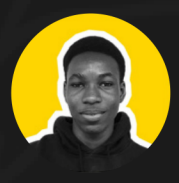

David Oga

react/tips 🔥

Understanding hooks in React

Save post

Swipe to know >>>

React hooks are a way to manage state and side effects in functional components in React. They make it easier to add state and functionality to your components without needing to convert them into class components.

useState

This hook allows you to add state to your functional components. It takes an initial value and returns an array with two elements: the current state and a function to update that state.

this holds the current state value

function that updates the state value

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

creates the state and sets the the initial value to 0

setCount(3) ⟶ updates the state value to 3

useEffect

This hook is used for handling side effects, like data fetching, DOM manipulation, and more. It takes two arguments: a function that contains your side effect code, and an array of dependencies that specify when the effect should run. If no dependency is specified, the side effect code runs only once when it is mounted initially.

```
useEffect(() => {
 // Do something when dependencies change
}, [dependencies]);
```

useContext

The useContext hook allows you to access a context (a global data store that you can use to share information across various parts of your application without the need to pass props through every level of the component tree). You pass a context object created using React.createContext and receive the current context value.

```
const value = useContext(MyContext);
```

useReducer

This hook is an alternative to useState when you have complex state logic. It takes a reducer function and an initial state or data, and returns the current state and a dispatch function to update it.

To change your state, you dispatch actions to the dispatch function. Each action tells the reducer what change to make.

```
const [state, dispatch] = useReducer(reducer, initialData);
```

This function describes how your state can change based on different actions.

Swipe >>>

useMemo

useMemo helps optimize your application by memoizing (caching) the result of a function. It's particularly useful for preventing unnecessary recalculations of values, which can improve performance in your app.

this holds the memoized or cached value returned by squareNumber

```
const squareNumber = useMemo(() => {
  return number * number;
}, [number]);
```

squareNumber updates only when number changes it reuses the previous result otherwise.

Swipe >>>

useCallback

While useMemo memoizes values, useCallback helps you optimize your application by memoizing (caching) functions. It's especially useful for preventing unnecessary function recreations, which can improve performance.

```
const handleClick = useCallback(() => {
// Your click handling logic here },
[dependencies]);
```

By using useCallback, you ensure that the handleClick function is only recreated if the dependencies change.

Swipe >>>

custom hooks

You can create your custom hooks to share stateful logic between components. They typically start with "use" and can encapsulate any hook, or combinations of hooks, to provide a specific behavior or functionality.

What other topics in react should i talk about?

Let me know in the comments!

↓

Save post Share post

Follow for more content like this!