // Calcolo costoso memoizzato
  const filteredProducts = useMemo(() => {
    console.log('Filtering products...');

    return products.filter(product =>
      product.name.toLowerCase().includes(filter.toLowerCase())
    );
  }, [products, filter]); // Ri-calcola solo se products o filter cambiano

  // Calcolo costoso: somma prezzi
  const totalPrice = useMemo(() => {
    console.log('Calculating total price...');

    return filteredProducts.reduce((sum, p) => sum + p.price, 0);
  }, [filteredProducts]);

  return (
    <div>
      <input
        value={filter}
        onChange={(e) => setFilter(e.target.value)}
        placeholder="Filtra prodotti..."
      />

      <p>Totale: {totalPrice.toFixed(2)}   </p>

      <ul>
        {filteredProducts.map(product => (
          <li key={product.id}>
            {product.name} - {product.price}
          </li>
        ))}
      </ul>
    </div>
  );
}
```

12.3.2 Esempi Pratici

Listing 12.5: useMemo per vari scenari

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

function DataAnalysis({ data }) {
  // 1. Calcolo statistiche
  const stats = useMemo(() => {
    const sum = data.reduce((acc, val) => acc + val, 0);
    const avg = sum / data.length;
    const max = Math.max(...data);
    const min = Math.min(...data);

    return { sum, avg, max, min };
  }, [data]);

  // 2. Ordinamento
  const sortedData = useMemo(() => {
    return [...data].sort((a, b) => a - b);
  }, [data]);

  // 3. Trasformazione complessa
  const processedData = useMemo(() => {
    return data.map(item => ({
      value: item,
      squared: item * item,
      isEven: item % 2 === 0
    }));
  }, [data]);

  // 4. Creazione oggetto config
  const chartConfig = useMemo(() => ({
    type: 'line',
    data: sortedData,
    options: {
      responsive: true,
      plugins: {
        legend: {
          position: 'top'
        }
      }
    }
  }), [sortedData]);

  return (
    <div>
      <h3>Statistiche</h3>
      <p>Media: {stats.avg.toFixed(2)}</p>
      <p>Max: {stats.max}</p>
      <p>Min: {stats.min}</p>
    </div>
  );
}
```

12.3.3 Quando NON usare useMemo

Listing 12.6: useMemo inutile o dannoso

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

function Component() {
  // SBAGLIATO: Calcolo semplice, overhead inutile
  const doubleValue = useMemo(() => value * 2, [value]);

  // CORRETTO: Calcolo diretto
  const doubleValue = value * 2;

  // SBAGLIATO: String interpolation
  const greeting = useMemo(() => `Hello, ${name}`, [name]);

  // CORRETTO: Calcolo diretto
  const greeting = `Hello, ${name}`;

  // SBAGLIATO: Array.map semplice
  const doubled = useMemo(() => numbers.map(n => n * 2), [numbers]);

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

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

```
import { useState, useCallback, memo } from 'react';

// Child component memoizzato
const Button = memo(({ onClick, children }) => {
  console.log('Rendering Button:', children);
  return <button onClick={onClick}>{children}</button>;
});

function Parent() {
  const [count, setCount] = useState(0);
  const [otherState, setOtherState] = useState(0);

  // SBAGLIATO: Funzione ricreata ogni render
  const handleClick = () => {
    setCount(count + 1);
  };

  // CORRETTO: Funzione memoizzata
  const handleClickMemo = useCallback(() => {
    setCount(prev => prev + 1);
  }, []); // Dipendenze vuote: funzione mai ricreata

  return (
    <div>
      <p>Count: {count}</p>
      <p>Other: {otherState}</p>

      {/* Questo ri-renderizza sempre */}
      <Button onClick={handleClick}>Increment</Button>

      {/* Questo NON ri-renderizza quando otherState cambia */}
      <Button onClick={handleClickMemo}>Increment Memo</Button>

      <button onClick={() => setOtherState(otherState + 1)}>
        Update Other
      </button>
    </div>
  );
}
```

12.4.2 useCallback con Dipendenze

Listing 12.8: useCallback con dipendenze

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

function TodoList() {
  const [todos, setTodos] = useState([]);
  const [filter, setFilter] = useState('all');

  // Callback con dipendenza da filter
  const handleAddTodo = useCallback((text) => {
    const newTodo = {
      id: Date.now(),
      text,
      completed: false,
      category: filter
    };

    setTodos(prev => [...prev, newTodo]);
  }, [filter]); // Ri-crea quando filter cambia

  // Callback senza dipendenze esterne
  const handleToggleTodo = useCallback((id) => {
    setTodos(prev =>
      prev.map(todo =>
        todo.id === id
          ? { ...todo, completed: !todo.completed }
          : todo
      )
    );
  }, []); // Mai ri-creata

  // Callback con multiple dipendenze
  const handleDeleteCompleted = useCallback(() => {
    setTodos(prev => prev.filter(todo => !todo.completed));
    console.log('Deleted in category:', filter);
  }, [filter]); // Dipende da filter

  return (
    <div>
      <AddTodoForm onAdd={handleAddTodo} />
      <TodoItems todos={todos} onToggle={handleToggleTodo} />
      <button onClick={handleDeleteCompleted}>
        Delete Completed
      </button>
    </div>
  );
}
```

12.4.3 Pattern: Event Handlers

Listing 12.9: useCallback per event handlers

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

function UserProfile({ user, onUpdate }) {
  // Handler per form submit
  const handleSubmit = useCallback((e) => {
    e.preventDefault();
    const formData = new FormData(e.target);
    const updates = Object.fromEntries(formData);
    onUpdate(user.id, updates);
  }, [user.id, onUpdate]);

  // Handler per campo specifico
  const handleNameChange = useCallback((e) => {
    onUpdate(user.id, { name: e.target.value });
  }, [user.id, onUpdate]);

  // Handler per click con parametro
  const handleDelete = useCallback(() => {
    if (confirm(`Delete ${user.name}?`)) {
      onUpdate(user.id, { deleted: true });
    }
  }, [user.id, user.name, onUpdate]);

  return (
    <form onSubmit={handleSubmit}>
      <input
        name="name"
        defaultValue={user.name}
        onChange={handleNameChange}
      />
      <button type="submit">Save</button>
      <button type="button" onClick={handleDelete}>Delete</button>
    </form>
  );
}
```

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

```
import { lazy, Suspense } from 'react';

// Lazy import
const Dashboard = lazy(() => import('./pages/Dashboard'));
const Profile = lazy(() => import('./pages/Profile'));
const Settings = lazy(() => import('./pages/Settings'));

// Loading component
function LoadingSpinner() {
  return (
    <div className="loading">
      <div className="spinner"></div>
      <p>Caricamento...</p>
    </div>
  );
}

function App() {
  return (
    <Suspense fallback={<LoadingSpinner />}>
      <Routes>
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/settings" element={<Settings />} />
      </Routes>
    </Suspense>
  );
}
```

12.5.2 Route-based Code Splitting

Listing 12.11: Code splitting per route

```
import { lazy, Suspense } from 'react';
import { Routes, Route } from 'react-router-dom';

// Lazy load route components
const Home = lazy(() => import('./routes/Home'));
const About = lazy(() => import('./routes/About'));
const Products = lazy(() => import('./routes/Products'));
const ProductDetail = lazy(() => import('./routes/ProductDetail'));
const Admin = lazy(() => import('./routes/Admin'));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/products" element={<Products />} />
        <Route path="/products/:id" element={<ProductDetail />} />
        <Route path="/admin/*" element={<Admin />} />
      </Routes>
    </Suspense>
  );
}
```

12.5.3 Component-based Splitting

Listing 12.12: Lazy loading di componenti pesanti

```
import { lazy, Suspense, useState } from 'react';

// Componenti pesanti caricati solo quando necessario
const ChartComponent = lazy(() => import('./components/Chart'));
const DataTable = lazy(() => import('./components/DataTable'));
const VideoPlayer = lazy(() => import('./components/VideoPlayer'));

function Dashboard() {
  const [showChart, setShowChart] = useState(false);
  const [showTable, setShowTable] = useState(false);

  return (
    <div>
```

```
      <h1>Dashboard</h1>

      <button onClick={() => setShowChart(true)}>Show Chart</button>
      <button onClick={() => setShowTable(true)}>Show Table</button>

      {showChart && (
        <Suspense fallback={<div>Loading chart...</div>}>
          <ChartComponent data={chartData} />
        </Suspense>
      )}

      {showTable && (
        <Suspense fallback={<div>Loading table...</div>}>
          <DataTable data={tableData} />
        </Suspense>
      )}
    </div>
  );
}
```

