# Complete React CheatSheet

**- Developer Shaurya**

## Important Links

- **React Cheatsheet :** https://developershaurya.com/react-cheatsheet/

- **Website:** https://developershaurya.com/

- **All Cheatsheets :** https://developershaurya.com/cheatsheets/

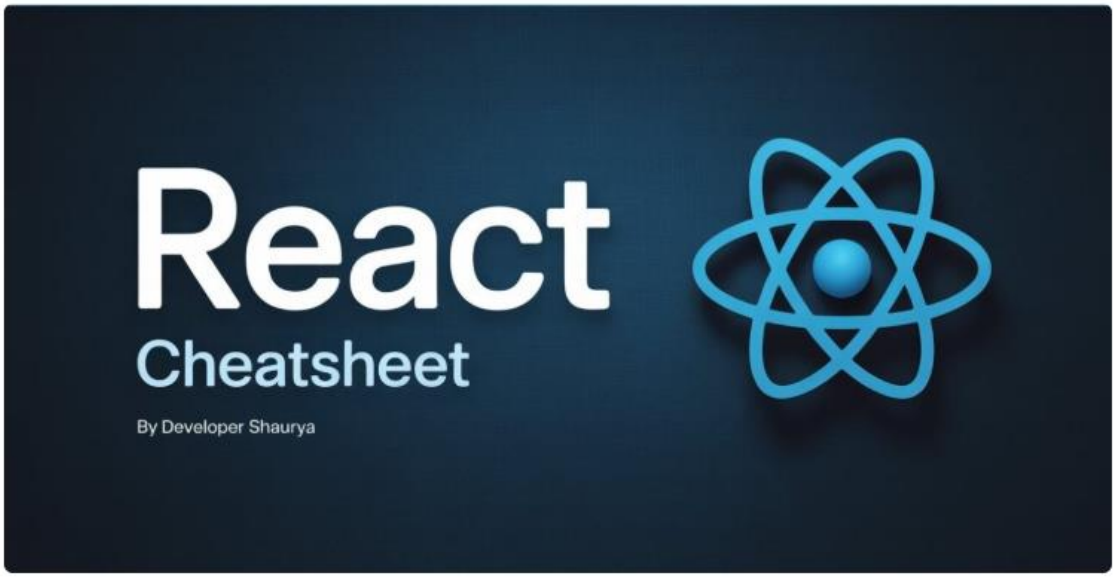
- **Youtube:** https://www.youtube.com/@DeveloperShaurya

- **(Developer Shaurya) Community :** https://t.me/developer_shaurya (Telegram)

CheatSheet React

# React CheatSheet – A Quick Guide for Beginners & Developers

by Developer Shaurya • September 5, 2025 • 0




This is an **Ultimate All-in-One React Master Cheatsheet** — literally covering **everything a React developer would need from beginner to advance**: CRA, Vite, components, hooks, routing, forms, state management, context, TypeScript, Redux, performance optimization, lazy loading, error boundaries, portals, testing, animations, SSR concepts, advanced patterns, and more.

**Download Pdf**

## 1. Project Setup

### Create React App (CRA)

```
npx create-react-app my-app
cd my-app
npm start
```

This one (CRA) has deprecated now so I would recommend you to go with Vite.

### Vite

```
npm create vite@latest my-app
# Choose: React + JS or React + TS
cd my-app
npm install
npm run dev
```

**Vite Advantages:** fast HMR, lightweight, modern bundler.

## 2. File Structure

**CRA**

```
src/
 ├── App.js
 ├── index.js
 ├── components/
 └── assets/
```

**Vite**

```
src/
 ├── main.jsx
 ├── App.jsx
 ├── components/
 └── assets/
index.html
vite.config.js
```

## 3. Components

### Functional Component

```
const Hello = ({ name }) => <h1>Hello, {name}!</h1>;
```

### Class Component

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

### Arrow Function

```
const Hello = ({ name }) => <h1>Hello, {name}</h1>;
```

### Props & State

```
import { useState } from 'react';
const Counter = () => {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count+1)}>{count}</button>;
};
```

