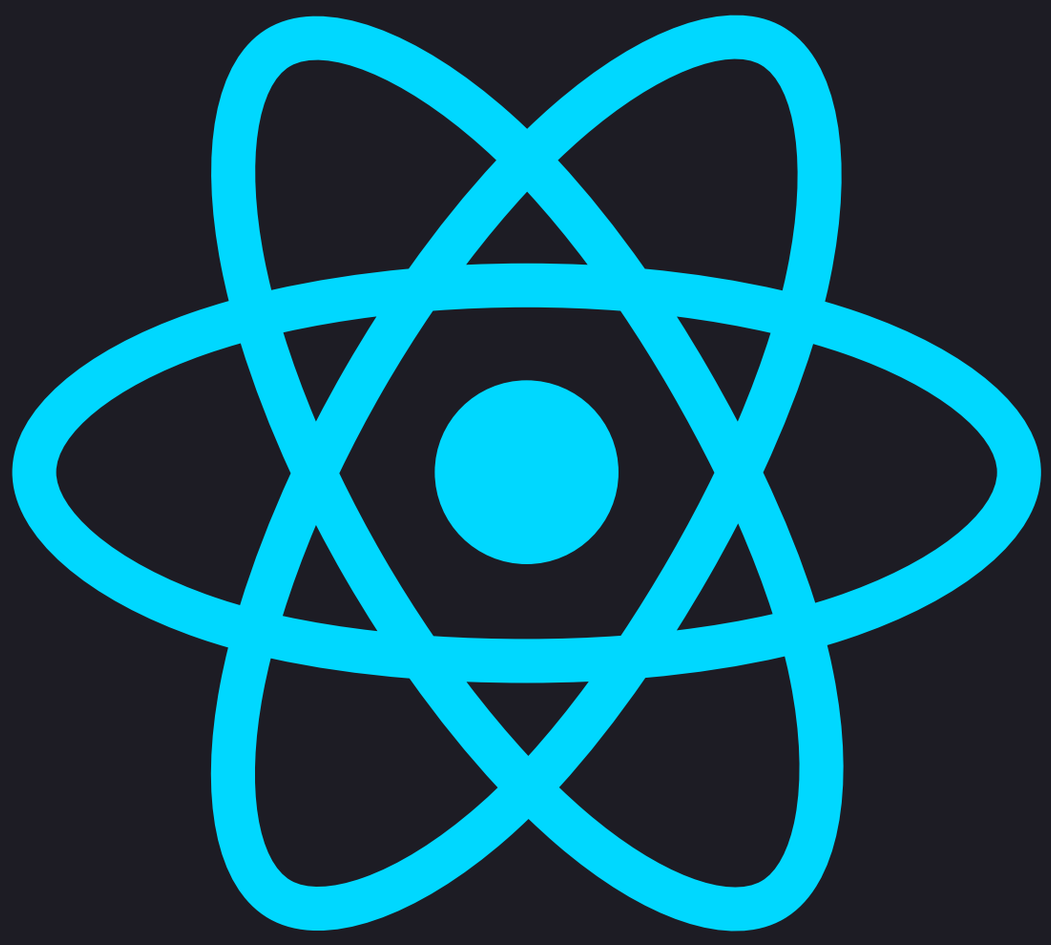
Motivation Tree



React Hooks Interview Questions

Index

Motivation Tree
www.motivationtree.com

use State

Motivation Tree
www.motivationtree.com

use state hook return an array with two elements
1- Current State
2- State Setter function

```
import React, { useState } from "react";

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

 const incrementByFive = () => {
 for (let i = 0; i < 5; i++) {
setCount((prevCount) => prevCount + 1);
}
};
return (
<div>
<h1>{count}</h1>
<button onClick={() => setCount(initialCount)}>Reset Count</button>
<button onClick={() => setCount(count + 1)}> increment count</button>
<button onClick={() => setCount(count - 1)}> Decrement count</button>
<button onClick={incrementByFive}>Increment by Five</button>
 </div>
 );
}

export default HookCounterTwo;
```

Gotop

UseEffect

Motivation Tree
www.motivationtree.com

useEffect lets us express different kinds of side effects after a component renders.