12.5.4 Preloading

Listing 12.13: Preload componenti al hover

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

const HeavyComponent = lazy(() => import('./HeavyComponent'));

// Preload function
const preloadHeavyComponent = () => {
  import('./HeavyComponent');
};

function Navigation() {
  return (
    <nav>
      <Link
        to="/heavy"
        onMouseEnter={preloadHeavyComponent}
        onFocus={preloadHeavyComponent}
      >
        Heavy Page
      </Link>
    </nav>
  );
}
```

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

```
import { FixedSizeList } from 'react-window';

function VirtualizedList({ items }) {
  const Row = ({ index, style }) => (
    <div style={style} className="list-item">
      {items[index].name}
    </div>
  );

  return (
    <FixedSizeList
      height={600}
      itemCount={items.length}
      itemSize={50}
      width="100%"
    >
      {Row}
    </FixedSizeList>
  );
}

// Utilizzo
function App() {
  const items = Array.from({ length: 10000 }, (_, i) => ({
    id: i,
    name: `Item ${i}`
  }));

  return <VirtualizedList items={items} />;
}
```

12.6.2 Liste a Dimensioni Variabili

Listing 12.15: Liste con altezze variabili

```
import { VariableSizeList } from 'react-window';

function VariableList({ items }) {
  // Funzione per calcolare altezza
  const getItemSize = (index) => {
    const item = items[index];
    // Calcola in base al contenuto
    return item.content.length > 100 ? 100 : 50;
  };

  const Row = ({ index, style }) => (
    <div style={style} className="list-item">
      <h3>{items[index].title}</h3>
      <p>{items[index].content}</p>
    </div>
  );

  return (
    <VariableSizeList
      height={600}
      itemCount={items.length}
      itemSize={getItemSize}
      width="100%"
    >
      {Row}
    </VariableSizeList>
  );
}
```

12.7 React Profiler

Strumento per misurare performance e identificare bottleneck.

12.7.1 Utilizzo del Profiler

Listing 12.16: Profiler Component

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

function onRenderCallback(
  id, // id del Profiler
  phase, // "mount" o "update"
  actualDuration, // tempo speso per render
  baseDuration, // tempo stimato senza memoization
  startTime, // quando React ha iniziato
  commitTime, // quando React ha committato
  interactions // Set di interazioni
) {
  console.log(`${id} (${phase}):`, {
    actualDuration,
    baseDuration
  });

  // Log su servizio analytics
  // analytics.track('render', { id, phase, actualDuration });
}

function App() {
  return (
    <Profiler id="App" onRender={onRenderCallback}>
      <Dashboard />
    </Profiler>
  );
}

// Multiple profilers
function ComplexApp() {
  return (
    <div>
      <Profiler id="Navigation" onRender={onRenderCallback}>
        <Navigation />
      </Profiler>

      <Profiler id="Content" onRender={onRenderCallback}>
        <Content />
      </Profiler>

      <Profiler id="Sidebar" onRender={onRenderCallback}>
```

```
        <Sidebar />
      </Profiler>
    </div>
  );
}
```

12.7.2 Custom Performance Hook

Listing 12.17: Hook per misurare performance

```
import { useEffect, useRef } from 'react';

function useRenderTime(componentName) {
  const renderCount = useRef(0);
  const startTime = useRef(performance.now());

  useEffect(() => {
    renderCount.current += 1;
    const endTime = performance.now();
    const renderTime = endTime - startTime.current;

    console.log(
      `${componentName} render #${renderCount.current}: ${renderTime.
         toFixed(2)}ms`
    );

    startTime.current = performance.now();
  });
}

// Utilizzo
function ExpensiveComponent() {
  useRenderTime('ExpensiveComponent');

  // ... component logic
  return <div>Component</div>;
}
```

12.8 Ottimizzazioni Varie

12.8.1 Debounce e Throttle

Listing 12.18: Debounce per input

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

function useDebounce(value, delay) {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const timer = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => {
      clearTimeout(timer);
    };
  }, [value, delay]);

  return debouncedValue;
}

// Utilizzo
function SearchComponent() {
  const [searchTerm, setSearchTerm] = useState('');
  const debouncedSearch = useDebounce(searchTerm, 500);

  useEffect(() => {
    if (debouncedSearch) {
      // Esegui ricerca API
      console.log('Searching for:', debouncedSearch);
    }
  }, [debouncedSearch]);

  return (
    <input
      value={searchTerm}
      onChange={(e) => setSearchTerm(e.target.value)}
      placeholder="Search..."
    />
  );
}
```

12.8.2 Immagini Ottimizzate

Listing 12.19: Lazy loading immagini

```
import { useState, useEffect, useRef } from 'react';

function LazyImage({ src, alt, placeholder }) {
  const [imageSrc, setImageSrc] = useState(placeholder);
  const [imageRef, setImageRef] = useState();

  useEffect(() => {
    let observer;

    if (imageRef && imageSrc === placeholder) {
      observer = new IntersectionObserver(
        entries => {
          entries.forEach(entry => {
            if (entry.isIntersecting) {
              setImageSrc(src);
              observer.unobserve(imageRef);
            }
          });
        }
      );

      observer.observe(imageRef);
    }

    return () => {
      if (observer && imageRef) {
        observer.unobserve(imageRef);
      }
    };
  }, [imageRef, imageSrc, placeholder, src]);

  return (
    <img
      ref={setImageRef}
      src={imageSrc}
      alt={alt}
      className={imageSrc === placeholder ? 'loading' : 'loaded'}
    />
  );
}

// Utilizzo
function Gallery({ images }) {
  return (
    <div className="gallery">
      {images.map(img => (
        <LazyImage
          key={img.id}
          src={img.url}
          alt={img.title}
          placeholder="/placeholder.jpg"
        />
      ))}
    </div>
  );
}
```

12.8.3 Web Workers

Listing 12.20: Calcoli pesanti in Web Worker

```
// worker.js
self.addEventListener('message', (e) => {
  const { data } = e.data;

  // Calcolo pesante
  const result = data.map(item => {
    // Elaborazione complessa
    return item * item * Math.random();
  });

  self.postMessage({ result });
});

// Componente React
import { useState, useEffect } from 'react';

function HeavyCalculation({ data }) {
  const [result, setResult] = useState(null);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    const worker = new Worker(new URL('./worker.js', import.meta.url));

    worker.onmessage = (e) => {
      setResult(e.data.result);
      setLoading(false);
    };

    setLoading(true);
    worker.postMessage({ data });

    return () => {
      worker.terminate();
    };
  }, [data]);

  if (loading) return <div>Calculating...</div>;

  return (
    <div>
      Result: {result?.length} items processed
    </div>
  );
}
```

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

```
# Con Create React App, già configurato
# Altrimenti installa:

npm install --save-dev jest @testing-library/react @testing-library/jest
    -dom @testing-library/user-event

# o con yarn
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

```
// Button.jsx
function Button({ onClick, children }) {
  return <button onClick={onClick}>{children}</button>;
}

export default Button;
```

```

// Button.test.jsx
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import Button from './Button';

describe('Button Component', () => {
  test('renders button with text', () => {
    render(<Button>Click me</Button>);

    const buttonElement = screen.getByText(/click me/i);
    expect(buttonElement).toBeInTheDocument();
  });

  test('calls onClick when clicked', async () => {
    const handleClick = jest.fn();
    const user = userEvent.setup();

    render(<Button onClick={handleClick}>Click</Button>);

    const button = screen.getByText(/click/i);
    await user.click(button);

    expect(handleClick).toHaveBeenCalledTimes(1);
  });
});
```