## 4. JSX Basics

```
<div>
  <h1>Hello React!</h1>
  {isLoggedIn ? <Dashboard /> : <Login />}
  {items.map(item => <li key={item.id}>{item.name}</li>)}
</div>
```

**Rules**

- Use {} for JS expressions
- Use `className` instead of `class`
- Single root element or fragment `<>...</>`

## 5. Lifecycle & useEffect

```
useEffect(() => { console.log("Mounted"); return () =>
console.log("Unmounted"); }, []);
useEffect(() => { console.log("Dep changed"); }, [dep]);
useEffect(() => { console.log("Runs every render"); });
```

## 6. Event Handling

```
<button onClick={handleClick}>Click</button>
<input onChange={e => setValue(e.target.value)} />
<form onSubmit={handleSubmit}></form>
```

## 7. Forms

```
const [name, setName] = useState("");
<input value={name} onChange={e => setName(e.target.value)} />
<form onSubmit={e => { e.preventDefault(); alert(name); }}></form>
```

- **Controlled Components:** value linked to state
- **Uncontrolled Components:** use `ref`

## 8. Conditional Rendering & Lists

```
{isLoggedIn ? <Dashboard /> : <Login />}
<ul>{items.map(item => <li key={item.id}>{item.name}</li>)}</ul>
```

- Always use **unique keys** for lists.

## 9. Context API

```
import { createContext, useContext } from 'react';
const ThemeContext = createContext('light');

const App = () => <ThemeContext.Provider value="dark"><Toolbar/>
</ThemeContext.Provider>;
const Toolbar = () => <Button />;
const Button = () => {
  const theme = useContext(ThemeContext);
  return <button>{theme}</button>;
};
```

## 10. React Router v6

```
npm install react-router-dom
```

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

<BrowserRouter>
  <nav>
    <Link to="/">Home</Link>
    <Link to="/about">About</Link>
  </nav>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/about" element={<About />} />
  </Routes>
</BrowserRouter>
```

## 11. Data Fetching

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

const DataFetcher = () => {
  const [data, setData] = useState([]);
  useEffect(() => { axios.get('/api').then(res => setData(res.data)); },
[]);
  return <ul>{data.map(d => <li key={d.id}>{d.title}</li>)}</ul>;
}
```

## 12. Hooks Overview

| Hook | Usage |
| --- | --- |
| `useState` | Local state |
| `useEffect` | Lifecycle / side effects |
| `useContext` | Access context |
| `useReducer` | Complex state |
| `useRef` | DOM access / persistent values |
| `useMemo` | Memoize value |
| `useCallback` | Memoize function |
| `useLayoutEffect` | Runs before paint |
| `useImperativeHandle` | Customize child ref |
| `useDebugValue` | Debug custom hooks |

## 13. Custom Hooks

```
function useFetch(url) {
  const [data, setData] = useState([]);
  useEffect(() => { fetch(url).then(res => res.json()).then(setData); },
[url]);
  return data;
}
```

## 14. Performance Optimization

- `React.memo(Component)`
- `useMemo(() => value, [deps])`
- `useCallback(() => func, [deps])`
- Lazy loading:

```
import { lazy, Suspense } from 'react';
const About = lazy(() => import('./About'));
<Suspense fallback={<div>Loading...</div>}><About /></Suspense>
```

## 15. Error Boundaries

```
class ErrorBoundary extends React.Component {
  state = { hasError: false };
  static getDerivedStateFromError() { return { hasError: true }; }
  componentDidCatch(error, info) { console.log(error, info); }
  render() { return this.state.hasError ? <h1>Error!</h1> :
this.props.children; }
}
```

## 16. Portals

```
import { createPortal } from 'react-dom';
const Modal = ({ children }) => createPortal(<div>{children}</div>,
document.getElementById('modal-root'));
```

## 17. TypeScript

```
type ButtonProps = { text: string; onClick: () => void };
const Button: React.FC<ButtonProps> = ({ text, onClick }) => <button
onClick={onClick}>{text}</button>;
```

