React e Vite: Guida Completa

Fondamenti di React

Componenti e JSX

```jsx
// Componente funzionale
function Welcome(props) {
  return <h1>Ciao, {props.name}</h1>;
}

// Componente classe (approccio legacy)
class Welcome extends React.Component {
  render() {
    return <h1>Ciao, {this.props.name}</h1>;
  }
}
```

React Hooks Essenziali

useState

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

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

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

useEffect

jsx

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

function Example() {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Eseguito dopo il render
    fetchData().then(result => setData(result));

    // Cleanup function (opzionale)
    return () => {
      // Eseguito quando il componente viene smontato
      cleanup();
    };
  }, [/* dependencies */]); // Array di dipendenze

  return <div>{data ? data.title : 'Caricamento...'}</div>;
}
```

useContext

jsx

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

// Creazione context
const ThemeContext = createContext(null);

// Provider
function App() {
  const [theme, setTheme] = useState('light');

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      <MainComponent />
    </ThemeContext.Provider>
  );
}

// Consumer
function MainComponent() {
  const { theme, setTheme } = useContext(ThemeContext);

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

useReducer

jsx

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

// Reducer function
function counterReducer(state, action) {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(counterReducer, { count: 0 });

  return (
    <div>
      Count: {state.count}
      <button onClick={() => dispatch({ type: 'increment' })}>+</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
    </div>
  );
}
```

useRef

jsx

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

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

  useEffect(() => {
    // Focus sull'input quando il componente viene montato
    inputRef.current.focus();
  }, []);

  return <input ref={inputRef} type="text" />;
}
```

useMemo e useCallback

jsx

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

function ExpensiveCalculation({ list, onItemClick }) {
  // Memorizza il risultato di un calcolo costoso
  const processedList = useMemo(() => {
    return list.map(item => expensiveProcess(item));
  }, [list]);

  // Memorizza una funzione callback
  const handleClick = useCallback((id) => {
    console.log('Item clicked:', id);
    onItemClick(id);
  }, [onItemClick]);

  return (
    <ul>
      {processedList.map(item => (
        <li key={item.id} onClick={() => handleClick(item.id)}>
          {item.name}
        </li>
      ))}
    </ul>
  );
}
```

React Custom Hooks

jsx

```
// Custom hook per gestire form
function useForm(initialValues) {
  const [values, setValues] = useState(initialValues);

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

  const resetForm = () => {
    setValues(initialValues);
  };

  return { values, handleChange, resetForm };
}

// Utilizzo
function LoginForm() {
  const { values, handleChange, resetForm } = useForm({
    username: '',
    password: ''
  });

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log('Form values:', values);
    resetForm();
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        name="username"
        value={values.username}
        onChange={handleChange}
      />
      <input
        type="password"
        name="password"
        value={values.password}
        onChange={handleChange}
      />
      <button type="submit">Login</button>
```

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

Vite per React

Setup di un Progetto

bash

```bash
# Creazione nuovo progetto
npm create vite@latest my-react-app -- --template react-ts

# Installazione dipendenze
cd my-react-app
npm install

# Avvio server di sviluppo
npm run dev
```

Configurazione di Vite

javascript

```javascript
// vite.config.js
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import path from 'path';

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
  server: {
    port: 3000,
    open: true,
    proxy: {
      '/api': {
        target: 'http://localhost:8080',
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, '')
      }
    }
  },
  build: {
    outDir: 'dist',
    sourcemap: true,
  }
});
```

Ottimizzazione con Vite

- **Hot Module Replacement (HMR)**: Aggiorna il browser senza refresh completo
- **Code splitting automatico**: Divide il codice in chunks per caricamento ottimizzato
- **Lazy loading**: Importazioni dinamiche per componenti caricati on-demand

jsx

```jsx
import { lazy, Suspense } from 'react';

// Lazy loading di componenti
const LazyComponent = lazy(() => import('./LazyComponent'));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}
```