13.2.2 Query Methods

Listing 13.3: Metodi per trovare elementi

```
import { render, screen } from '@testing-library/react';

function MyComponent() {
  return (
    <div>
      <h1>Hello World</h1>
      <button aria-label="submit">Submit</button>
      <input type="text" placeholder="Enter name" />
      <img src="/logo.png" alt="Logo" />
    </div>
  );
}

test('query methods', () => {
  render(<MyComponent />);

  // getBy* - Trova elemento, error se non trovato
  const heading = screen.getByText(/hello world/i);
  const button = screen.getByRole('button', { name: /submit/i });
  const input = screen.getByPlaceholderText(/enter name/i);
  const image = screen.getByAltText(/logo/i);

  // queryBy* - Trova elemento, null se non trovato
  const missing = screen.queryByText(/not here/i);
  expect(missing).not.toBeInTheDocument();

  // findBy* - Async, aspetta elemento (per caricamenti asincroni)
  const asyncElement = await screen.findByText(/loaded/i);

  // getAllBy*, queryAllBy*, findAllBy* - Multipli elementi
  const buttons = screen.getAllByRole('button');
  expect(buttons).toHaveLength(1);
});
```

13.2.3 Test di Componenti con State

Listing 13.4: Test componente con useState

```
// Counter.jsx
import { useState } from 'react';

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
}

export default Counter;

// Counter.test.jsx
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import Counter from './Counter';

describe('Counter', () => {
  test('renders initial count of 0', () => {
    render(<Counter />);
    expect(screen.getByText(/count: 0/i)).toBeInTheDocument();
  });

  test('increments count when increment button is clicked', async () =>
     {
    const user = userEvent.setup();
    render(<Counter />);

    const incrementBtn = screen.getByRole('button', { name: /increment/i
        });
    await user.click(incrementBtn);

    expect(screen.getByText(/count: 1/i)).toBeInTheDocument();
  });

  test('decrements count when decrement button is clicked', async () =>
     {
    const user = userEvent.setup();
    render(<Counter />);

    const decrementBtn = screen.getByRole('button', { name: /decrement/i
        });
    await user.click(decrementBtn);

    expect(screen.getByText(/count: -1/i)).toBeInTheDocument();
  });

  test('resets count to 0', async () => {
    const user = userEvent.setup();
    render(<Counter />);

    // Incrementa a 5
    const incrementBtn = screen.getByRole('button', { name: /increment/i
        });
    await user.click(incrementBtn);
    await user.click(incrementBtn);
    await user.click(incrementBtn);
    await user.click(incrementBtn);
    await user.click(incrementBtn);

    // Reset
    const resetBtn = screen.getByRole('button', { name: /reset/i });
    await user.click(resetBtn);

    expect(screen.getByText(/count: 0/i)).toBeInTheDocument();
  });
});
```

13.2.4 Test di Form

Listing 13.5: Test interazioni form

```
// LoginForm.jsx
import { useState } from 'react';

function LoginForm({ onSubmit }) {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();

    if (!email || !password) {
      setError('Email and password are required');
      return;
    }

    onSubmit({ email, password });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
        aria-label="Email"
      />

      <input
        type="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        placeholder="Password"
        aria-label="Password"
      />

      <button type="submit">Login</button>

      {error && <div role="alert">{error}</div>}
    </form>
  );
}

export default LoginForm;

// LoginForm.test.jsx
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import LoginForm from './LoginForm';

describe('LoginForm', () => {
  test('renders login form', () => {
    render(<LoginForm onSubmit={() => {}} />);

    expect(screen.getByLabelText(/email/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/password/i)).toBeInTheDocument();
    expect(screen.getByRole('button', { name: /login/i })).
        toBeInTheDocument();
  });

  test('submits form with email and password', async () => {
    const handleSubmit = jest.fn();
    const user = userEvent.setup();

    render(<LoginForm onSubmit={handleSubmit} />);

    // Compila form
    await user.type(screen.getByLabelText(/email/i), 'test@example.com')
        ;
    await user.type(screen.getByLabelText(/password/i), 'password123');

    // Submit
    await user.click(screen.getByRole('button', { name: /login/i }));

    // Verifica chiamata
    expect(handleSubmit).toHaveBeenCalledWith({
      email: 'test@example.com',
      password: 'password123'
    });
  });

  test('shows error when fields are empty', async () => {
    const user = userEvent.setup();
    render(<LoginForm onSubmit={() => {}} />);

    // Submit senza compilare
    await user.click(screen.getByRole('button', { name: /login/i }));

    // Verifica errore
    expect(screen.getByRole('alert')).toHaveTextContent(
      /email and password are required/i
    );
  });
});
```

13.3 Test di Componenti Asincroni

13.3.1 Mock di Fetch/API

Listing 13.6: Test con chiamate API mockate

```
// UserList.jsx
import { useState, useEffect } from 'react';

function UserList() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/users')
      .then(res => res.json())
      .then(data => {
        setUsers(data);
        setLoading(false);
      })
      .catch(err => {
        setError(err.message);
        setLoading(false);
      });
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default UserList;

// UserList.test.jsx
import { render, screen, waitFor } from '@testing-library/react';
import UserList from './UserList';

// Mock fetch globalmente
global.fetch = jest.fn();

describe('UserList', () => {
  beforeEach(() => {
    fetch.mockClear();
  });

  test('displays loading state initially', () => {
    fetch.mockImplementation(() => new Promise(() => {})); // Never
       resolves

    render(<UserList />);

    expect(screen.getByText(/loading/i)).toBeInTheDocument();
  });

  test('displays users after successful fetch', async () => {
    const mockUsers = [
      { id: 1, name: 'Alice' },
      { id: 2, name: 'Bob' }
    ];

    fetch.mockResolvedValueOnce({
      json: async () => mockUsers
    });

    render(<UserList />);

    // Aspetta che loading scompaia
    await waitFor(() => {
      expect(screen.queryByText(/loading/i)).not.toBeInTheDocument();
    });

    // Verifica utenti renderizzati
    expect(screen.getByText('Alice')).toBeInTheDocument();
    expect(screen.getByText('Bob')).toBeInTheDocument();
  });

  test('displays error on fetch failure', async () => {
    fetch.mockRejectedValueOnce(new Error('Network error'));

    render(<UserList />);

    await waitFor(() => {
      expect(screen.getByText(/error: network error/i)).
          toBeInTheDocument();
    });
  });
});
```

13.3.2 Mock Service Worker (MSW)

Listing 13.7: Setup MSW per mock API

```
// mocks/handlers.js
import { rest } from 'msw';

export const handlers = [
  rest.get('https://api.example.com/users', (req, res, ctx) => {
    return res(
      ctx.json([
        { id: 1, name: 'Alice' },
        { id: 2, name: 'Bob' }
      ])
    );
  }),

  rest.post('https://api.example.com/users', (req, res, ctx) => {
    const { name } = req.body;

    return res(
      ctx.json({
        id: 3,
        name
      })
    );
  })
];

// mocks/server.js
import { setupServer } from 'msw/node';
import { handlers } from './handlers';

export const server = setupServer(...handlers);

// setupTests.js
import '@testing-library/jest-dom';
import { server } from './mocks/server';

beforeAll(() => server.listen());
afterEach(() => server.resetHandlers());
afterAll(() => server.close());

// Test con MSW
import { render, screen, waitFor } from '@testing-library/react';
import UserList from './UserList';

test('fetches and displays users', async () => {
  render(<UserList />);

  await waitFor(() => {
    expect(screen.getByText('Alice')).toBeInTheDocument();
    expect(screen.getByText('Bob')).toBeInTheDocument();
  });
});
```