## 18. Redux

```
import { createStore } from 'redux';
const store = createStore((state={count:0}, action) => {
  switch(action.type){ case 'INC': return {count: state.count+1}; default:
return state; }
});

import { Provider, useSelector, useDispatch } from 'react-redux';
<Provider store={store}><App /></Provider>

const count = useSelector(state => state.count);
const dispatch = useDispatch();
dispatch({type:'INC'});
```

## 19. Advanced Patterns

- **Render Props**

```
<DataProvider render={data => <Child data={data}/>} />
```

- **Higher-Order Components (HOC)**

```
const withAuth = Component => props => isLoggedIn ? <Component {...props}/>
: <Login />;
```

- **Compound Components**

```
<Tabs><Tabs.List>...</Tabs.List><Tabs.Panel>...</Tabs.Panel></Tabs>
```

## 20. Animations

- **Framer Motion**

```
npm install framer-motion
```

```
import { motion } from 'framer-motion';
<motion.div animate={{ x: 100 }} transition={{ duration: 1
}}>Move</motion.div>
```

## 21. Testing

- **Jest + React Testing Library**

```
npm install --save-dev jest @testing-library/react
```

```
import { render, screen } from '@testing-library/react';
render(<Button text="Click"/>);
expect(screen.getByText("Click")).toBeInTheDocument();
```

## 22. SSR & Next.js Concepts

- React can be **server-rendered** using Next.js
- Supports **getServerSideProps, getStaticProps**
- Routing is **file-based**
- Supports **API routes**

## 23. Quick Tips

- Components **start with uppercase**
- Always **cleanup effects**
- Use **unique keys** for lists
- Prefer **functional components + hooks**
- Memoize expensive computations
- Keep **state minimal**

Here's what it **includes**:

### Setup & Project

- Create React App (CRA)
- Vite setup
- Dev server commands
- File structures (CRA & Vite)

### Components

- Functional, class, arrow functions
- Props & state
- Controlled & uncontrolled forms

### JSX

- Expressions, fragments, conditional rendering
- Lists with keys

### Lifecycle & Side Effects

- `useEffect` (mount, update, cleanup)

### Event Handling

- `onClick`, `onChange`, `onSubmit`

### Routing

- React Router v6 (BrowserRouter, Routes, Route, Link)

### State Management

- `useState`, `useReducer`
- Context API (`useContext`)
- Redux basics (`store`, `actions`, `useSelector`, `useDispatch`)

### Hooks

- `useState`, `useEffect`, `useContext`, `useReducer`
- `useRef`, `useMemo`, `useCallback`, `useLayoutEffect`, `useImperativeHandle`, `useDebugValue`
- Custom hooks

### Data Fetching

- `fetch`, `axios`

### Performance

- Memoization (`React.memo`, `useMemo`, `useCallback`)
- Lazy loading + `Suspense`

## Advanced Features

- Error boundaries
- Portals
- Advanced patterns (Render props, HOC, Compound components)
- Animations (Framer Motion)
- Testing (Jest + React Testing Library)
- TypeScript basics (props typing, functional components)
- SSR concepts (Next.js)

## Quick Tips

- Use uppercase for components
- Always clean up effects
- Unique keys for lists
- Prefer functional components + hooks
- Keep state minimal
- Memoize expensive computations

**What it doesn't include in detail:**

- Full Next.js tutorials or advanced SSR patterns
- Enterprise-scale architecture patterns
- Some niche animation libraries outside Framer Motion
- Full integration with other state management libraries (Zustand, Recoil, MobX)

But for **practical React development, full-stack apps, and advanced features**, this cheatsheet is **comprehensive and complete**.

**My Suggestions:**

1. For a beginner → start with **Vite + functional components + hooks**.
2. Gradually learn **Context & Redux** for state management.
3. Optimize performance using **memoization** when needed, not everywhere.
4. Explore **Next.js** once comfortable with React for SSR.
5. Use this cheatsheet as a reference while building **real projects**, because hands-on practice solidifies everything.