Patterns React Avanzati

Compound Components

jsx

```jsx
// Esempio di pattern Compound Components
function Tabs({ children, defaultTab }) {
  const [activeTab, setActiveTab] = useState(defaultTab);

  return (
    <TabsContext.Provider value={{ activeTab, setActiveTab }}>
      {children}
    </TabsContext.Provider>
  );
}

Tabs.TabList = function TabList({ children }) {
  return <div className="tab-list">{children}</div>;
};

Tabs.Tab = function Tab({ id, children }) {
  const { activeTab, setActiveTab } = useContext(TabsContext);
  return (
    <button
      className={activeTab === id ? 'active' : ''}
      onClick={() => setActiveTab(id)}
    >
      {children}
    </button>
  );
};

Tabs.TabPanel = function TabPanel({ id, children }) {
  const { activeTab } = useContext(TabsContext);
  if (activeTab !== id) return null;
  return <div className="tab-panel">{children}</div>;
};

// Utilizzo
<Tabs defaultTab="tab1">
  <Tabs.TabList>
    <Tabs.Tab id="tab1">Tab 1</Tabs.Tab>
    <Tabs.Tab id="tab2">Tab 2</Tabs.Tab>
  </Tabs.TabList>
  <Tabs.TabPanel id="tab1">Contenuto Tab 1</Tabs.TabPanel>
  <Tabs.TabPanel id="tab2">Contenuto Tab 2</Tabs.TabPanel>
</Tabs>
```

Render Props

jsx

```jsx
// Pattern Render Props
function MouseTracker({ render }) {
  const [position, setPosition] = useState({ x: 0, y: 0 });

  useEffect(() => {
    const handleMouseMove = (e) => {
      setPosition({ x: e.clientX, y: e.clientY });
    };

    window.addEventListener('mousemove', handleMouseMove);
    return () => {
      window.removeEventListener('mousemove', handleMouseMove);
    };
  }, []);

  return render(position);
}

// Utilizzo
<MouseTracker
  render={({ x, y }) => (
    <div>
      La posizione del mouse è: {x}, {y}
    </div>
  )}
/>
```

Higher-Order Components (HOC)

jsx

```jsx
// Higher-Order Component
function withAuth(Component) {
  return function AuthenticatedComponent(props) {
    const { isAuthenticated } = useAuth();

    if (!isAuthenticated) {
      return <Redirect to="/login" />;
    }

    return <Component {...props} />;
  };
}

// Utilizzo
const ProtectedDashboard = withAuth(Dashboard);
```

Testing in React

Jest e React Testing Library

jsx

```jsx
// Esempio di test con React Testing Library
import { render, screen, fireEvent } from '@testing-library/react';
import Counter from './Counter';

test('renders counter and increments correctly', () => {
  render(<Counter />);

  // Verifica il rendering iniziale
  expect(screen.getByText(/count: 0/i)).toBeInTheDocument();

  // Simula il click sul bottone
  fireEvent.click(screen.getByRole('button', { name: /increment/i }));

  // Verifica l'aggiornamento del contatore
  expect(screen.getByText(/count: 1/i)).toBeInTheDocument();
});
```

State Management in React

Context API

Già illustrato sopra con `useContext`

Redux (cenni)

jsx

```
// Action types
const INCREMENT = 'INCREMENT';
const DECREMENT = 'DECREMENT';

// Action creators
const increment = () => ({ type: INCREMENT });
const decrement = () => ({ type: DECREMENT });

// Reducer
function counterReducer(state = { count: 0 }, action) {
  switch (action.type) {
    case INCREMENT:
      return { count: state.count + 1 };
    case DECREMENT:
      return { count: state.count - 1 };
    default:
      return state;
  }
}

// Store setup (tramite react-redux)
import { Provider } from 'react-redux';
import { createStore } from 'redux';

const store = createStore(counterReducer);

function App() {
  return (
    <Provider store={store}>
      <Counter />
    </Provider>
  );
}

// Utilizzo con hooks
import { useSelector, useDispatch } from 'react-redux';

function Counter() {
  const count = useSelector(state => state.count);
  const dispatch = useDispatch();

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

```
    </div>
  );
}
```

Esercizi Pratici

1. **Todo App con React e Vite**:
   - Implementa CRUD completo
   - Usa useState e useReducer per la gestione dello stato
   - Applica CSS modules o Tailwind per lo styling
2. **Dashboard con dati fetching**:
   - Usa useEffect per caricare dati da API
   - Implementa loading states e gestione errori
   - Crea componenti riutilizzabili per visualizzare i dati
3. **Form complesso con validazione**:
   - Implementa form multi-step
   - Validazione custom con useState e useEffect
   - Gestione dello stato del form tramite useReducer