13.4 Test di Context e State Management

13.4.1 Test con Context

Listing 13.8: Test componenti con Context

```
// ThemeContext.jsx
import { createContext, useContext, useState } from 'react';

const ThemeContext = createContext();

export function ThemeProvider({ children }) {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme(prev => prev === 'light' ? 'dark' : 'light');
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

export function useTheme() {
  return useContext(ThemeContext);
}

// ThemedButton.jsx
import { useTheme } from './ThemeContext';

function ThemedButton() {
  const { theme, toggleTheme } = useTheme();

  return (
    <button className={theme} onClick={toggleTheme}>
      Current theme: {theme}
    </button>
  );
}

// ThemedButton.test.jsx
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import { ThemeProvider } from './ThemeContext';
import ThemedButton from './ThemedButton';

describe('ThemedButton', () => {
  test('displays current theme', () => {
    render(
      <ThemeProvider>
        <ThemedButton />
      </ThemeProvider>
    );

    expect(screen.getByText(/current theme: light/i)).toBeInTheDocument
       ();
  });

  test('toggles theme on click', async () => {
    const user = userEvent.setup();

    render(
      <ThemeProvider>
        <ThemedButton />
      </ThemeProvider>
    );

    const button = screen.getByRole('button');

    // Inizialmente light
    expect(button).toHaveTextContent(/light/i);

    // Click per toggle
    await user.click(button);
    expect(button).toHaveTextContent(/dark/i);

    // Click di nuovo
    await user.click(button);
    expect(button).toHaveTextContent(/light/i);
  });
});
```

13.4.2 Custom Render con Providers

Listing 13.9: Helper per render con providers

```
// test-utils.jsx
import { render } from '@testing-library/react';
import { ThemeProvider } from './ThemeContext';
import { AuthProvider } from './AuthContext';
import { BrowserRouter } from 'react-router-dom';

function AllProviders({ children }) {
  return (
    <BrowserRouter>
      <AuthProvider>
        <ThemeProvider>
          {children}
        </ThemeProvider>
      </AuthProvider>
    </BrowserRouter>
  );
}

function customRender(ui, options) {
  return render(ui, { wrapper: AllProviders, ...options });
}

// Re-export tutto
export * from '@testing-library/react';
export { customRender as render };

// Utilizzo
import { render, screen } from './test-utils';

test('my test', () => {
  render(<MyComponent />);
  // Component automaticamente wrappato con providers
});
```

13.5 Test di Hooks Personalizzati

Listing 13.10: Test custom hooks

```
// useCounter.js
import { useState, useCallback } from 'react';

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

  const increment = useCallback(() => {
    setCount(c => c + 1);
  }, []);

  const decrement = useCallback(() => {
    setCount(c => c - 1);
  }, []);

  const reset = useCallback(() => {
    setCount(initialValue);
  }, [initialValue]);

  return { count, increment, decrement, reset };
}

export default useCounter;

// useCounter.test.js
import { renderHook, act } from '@testing-library/react';
import useCounter from './useCounter';

describe('useCounter', () => {
  test('initializes with default value', () => {
    const { result } = renderHook(() => useCounter());
    expect(result.current.count).toBe(0);
  });

  test('initializes with custom value', () => {
    const { result } = renderHook(() => useCounter(10));
    expect(result.current.count).toBe(10);
  });

  test('increments count', () => {
    const { result } = renderHook(() => useCounter());

    act(() => {
      result.current.increment();
    });

    expect(result.current.count).toBe(1);
  });

  test('decrements count', () => {
    const { result } = renderHook(() => useCounter(5));

    act(() => {
      result.current.decrement();
    });

    expect(result.current.count).toBe(4);
  });

  test('resets to initial value', () => {
    const { result } = renderHook(() => useCounter(10));

    act(() => {
      result.current.increment();
      result.current.increment();
    });

    expect(result.current.count).toBe(12);

    act(() => {
      result.current.reset();
    });

    expect(result.current.count).toBe(10);
  });
});
```

13.6 Snapshot Testing

Listing 13.11: Snapshot tests

```
import { render } from '@testing-library/react';
import Button from './Button';

test('matches snapshot', () => {
  const { container } = render(<Button>Click me</Button>);
  expect(container).toMatchSnapshot();
});

// Snapshot inline
test('inline snapshot', () => {
  const { container } = render(<Button primary>Submit</Button>);

  expect(container.firstChild).toMatchInlineSnapshot(`
    <button
      class="primary"
    >
      Submit
    </button>
  `);
});
```

13.7 Integration Tests

Listing 13.12: Test integrazione completa

```
// TodoApp.jsx
```

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

function TodoApp() {
  const [todos, setTodos] = useState([]);
  const [input, setInput] = useState('');

  const addTodo = () => {
    if (input.trim()) {
      setTodos([...todos, { id: Date.now(), text: input, completed:
         false }]);
      setInput('');
    }
  };

  const toggleTodo = (id) => {
    setTodos(todos.map(todo =>
      todo.id === id ? { ...todo, completed: !todo.completed } : todo
    ));
  };

  const deleteTodo = (id) => {
    setTodos(todos.filter(todo => todo.id !== id));
  };

  return (
    <div>
      <h1>Todo App</h1>

      <input
        value={input}
        onChange={(e) => setInput(e.target.value)}
        onKeyDown={(e) => e.key === 'Enter' && addTodo()}
        placeholder="Add todo"
      />
      <button onClick={addTodo}>Add</button>

      <ul>
        {todos.map(todo => (
          <li key={todo.id}>
            <input
              type="checkbox"
              checked={todo.completed}
              onChange={() => toggleTodo(todo.id)}
            />
            <span style={{ textDecoration: todo.completed ? 'line-
               through' : 'none' }}>
              {todo.text}
            </span>
            <button onClick={() => deleteTodo(todo.id)}>Delete</button>
          </li>
        ))}
      </ul>

      <p>Total: {todos.length}, Completed: {todos.filter(t => t.
         completed).length}</p>
    </div>
  );
}
```

```

export default TodoApp;

// TodoApp.test.jsx
import { render, screen } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import TodoApp from './TodoApp';

describe('TodoApp Integration', () => {
  test('complete todo workflow', async () => {
    const user = userEvent.setup();
    render(<TodoApp />);

    // Verifica stato iniziale
    expect(screen.getByText(/total: 0/i)).toBeInTheDocument();

    // Aggiungi primo todo
    const input = screen.getByPlaceholderText(/add todo/i);
    await user.type(input, 'Buy milk');
    await user.click(screen.getByRole('button', { name: /add/i }));

    // Verifica todo aggiunto
    expect(screen.getByText('Buy milk')).toBeInTheDocument();
    expect(screen.getByText(/total: 1/i)).toBeInTheDocument();

    // Aggiungi secondo todo
    await user.type(input, 'Walk dog');
    await user.click(screen.getByRole('button', { name: /add/i }));

    expect(screen.getByText('Walk dog')).toBeInTheDocument();
    expect(screen.getByText(/total: 2/i)).toBeInTheDocument();

    // Completa primo todo
    const checkboxes = screen.getAllByRole('checkbox');
    await user.click(checkboxes[0]);

    expect(screen.getByText(/completed: 1/i)).toBeInTheDocument();

    // Elimina secondo todo
    const deleteButtons = screen.getAllByRole('button', { name: /delete/
      i });
    await user.click(deleteButtons[1]);

    expect(screen.queryByText('Walk dog')).not.toBeInTheDocument();
    expect(screen.getByText(/total: 1/i)).toBeInTheDocument();
  });
});
```

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

```
# Run tests con coverage
npm test -- --coverage --watchAll=false

# o aggiungi script in package.json
{
  "scripts": {
    "test": "react-scripts test",
    "test:coverage": "react-scripts test --coverage --watchAll=false"
  }
}

# Coverage output mostra:
# - Statements: linee di codice eseguite
# - Branches: condizioni if/else testate
# - Functions: funzioni chiamate
# - 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

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

// Sintassi base
const [state, setState] = useState(initialState);

// Con valore primitivo
const [count, setCount] = useState(0);
const [name, setName] = useState('');
const [isActive, setActive] = useState(false);

// Con oggetto
const [user, setUser] = useState({ name: '', age: 0 });

// Con array
const [items, setItems] = useState([]);

// Con funzione di inizializzazione (lazy init)
const [expensive, setExpensive] = useState(() => {
  return computeExpensiveValue();
});

// Update con valore diretto
setCount(5);
setName('Alice');

// Update con funzione (prev state)
setCount(prevCount => prevCount + 1);
setUser(prevUser => ({ ...prevUser, age: 30 }));
setItems(prevItems => [...prevItems, newItem]);

```