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

function HookCounterTwo() {
 const [count, setCount] = useState(0);
 const [name, setName] = useState("");

 useEffect(() => {
 console.log("useEffect- updating document tittle");
 document.title = `you have clicked ${count}`;
 }, [count]);

 return (
 <div>
 <h1>{count}</h1>
 <input
 type="text"
 value={name}
 onChange={(e) => setName(e.target.value)}
 />
 <button onClick={() => setCount(count + 1)}>inc Count</button>
 </div>
 );
}

export default HookCounterTwo;
```

Gotop

useContext

Motivation Tree
www.motivationtree.com

It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

```
import { useState } from "react";
import ReactDOM from "react-dom/client";
function Component1()
{const [user, setUser] = useState("Jesse Hall");

return (
<>
<h1>{`Hello ${user}!`}</h1>
<Component2 user={user} />
</>);}
function Component2({ user })
{return (
<>
<h1>Component 2</h1>
<Component3 user={user} />
</>
);}
function Component3({ user }) {
return (
<>
<h1>Component 3</h1>
<Component4 user={user} >
</>);}
function Component4({ user }) {return (
<>
<h1>Component 4</h1>
<Component5 user={user} />
</>);}function Component5({ user }) {return (<>
<h1>Component 5</h1>
<h2>{`Hello ${user} again!`}</h2>
</>);}
```

Problem

Gotop

```
const UserContext = createContext();

function Component1() {
  const [user, setUser] = useState("Jesse Hall");

  return (
    <UserContext.Provider value={user}>
      <h1>{`Hello ${user}!`}</h1>
      <Component2 />
    </UserContext.Provider>
  );
}

function Component2() {
  return (
    <>
      <h1>Component 2</h1>
      <Component3 />
    </>
  );
}

function Component3() {
  return (
    <>
      <h1>Component 3</h1>
      <Component4 />
    </>
  );
}

function Component4() {
  return (
    <>
      <h1>Component 4</h1>
      <Component5 />
    </>
  );
```


Solution using
useContext

useReducer

Motivation Tree
www.motivationtree.com

- used for sate management
- Alternative of useState
- usestate is built using usereducer
- useReducer(reducer, initialState)

```
import React, { useReducer, useState } from "react";

const initState = 0;

const reducer = (state, act) => {
 switch (act) {
 case "increment":
 return state + 1;
 case "decrement":
 return state - 1;
 case "reset":
 return initState;
 default:
 return state;
 }
};

function RedCounter() {
 const [count, dispatch] = useReducer(reducer, initState);
 return (
 <div>
 <div>{count}</div>
 <button onClick={() => dispatch("increment")}> Increment {count}</button>
 <button onClick={() => dispatch("decrement")}> decrement {count}</button>
 <button onClick={() => dispatch("reset")}> reset {count}</button>
 </div>
 );
}

export default RedCounter;
```

Gotop

UseMemo

Motivation Tree
www.motivationtree.com

- The React useMemo Hook returns a memoized value.
- The useMemo Hook only runs when one of its dependencies update.
- This can improve performance.

```
import React, { useState, useMemo } from "react";
function CounterMemo() {
  const [counteOne, setCounterOne] = useState(0);
  const [counteTwo, setCounterTwo] = useState(0);

  const incrementOne = () => {
    setCounterOne(counteOne + 1);
  };
  const incrementTwo = () => {
    setCounterTwo(counteTwo + 1);
  };

  const isEven = useMemo(() => {
    let i = 0;
    let j = 0;
    while (i < 9000 && j < 9000022) i = j++;
    return counteOne % 2 === 0;
  }, [counteOne]);

  return (
    <div>
      <button onClick={incrementOne}>Increment one{counteOne}</button>
      <br />
      <span>{isEven ? "even" : "odd"}</span>
      <br />
      <button onClick={incrementTwo}>Increment two{counteTwo}</button>
    </div>
  );
}

export default CounterMemo;
```

Gotop

UseCallback

Motivation Tree
www.motivationtree.com

- The React useCallback Hook returns a memoized callback function.
- This allows us to isolate resource intensive functions so that they will not automatically run on every render.
- The useCallback Hook only runs when one of its dependencies update.

useCallback

```
const addTodo = useCallback(() => {
   setTodos((t) => [...t, "New Todo"]);
 }, [todos]);
```

useMemo

```
const isEven = useMemo(() => {
   let i = 0;
   let j = 0;
   while (i < 9000 && j < 9000022) i = j++;
   return counteOne % 2 === 0;
 }, [counteOne]);
```

Gotop

useRef

Motivation Tree
www.motivationtree.com

- The useRef Hook allows you to persist values between renders.
- It can be used to store a mutable value that does not cause a re-render when updated.
- It can be used to access a DOM element directly.
- The useRef Hook can also be used to keep track of previous state values.

Use useRef to focus the input:

```
import { useRef } from "react";
import ReactDOM from "react-dom/client";

function App() {
const inputElement = useRef();

const focusInput = () => {
 inputElement.current.focus();
};

return (
 <>
  <input type="text" ref={inputElement} />
  <button onClick={focusInput}>Focus Input</button>
 </>
 );
}
```

www.motivationtree.com