Rules of React

Just as different programming languages have their own ways of expressing concepts, React has its own idioms — or rules — for how to express patterns in a way that is easy to understand and yields high-quality applications.

- Components and Hooks must be pure
- React calls Components and Hooks
- Rules of Hooks

Note

To learn more about expressing UIs with React, we recommend reading Thinking in React.

This section describes the rules you need to follow to write idiomatic React code. Writing idiomatic React code can help you write well organized, safe, and composable applications. These properties make your app more resilient to changes and makes it easier to work with other developers, libraries, and tools.

These rules are known as the **Rules of React**. They are rules – and not just guidelines – in the sense that if they are broken, your app likely has bugs. Your code also becomes unidiomatic and harder to understand and reason about.

We strongly recommend using Strict Mode alongside React's ESLint plugin to help your codebase follow the Rules of React. By following the Rules of React, you'll be able to find and address these bugs and keep your application maintainable.

Components and Hooks must be pure

Purity in Components and Hooks is a key rule of React that makes your app predictable, easy to debug, and allows React to automatically optimize your code.

- Components must be idempotent – React components are assumed to always return the same output with respect to their inputs – props, state, and context.
- Side effects must run outside of render – Side effects should not run in render, as React can render components multiple times to create the best possible user experience.
- Props and state are immutable – A component's props and state are immutable snapshots with respect to a single render. Never mutate them directly.
- Return values and arguments to Hooks are immutable – Once values are passed to a Hook, you should not modify them. Like props in JSX, values become immutable when passed to a Hook.
- Values are immutable after being passed to JSX – Don't mutate values after they've been used in JSX. Move the mutation before the JSX is created.

React calls Components and Hooks

React is responsible for rendering components and hooks when necessary to optimize the user experience. It is declarative: you tell React what to render in your component's logic, and React will figure out how best to display it to your user.

- Never call component functions directly – Components should only be used in JSX. Don't call them as regular functions.
- Never pass around hooks as regular values – Hooks should only be called inside of components. Never pass it around as a regular value.

Rules of Hooks

Hooks are defined using JavaScript functions, but they represent a special type of reusable UI logic with restrictions on where they can be called. You need to follow the Rules of Hooks when using them.

- Only call Hooks at the top level – Don't call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function, before any early returns.
- Only call Hooks from React functions – Don't call Hooks from regular JavaScript functions.

NEXT
Components and Hooks must be pure

Learn React

Quick Start

Installation

API Reference

React APIs

React DOM APIs

Community

More