```
// Esempio completo
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>{count}</p>
      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(prev => prev - 1)}>-</button>
    </div>
  );
}
```

A.2.2 useReducer

Hook per gestire stato complesso con reducer pattern.

Listing A.2: useReducer - Reference completa

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

// Sintassi
const [state, dispatch] = useReducer(reducer, initialState, init);

// Reducer function
function reducer(state, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    case 'RESET':
      return { count: 0 };
    default:
      throw new Error('Unknown action');
  }
}

// Uso base
function Counter() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <div>
      <p>{state.count}</p>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>+</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>-</button>
      <button onClick={() => dispatch({ type: 'RESET' })}>Reset</button>
    </div>
  );
}

// Con payload
function todoReducer(state, action) {
  switch (action.type) {
    case 'ADD':
      return [...state, { id: Date.now(), text: action.payload }];
    case 'DELETE':
      return state.filter(todo => todo.id !== action.payload);
    case 'TOGGLE':
      return state.map(todo =>
        todo.id === action.payload
          ? { ...todo, completed: !todo.completed }
          : todo
      );
    default:
      return state;
  }
}

// Con lazy initialization
const init = (initialCount) => ({ count: initialCount });

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

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

// Sintassi
useEffect(effectFunction, dependencies);

// 1. Esegue ad ogni render
useEffect(() => {
  console.log('Ogni render');
});

// 2. Esegue solo al mount
useEffect(() => {
  console.log('Solo al mount');
}, []);

// 3. Esegue quando dipendenze cambiano
useEffect(() => {
  console.log('userId cambiato:', userId);
}, [userId]);

// Con cleanup
useEffect(() => {
  const timer = setInterval(() => {
    console.log('Tick');
  }, 1000);

  // Cleanup function
  return () => {
    clearInterval(timer);
  };
}, []);

// Esempi pratici
// Fetch dati
useEffect(() => {
  fetch(`/api/users/${userId}`)
    .then(res => res.json())
    .then(setUser);
}, [userId]);

// Event listener
useEffect(() => {
  const handleResize = () => setWidth(window.innerWidth);
  window.addEventListener('resize', handleResize);
  return () => window.removeEventListener('resize', handleResize);
}, []);

// Document title
useEffect(() => {
  document.title = `Count: ${count}`;
}, [count]);

// Subscription
useEffect(() => {
  const subscription = dataSource.subscribe();
  return () => subscription.unsubscribe();
}, []);
```

A.3.2 useLayoutEffect

Come useEffect ma esegue sincronamente dopo DOM mutations.

Listing A.4: useLayoutEffect - Reference

```
import { useLayoutEffect, useRef } from 'react';

// Usa per misurazioni DOM o scroll position
useLayoutEffect(() => {
  const { height } = ref.current.getBoundingClientRect();
  setHeight(height);
}, []);

// Esempio: Tooltip positioning
function Tooltip({ targetRef, children }) {
  const tooltipRef = useRef(null);
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useLayoutEffect(() => {
    const targetRect = targetRef.current.getBoundingClientRect();
    const tooltipRect = tooltipRef.current.getBoundingClientRect();

    setPosition({
      x: targetRect.left + targetRect.width / 2 - tooltipRect.width / 2,
      y: targetRect.top - tooltipRect.height - 10
    });
  }, [targetRef]);

  return (
    <div
      ref={tooltipRef}
      style={{ position: 'absolute', left: position.x, top: position.y
         }}
    >
      {children}
    </div>
  );
}
```

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

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

// Sintassi
const ref = useRef(initialValue);

// 1. DOM reference
function InputFocus() {
  const inputRef = useRef(null);

  const focusInput = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} />
      <button onClick={focusInput}>Focus</button>
    </div>
  );
}

// 2. Valore mutabile (non causa re-render)
function Timer() {
  const intervalRef = useRef(null);

  const start = () => {
    intervalRef.current = setInterval(() => {
      console.log('Tick');
    }, 1000);
  };

  const stop = () => {
    clearInterval(intervalRef.current);
  };

  return (
    <div>
      <button onClick={start}>Start</button>
      <button onClick={stop}>Stop</button>
    </div>
  );
}

// 3. Previous value
```

```
function Counter() {
  const [count, setCount] = useState(0);
  const prevCountRef = useRef();

  useEffect(() => {
    prevCountRef.current = count;
  }, [count]);

  return (
    <div>
      <p>Now: {count}, Before: {prevCountRef.current}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

// 4. Instance variable (come in class component)
function Component() {
  const renderCountRef = useRef(0);

  renderCountRef.current += 1;

  return <div>Renders: {renderCountRef.current}</div>;
}
```

A.4.2 useImperativeHandle

Personalizza l'istanza del ref esposto a componenti padre.

Listing A.6: useImperativeHandle - Reference

```
import { useRef, useImperativeHandle, forwardRef } from 'react';

// Componente figlio
const FancyInput = forwardRef((props, ref) => {
  const inputRef = useRef();

  useImperativeHandle(ref, () => ({
    focus: () => {
      inputRef.current.focus();
    },
    scrollIntoView: () => {
      inputRef.current.scrollIntoView();
    },
    getValue: () => {
      return inputRef.current.value;
    }
  }));

  return <input ref={inputRef} {...props} />;
});

// Componente padre
function Parent() {
  const inputRef = useRef();

  const handleClick = () => {
    inputRef.current.focus();
    console.log('Value:', inputRef.current.getValue());
  };

  return (
    <div>
      <FancyInput ref={inputRef} />
      <button onClick={handleClick}>Focus & Get Value</button>
    </div>
  );
}
```

A.5 Hooks di Performance

A.5.1 useMemo

Memoizza valore calcolato.

Listing A.7: useMemo - Reference completa

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

// Sintassi
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b])
    ;

// Esempi
// 1. Calcolo costoso
const sortedList = useMemo(() => {
  return items.sort((a, b) => a.value - b.value);
}, [items]);

// 2. Filtro
const filteredUsers = useMemo(() => {
  return users.filter(user =>
    user.name.toLowerCase().includes(searchTerm.toLowerCase())
  );
}, [users, searchTerm]);

// 3. Oggetto config
const chartConfig = useMemo(() => ({
  type: 'line',
  data: chartData,
  options: { responsive: true }
}), [chartData]);

// 4. Aggregazione
const stats = useMemo(() => {
  const sum = numbers.reduce((acc, n) => acc + n, 0);
  const avg = sum / numbers.length;
  return { sum, avg, count: numbers.length };
}, [numbers]);
```

A.5.2 useCallback

Memoizza funzione.

Listing A.8: useCallback - Reference completa

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

// Sintassi
const memoizedCallback = useCallback(fn, dependencies);

// Esempi
// 1. Event handler
const handleClick = useCallback(() => {
  console.log('Clicked:', count);
}, [count]);

// 2. Con parametro
const handleDelete = useCallback((id) => {
  setItems(items => items.filter(item => item.id !== id));
}, []);

// 3. Submit handler
const handleSubmit = useCallback((e) => {
  e.preventDefault();
  onSave({ name, email });
}, [name, email, onSave]);

// 4. Passato a child memoizzato
const MemoChild = memo(Child);

function Parent() {
  const handleUpdate = useCallback((data) => {
    console.log('Update:', data);
  }, []);

  return <MemoChild onUpdate={handleUpdate} />;
}
```

A.6 Hooks di Context

A.6.1 useContext

Accede a Context value.

Listing A.9: useContext - Reference completa

```
import { createContext, useContext } from 'react';

// Crea context
const ThemeContext = createContext('light');
const UserContext = createContext(null);

// Provider
function App() {
  return (
    <ThemeContext.Provider value="dark">
      <UserContext.Provider value={{ name: 'Alice', role: 'admin' }}>
        <Component />
      </UserContext.Provider>
    </ThemeContext.Provider>
  );
}

```

```
// Consumer con useContext
function Component() {
  const theme = useContext(ThemeContext);
  const user = useContext(UserContext);

  return (
    <div className={theme}>
      Hello, {user.name} ({user.role})
    </div>
  );
}

// Pattern: Custom hook per context
const AuthContext = createContext(null);

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);

  const login = (userData) => setUser(userData);
  const logout = () => setUser(null);

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth must be used within AuthProvider');
  }
  return context;
}
```

A.7 Hooks di Transizione

A.7.1 useTransition

Marca aggiornamenti come transizioni (bassa priorità).

Listing A.10: useTransition - Reference

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

function SearchResults() {
  const [query, setQuery] = useState('');
  const [results, setResults] = useState([]);
  const [isPending, startTransition] = useTransition();

  const handleChange = (e) => {
    const value = e.target.value;
    setQuery(value); // Alta priorità (immediato)

    // Bassa priorità (può essere interrotto)
    startTransition(() => {
      const filtered = allItems.filter(item =>
        item.name.includes(value)
      );
      setResults(filtered);
    });
  };

  return (
    <div>
      <input value={query} onChange={handleChange} />
      {isPending && <div>Loading...</div>}
      <Results items={results} />
    </div>
  );
}
```

A.7.2 useDeferredValue

Differisce l'aggiornamento di un valore.

