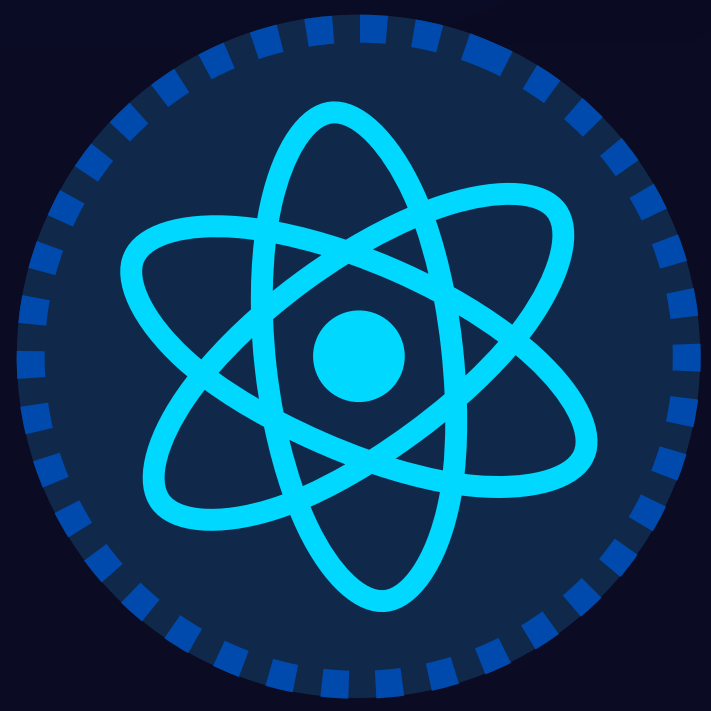
@saadjamilakhtar

LIFECYCLE METHODS & useEffect HOOK

SWIPE



01

Traditional Lifecycle Methods

React class components offer key lifecycle methods like **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** etc. for managing component behavior throughout their lifecycle.

@saadjamilakhtar

02

Enter the useEffect Hook

Functional components introduced the **useEffect** hook, providing a more elegant way to handle side effects and to replicate lifecycle behavior.

@saadjamilakhtar

03

Replicating componentDidMount

To mimic **componentDidMount**, use useEffect with an empty dependency array. Your code runs once after the initial render

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

function MyComponent() {
  useEffect(() => {
    // Code here runs after the initial render
  }, []);

  return <div>My Component</div>;
}
```

@saadjamilakhtar

04

Replicating componentDidUpdate

To replicate **componentDidUpdate**, specify the state or props you want to watch in the dependency array. The code inside useEffect runs when those dependencies change.

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

function MyComponent({ data }) {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // Code here runs after the initial render
  }, [count]);

  return <div>My Component</div>;
}
```

05

Replicating componentWillUnmount

Clean up resources when your component unmounts by returning a cleanup function inside useEffect. This function runs when the component is about to unmount.

```
function MyComponent({ data }) {

  useEffect(() => {
    // Code here runs after the initial render
    return () => {
      // Cleanup code, runs when the component unmounts
    };
  }, []);

  return <div>My Component</div>;
}
```

06

Replicating componentWillReceiveProps

To replicate **componentWillReceiveProps,** compare the previous and current props within useEffect. This allows you to react to prop changes.

```
function MyComponent({ data }) {
   const prePropsRef = useRef()  ;
   useEffect(() => {
      if (prevPropsRef.current) {
        // Compare previous and current props here
      }
      prevPropsRef.current = data;
   }, [data]);

   return <div>My Component</div>;
}
```

07

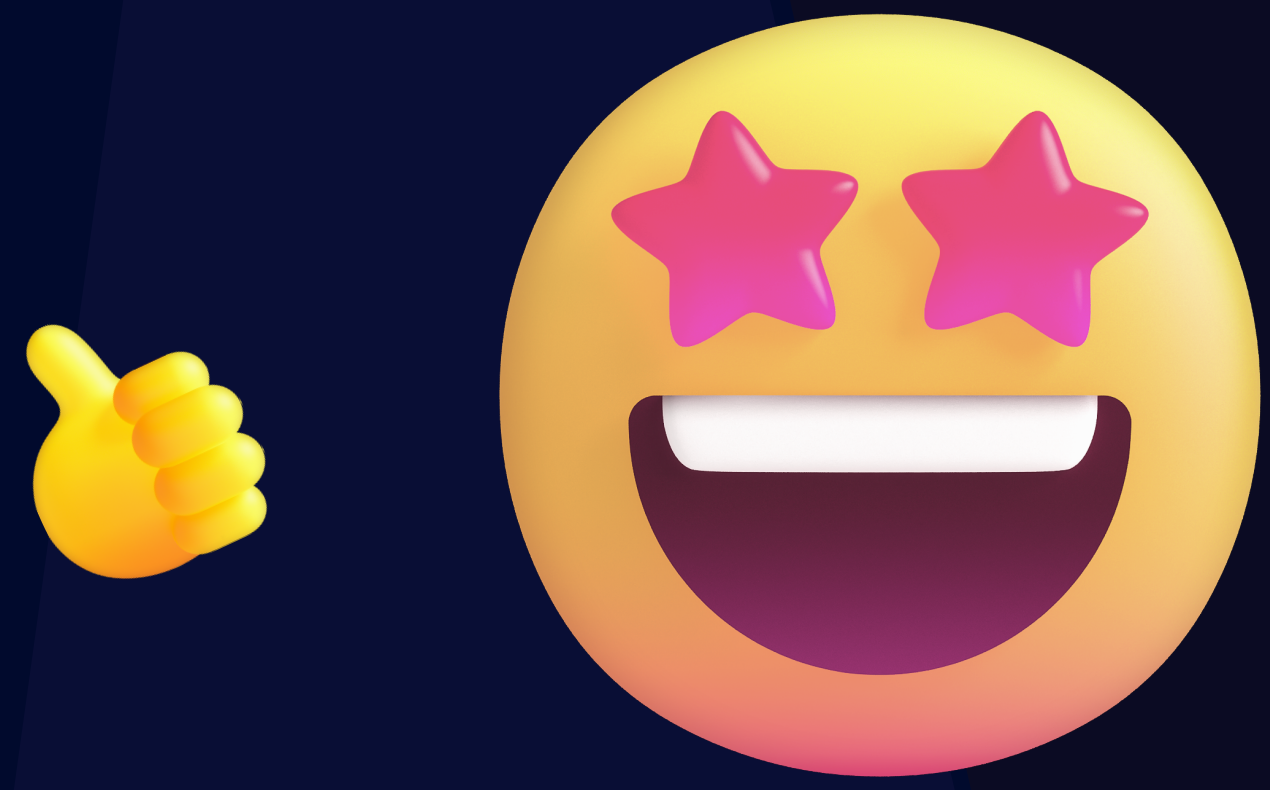
Replicating shouldComponentUpdate

To replicate shouldComponentUpdate, use useEffect to control whether a component re-renders based on prop or state changes.

```
function MyComponent({ data }) {
   const [shouldUpdate, setShouldUpdate] = useState(true);

   useEffect(() => {
      if (shouldUpdate) {
       // Render component based on prop or state changes
      }
   }, [data, shouldUpdate]);

   return <div>My Component</div>;
}
```



Was this post helpful ?
Follow for more

@saadjamilakhtar