Listing A.11: useDeferredValue - Reference

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

function Search() {
  const [query, setQuery] = useState('');
  const deferredQuery = useDeferredValue(query);

  // query si aggiorna immediatamente
  // deferredQuery si aggiorna con delay (bassa priorità)

  return (
    <div>
      <input value={query} onChange={(e) => setQuery(e.target.value)} />
      <SlowList query={deferredQuery} />
    </div>
  );
}

function SlowList({ query }) {
  // Questo componente usa deferredQuery
  // Re-render ritardato se user sta ancora digitando
  const items = useMemo(() => {
    return hugeList.filter(item => item.includes(query));
  }, [query]);

  return <div>{items.map(...)}</div>;
}
```

A.8 Altri Hooks

A.8.1 useId

Genera ID unici stabili.

Listing A.12: useId - Reference

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

```

```
function FormField({ label }) {
  const id = useId();

  return (
    <div>
      <label htmlFor={id}>{label}</label>
      <input id={id} />
    </div>
  );
}

// Multiple IDs
function Form() {
  const baseId = useId();

  return (
    <form>
      <label htmlFor={`${baseId}-name`}>Name</label>
      <input id={`${baseId}-name`} />

      <label htmlFor={`${baseId}-email`}>Email</label>
      <input id={`${baseId}-email`} />
    </form>
  );
}
```

A.8.2 useDebugValue

Label custom hook in DevTools.

Listing A.13: useDebugValue - Reference

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  // Mostra "Online" o "Offline" in DevTools
  useDebugValue(isOnline ? 'Online' : 'Offline');

  // Con formatter (eseguito solo quando DevTools ispeziona)
  useDebugValue(date, date => date.toDateString());

  return isOnline;
}
```

A.9 Custom Hooks Pattern

A.9.1 useFetch

Listing A.14: Custom hook per fetch

```
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const controller = new AbortController();

    setLoading(true);

    fetch(url, { signal: controller.signal })
      .then(res => res.json())
      .then(setData)
      .catch(setError)
      .finally(() => setLoading(false));

    return () => controller.abort();
  }, [url]);

  return { data, loading, error };
}
```

A.9.2 useLocalStorage

Listing A.15: Custom hook per localStorage

```
function useLocalStorage(key, initialValue) {
  const [value, setValue] = useState(() => {
    const stored = localStorage.getItem(key);
    return stored ? JSON.parse(stored) : initialValue;
  });

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}
```

A.9.3 useToggle

Listing A.16: Custom hook per toggle

```
function useToggle(initialValue = false) {
  const [value, setValue] = useState(initialValue);

  const toggle = useCallback(() => {
    setValue(v => !v);
  }, []);

  return [value, toggle];
}
```

A.9.4 usePrevious

Listing A.17: Custom hook per previous value

```
function usePrevious(value) {
  const ref = useRef();

  useEffect(() => {
    ref.current = value;
  }, [value]);

  return ref.current;
}
```

A.9.5 useOnClickOutside

Listing A.18: Custom hook per click fuori elemento

```
function useOnClickOutside(ref, handler) {
  useEffect(() => {
    const listener = (event) => {
      if (!ref.current || ref.current.contains(event.target)) {
        return;
      }
      handler(event);
    };

    document.addEventListener('mousedown', listener);
    document.addEventListener('touchstart', listener);

    return () => {
      document.removeEventListener('mousedown', listener);
      document.removeEventListener('touchstart', listener);
    };
  }, [ref, handler]);
}
```

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

```
// SBAGLIATO: Hook in condizione
if (condition) {
  useEffect(() => {});
}

// SBAGLIATO: Hook in loop
for (let i = 0; i < 10; i++) {
  useState(i);
}

// SBAGLIATO: Hook in funzione normale
```

```
function regularFunction() {
  const [state, setState] = useState(0);
}

// CORRETTO: Hook al top level
function Component() {
  const [state, setState] = useState(0);

  useEffect(() => {
    if (condition) {
      // Logica qui
    }
  });

  return <div>{state}</div>;
}
```

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

```
todo-app/
          src/
                components/
                      TodoForm.jsx
                      TodoList.jsx
                      TodoItem.jsx
                      TodoFilters.jsx
                      TodoStats.jsx
                hooks/
                      useTodos.js
                      useLocalStorage.js
                App.jsx
                main.jsx
          package.json
          vite.config.js
```

B.2.2 Custom Hooks

Listing B.2: useLocalStorage hook

```
// hooks/useLocalStorage.js
import { useState, useEffect } from 'react';

function useLocalStorage(key, initialValue) {
  const [value, setValue] = useState(() => {
    try {
      const item = window.localStorage.getItem(key);
      return item ? JSON.parse(item) : initialValue;
    } catch (error) {
      console.error('Error loading from localStorage:', error);
      return initialValue;
    }
  });

  useEffect(() => {
    try {
      window.localStorage.setItem(key, JSON.stringify(value));
    } catch (error) {
      console.error('Error saving to localStorage:', error);
    }
  }, [key, value]);

  return [value, setValue];
}

export default useLocalStorage;
```

Listing B.3: useTodos hook

```
// hooks/useTodos.js
import { useCallback } from 'react';
import useLocalStorage from './useLocalStorage';

function useTodos() {
  const [todos, setTodos] = useLocalStorage('todos', []);

  const addTodo = useCallback((text) => {
    const newTodo = {
      id: Date.now(),
      text,
      completed: false,
      createdAt: new Date().toISOString()
    };
    setTodos(prev => [...prev, newTodo]);
  }, [setTodos]);

  const toggleTodo = useCallback((id) => {
    setTodos(prev =>
      prev.map(todo =>
        todo.id === id
          ? { ...todo, completed: !todo.completed }
          : todo
      )
    );
  }, [setTodos]);

  const deleteTodo = useCallback((id) => {
    setTodos(prev => prev.filter(todo => todo.id !== id));
  }, [setTodos]);

  const updateTodo = useCallback((id, newText) => {
    setTodos(prev =>
      prev.map(todo =>
        todo.id === id
          ? { ...todo, text: newText }
          : todo
      )
    );
  }, [setTodos]);

  const clearCompleted = useCallback(() => {
    setTodos(prev => prev.filter(todo => !todo.completed));
  }, [setTodos]);

  return {
    todos,
    addTodo,
    toggleTodo,
    deleteTodo,
    updateTodo,
    clearCompleted
  };
}

export default useTodos;
```

B.2.3 Componenti

Listing B.4: TodoForm component

```
// components/TodoForm.jsx
import { useState } from 'react';

function TodoForm({ onAdd }) {
  const [input, setInput] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (input.trim()) {
      onAdd(input.trim());
      setInput('');
    }
  };

  return (
    <form onSubmit={handleSubmit} className="todo-form">
      <input
        type="text"
        value={input}
        onChange={(e) => setInput(e.target.value)}
        placeholder="Cosa devi fare?"
        className="todo-input"
      />
      <button type="submit" className="todo-add-btn">
        Aggiungi
      </button>
    </form>
  );
}

export default TodoForm;
```

Listing B.5: TodoItem component

```
// components/TodoItem.jsx
import { useState } from 'react';

function TodoItem({ todo, onToggle, onDelete, onUpdate }) {
  const [isEditing, setIsEditing] = useState(false);
  const [editText, setEditText] = useState(todo.text);

  const handleSave = () => {
    if (editText.trim()) {
      onUpdate(todo.id, editText.trim());
      setIsEditing(false);
    }
  };

  const handleCancel = () => {
    setEditText(todo.text);
    setIsEditing(false);
  };

  if (isEditing) {
    return (
      <li className="todo-item editing">
        <input
          type="text"
          value={editText}
          onChange={(e) => setEditText(e.target.value)}
          onKeyDown={(e) => {
            if (e.key === 'Enter') handleSave();
            if (e.key === 'Escape') handleCancel();
          }}
          className="todo-edit-input"
          autoFocus
        />
        <button onClick={handleSave} className="save-btn">Salva</button>
        <button onClick={handleCancel} className="cancel-btn">Annulla</
            button>
      </li>
    );
  }

  return (
    <li className={`todo-item ${todo.completed ? 'completed' : ''}`}>
      <input
        type="checkbox"
        checked={todo.completed}
        onChange={() => onToggle(todo.id)}
        className="todo-checkbox"
      />

      <span
        onDoubleClick={() => setIsEditing(true)}
        className="todo-text"
      >
        {todo.text}
      </span>

      <div className="todo-actions">
        <button
```

```
          onClick={() => setIsEditing(true)}
          className="edit-btn"
          title="Modifica"
        >

        </button>
        <button
          onClick={() => onDelete(todo.id)}
          className="delete-btn"
          title="Elimina"
        >

        </button>
      </div>
    </li>
  );
}

export default TodoItem;
```

Listing B.6: TodoList component

```
// components/TodoList.jsx
import TodoItem from './TodoItem';

function TodoList({ todos, onToggle, onDelete, onUpdate }) {
  if (todos.length === 0) {
    return (
      <div className="empty-state">
        <p>Nessun todo. Aggiungine uno!</p>
      </div>
    );
  }

  return (
    <ul className="todo-list">
      {todos.map(todo => (
        <TodoItem
          key={todo.id}
          todo={todo}
          onToggle={onToggle}
          onDelete={onDelete}
          onUpdate={onUpdate}
        />
      ))}
    </ul>
  );
}

export default TodoList;
```

Listing B.7: TodoFilters component

```
// components/TodoFilters.jsx
function TodoFilters({ filter, setFilter }) {
  const filters = [
    { value: 'all', label: 'Tutti' },
    { value: 'active', label: 'Attivi' },
    { value: 'completed', label: 'Completati' }
  ];

  return (
    <div className="todo-filters">
      {filters.map(f => (
        <button
          key={f.value}
          onClick={() => setFilter(f.value)}
          className={`filter-btn ${filter === f.value ? 'active' : ''}`}
        >
          {f.label}
        </button>
      ))}
    </div>
  );
}

export default TodoFilters;
```

Listing B.8: TodoStats component

```
// components/TodoStats.jsx
function TodoStats({ todos }) {
  const total = todos.length;
  const completed = todos.filter(t => t.completed).length;
  const active = total - completed;
  const percentage = total > 0 ? Math.round((completed / total) * 100) :
      0;

  return (
    <div className="todo-stats">
      <div className="stat">
        <span className="stat-label">Totale:</span>
        <span className="stat-value">{total}</span>
      </div>

      <div className="stat">
        <span className="stat-label">Attivi:</span>
        <span className="stat-value">{active}</span>
      </div>

      <div className="stat">
        <span className="stat-label">Completati:</span>
        <span className="stat-value">{completed}</span>
      </div>

      <div className="stat">
        <span className="stat-label">Progresso:</span>
        <span className="stat-value">{percentage}%</span>
      </div>

      <div className="progress-bar">
        <div
          className="progress-fill"
          style={{ width: `${percentage}%` }}
        />
      </div>
    </div>
  );
}

export default TodoStats;
```

B.2.4 App Principale

Listing B.9: App.jsx completo

```
// App.jsx
import { useState, useMemo } from 'react';
import useTodos from './hooks/useTodos';
import TodoForm from './components/TodoForm';
import TodoList from './components/TodoList';
import TodoFilters from './components/TodoFilters';
import TodoStats from './components/TodoStats';
import './App.css';

function App() {
  const {
    todos,
    addTodo,
    toggleTodo,
    deleteTodo,
    updateTodo,
    clearCompleted
  } = useTodos();

  const [filter, setFilter] = useState('all');

  // Filtra todos in base al filtro selezionato
  const filteredTodos = useMemo(() => {
    switch (filter) {
      case 'active':
        return todos.filter(todo => !todo.completed);
      case 'completed':
        return todos.filter(todo => todo.completed);
      default:
        return todos;
    }
  }, [todos, filter]);

  const hasCompleted = todos.some(todo => todo.completed);

  return (
    <div className="app">
      <header className="app-header">
        <h1>   Todo App</h1>
      </header>

      <main className="app-main">
        <TodoForm onAdd={addTodo} />

        <TodoStats todos={todos} />

        <TodoFilters filter={filter} setFilter={setFilter} />

        <TodoList
```

```
          todos={filteredTodos}
          onToggle={toggleTodo}
          onDelete={deleteTodo}
          onUpdate={updateTodo}
        />

        {hasCompleted && (
          <button
            onClick={clearCompleted}
            className="clear-completed-btn"
          >
            Elimina Completati
          </button>
        )}
      </main>
    </div>
  );
}

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

```
ecommerce-app/
          src/
                components/
                      ProductCard.jsx
                      ProductList.jsx
                      ProductFilters.jsx
                      Cart.jsx
                      CartItem.jsx
                      Checkout.jsx
                context/
                      CartContext.jsx
                data/
                      products.js
                App.jsx
                main.jsx
```

B.3.2 Context per Carrello

Listing B.11: CartContext.jsx

```
// context/CartContext.jsx
import { createContext, useContext, useReducer } from 'react';

const CartContext = createContext();

function cartReducer(state, action) {
  switch (action.type) {
    case 'ADD_ITEM':
      const existingIndex = state.items.findIndex(
        item => item.id === action.payload.id
      );

      if (existingIndex >= 0) {
        const newItems = [...state.items];
        newItems[existingIndex].quantity += 1;
        return { ...state, items: newItems };
      }

      return {
        ...state,
        items: [...state.items, { ...action.payload, quantity: 1 }]
      };

    case 'REMOVE_ITEM':
      return {
        ...state,
        items: state.items.filter(item => item.id !== action.payload)
      };

    case 'UPDATE_QUANTITY':
      return {
        ...state,
        items: state.items.map(item =>
          item.id === action.payload.id
            ? { ...item, quantity: action.payload.quantity }
            : item
        )
      };

    case 'CLEAR_CART':
      return { ...state, items: [] };

    default:
      return state;
  }
}

export function CartProvider({ children }) {
  const [state, dispatch] = useReducer(cartReducer, { items: [] });

  const addItem = (product) => {
    dispatch({ type: 'ADD_ITEM', payload: product });
  };

  const removeItem = (productId) => {
    dispatch({ type: 'REMOVE_ITEM', payload: productId });
  };

  const updateQuantity = (productId, quantity) => {
    if (quantity <= 0) {
      removeItem(productId);
    } else {
      dispatch({ type: 'UPDATE_QUANTITY', payload: { id: productId,
         quantity } });
    }
  };

  const clearCart = () => {
    dispatch({ type: 'CLEAR_CART' });
  };

  const totalItems = state.items.reduce((sum, item) => sum + item.
     quantity, 0);

  const totalPrice = state.items.reduce(
    (sum, item) => sum + (item.price * item.quantity),
    0
  );

  return (
    <CartContext.Provider
      value={{
        items: state.items,
        addItem,
        removeItem,
        updateQuantity,
        clearCart,
        totalItems,
        totalPrice
      }}
    >
      {children}
    </CartContext.Provider>
  );
}

export function useCart() {
  const context = useContext(CartContext);
  if (!context) {
    throw new Error('useCart must be used within CartProvider');
  }
  return context;
}
```

B.3.3 Componenti Prodotto

Listing B.12: ProductCard component

```
// components/ProductCard.jsx
import { useCart } from '../context/CartContext';

function ProductCard({ product }) {
  const { addItem } = useCart();

  return (
    <div className="product-card">
      <img
        src={product.image}
        alt={product.name}
        className="product-image"
      />

      <div className="product-info">
        <h3 className="product-name">{product.name}</h3>
        <p className="product-category">{product.category}</p>
        <p className="product-description">{product.description}</p>

        <div className="product-footer">
          <span className="product-price">{product.price.toFixed(2)}
             </span>

          <button
            onClick={() => addItem(product)}
            className="add-to-cart-btn"
          >
            Aggiungi al Carrello
          </button>
        </div>
      </div>
    </div>
  );
}

export default ProductCard;
```

Listing B.13: ProductList component

```
// components/ProductList.jsx
import { useState, useMemo } from 'react';
import ProductCard from './ProductCard';
import ProductFilters from './ProductFilters';

function ProductList({ products }) {
  const [searchTerm, setSearchTerm] = useState('');
  const [category, setCategory] = useState('all');
  const [sortBy, setSortBy] = useState('name');

  // Filtra e ordina prodotti
  const filteredProducts = useMemo(() => {
    let result = products;

    // Filtra per ricerca
    if (searchTerm) {
      result = result.filter(p =>
        p.name.toLowerCase().includes(searchTerm.toLowerCase())
      );
    }

    // Filtra per categoria
    if (category !== 'all') {
      result = result.filter(p => p.category === category);
    }

    // Ordina
    result = [...result].sort((a, b) => {
      switch (sortBy) {
        case 'price-asc':
          return a.price - b.price;
        case 'price-desc':
          return b.price - a.price;
        case 'name':
        default:
          return a.name.localeCompare(b.name);
      }
    });

    return result;
  }, [products, searchTerm, category, sortBy]);

  return (
    <div className="product-list-container">
      <ProductFilters
        searchTerm={searchTerm}
        setSearchTerm={setSearchTerm}
        category={category}
        setCategory={setCategory}
        sortBy={sortBy}
        setSortBy={setSortBy}
      />

      <div className="product-grid">
        {filteredProducts.length === 0 ? (
          <p>Nessun prodotto trovato</p>
        ) : (
          filteredProducts.map(product => (
            <ProductCard key={product.id} product={product} />
          ))
        )}
      </div>
    </div>
  );
}

export default ProductList;
```

Listing B.14: Cart component

```
// components/Cart.jsx
import { useCart } from '../context/CartContext';
import CartItem from './CartItem';

function Cart({ isOpen, onClose }) {
  const { items, totalItems, totalPrice, clearCart } = useCart();

  if (!isOpen) return null;

  return (
    <div className="cart-overlay" onClick={onClose}>
      <div className="cart-sidebar" onClick={(e) => e.stopPropagation()
         }>
        <div className="cart-header">
          <h2>Carrello ({totalItems})</h2>
          <button onClick={onClose} className="close-btn">  </button>
        </div>

        <div className="cart-items">
          {items.length === 0 ? (
            <p className="empty-cart">Il carrello è vuoto</p>
          ) : (
            items.map(item => (
              <CartItem key={item.id} item={item} />
            ))
          )}
        </div>

        {items.length > 0 && (
          <div className="cart-footer">
            <div className="cart-total">
              <span>Totale:</span>
              <span className="total-price">{totalPrice.toFixed(2)}   </
                span>
            </div>

            <button className="checkout-btn">
              Procedi al Checkout
            </button>

            <button onClick={clearCart} className="clear-cart-btn">
              Svuota Carrello
            </button>
          </div>
        )}
      </div>
    </div>
  );
}

export default Cart;
```

B.3.4 App Principale E-commerce

Listing B.15: App E-commerce completo

```
// App.jsx
import { useState } from 'react';
import { CartProvider } from './context/CartContext';
import ProductList from './components/ProductList';
import Cart from './components/Cart';
import { products } from './data/products';
import './App.css';

function App() {
  const [cartOpen, setCartOpen] = useState(false);

  return (
    <CartProvider>
      <div className="app">
        <header className="app-header">
          <h1>     E-Shop</h1>

          <button
            onClick={() => setCartOpen(true)}
            className="cart-icon-btn"
          >
            Carrello
          </button>
        </header>

        <main className="app-main">
          <ProductList products={products} />
        </main>

        <Cart isOpen={cartOpen} onClose={() => setCartOpen(false)} />
      </div>
    </CartProvider>
  );
}

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

```
dashboard-app/
          src/
                components/
                      MetricCard.jsx
                      Chart.jsx
                      RecentActivity.jsx
                      UserTable.jsx
                      Sidebar.jsx
                hooks/
                      useAnalytics.js
                App.jsx
                main.jsx
```

B.4.2 Hook per Analytics

Listing B.17: useAnalytics hook

```
// hooks/useAnalytics.js
import { useState, useEffect } from 'react';

function useAnalytics() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchAnalytics = async () => {
      try {
        setLoading(true);

        // Simula API call
        await new Promise(resolve => setTimeout(resolve, 1000));

        const mockData = {
          metrics: {
            totalUsers: 15420,
            activeUsers: 8732,
            revenue: 245689,
            conversion: 3.2
          },
          chartData: Array.from({ length: 12 }, (_, i) => ({
            month: ['Gen', 'Feb', 'Mar', 'Apr', 'Mag', 'Giu',
                    'Lug', 'Ago', 'Set', 'Ott', 'Nov', 'Dic'][i],
            value: Math.floor(Math.random() * 5000) + 2000
          })),
          recentActivity: [
            { id: 1, user: 'Mario Rossi', action: 'Nuovo ordine', time:
                '2 min fa' },
            { id: 2, user: 'Laura Bianchi', action: 'Registrazione',
                time: '15 min fa' },
            { id: 3, user: 'Giuseppe Verdi', action: 'Pagamento', time:
                '1 ora fa' }
          ]
        };

        setData(mockData);
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };

    fetchAnalytics();

    // Refresh ogni 30 secondi
    const interval = setInterval(fetchAnalytics, 30000);

    return () => clearInterval(interval);
  }, []);

  return { data, loading, error };
}

export default useAnalytics;
```

B.4.3 Dashboard App Completo

Listing B.18: Dashboard App.jsx

```
// App.jsx
import useAnalytics from './hooks/useAnalytics';
import MetricCard from './components/MetricCard';
import Chart from './components/Chart';
import RecentActivity from './components/RecentActivity';

function App() {
  const { data, loading, error } = useAnalytics();

  if (loading) {
    return (
      <div className="loading-container">
        <div className="spinner"></div>
        <p>Caricamento dashboard...</p>
      </div>
    );
  }

  if (error) {
    return (
      <div className="error-container">
        <p>Errore: {error}</p>
      </div>
    );
  }

  return (
    <div className="dashboard">
      <header className="dashboard-header">
        <h1>    Analytics Dashboard</h1>
      </header>

      <main className="dashboard-main">
        <section className="metrics-grid">
          <MetricCard
            title="Utenti Totali"
            value={data.metrics.totalUsers.toLocaleString()}
            icon="    "
            trend="+12%"
          />

          <MetricCard
            title="Utenti Attivi"
            value={data.metrics.activeUsers.toLocaleString()}
            icon="   "
            trend="+8%"
          />

          <MetricCard
            title="Revenue"
            value={` $ {(data.metrics.revenue / 1000).toFixed(1)}K`}
            icon="    "
            trend="+23%"
          />

          <MetricCard
            title="Conversione"
            value={`${data.metrics.conversion}%`}
            icon="    "
            trend="+0.5%"
          />
        </section>

        <section className="chart-section">
          <Chart data={data.chartData} />
        </section>

        <section className="activity-section">
```

```
          <RecentActivity activities={data.recentActivity} />
        </section>
      </main>
    </div>
  );
}

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

```
# Build con Vite
npm run build

# Preview build locale
npm run preview

# Deploy su Netlify
netlify deploy --prod --dir=dist

# Deploy su Vercel
vercel --prod

# Deploy su GitHub Pages
npm run build
git add dist -f
git commit -m "Deploy"
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.