Learn with Rabbil

REACT HOOK

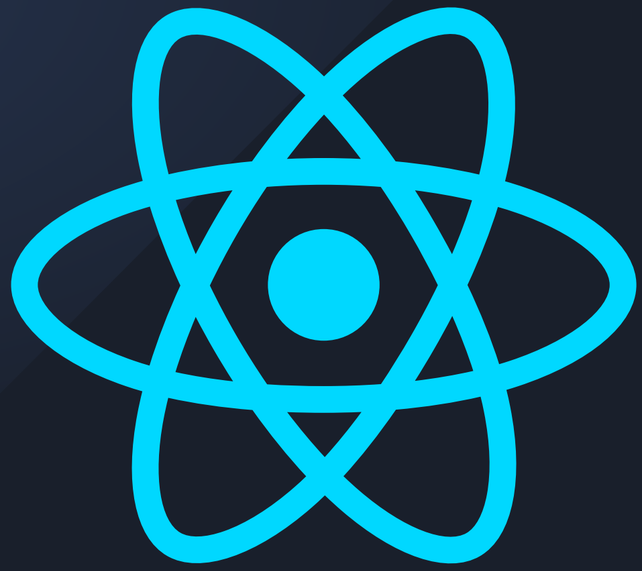
01. **React Hook is a feature in the React library** that allows developers to use state and other React features in functional components, which were previously only available in class components.

02. It was introduced in **React version 16.8**.

03. Hooks can be used to **manage state, handle side effects, and access lifecycle methods** in functional components.

04. There are several built-in Hooks provided by React, such as **useState**, **useEffect**, useContext, and **useRef**.

05. React Hooks have greatly simplified the development process in React and have made i**t easier to write reusable and composable code.**

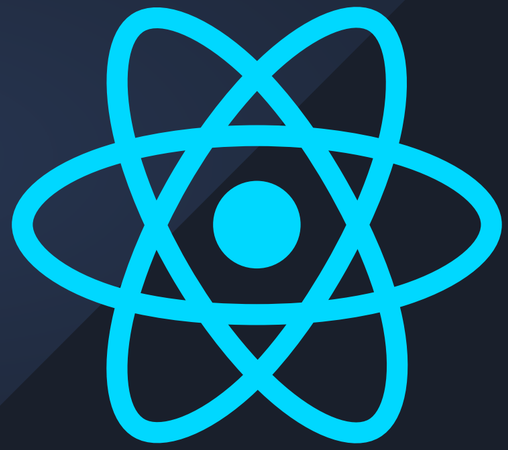


Learn with Rabbil

REACT HOOK

useRef() Method

01. The useRef Hook allows you to **persist values between renders**
02. It can be **used to store a mutable value** that does not cause a re-render when updated.
03. It can be used to access a **DOM element** directly.



Learn with Rabbil

REACT HOOK

useRef() Method Changing HTML Elements

index.js

```
1   import React, {useRef} from 'react';
2   const Index = () => {
3       let demoRef=useRef();
4       const Change=()=>{
5          // demoRef.innerHTML="<h1>Learn</h1>"
6           //demoRef.innerText="<h1>Learn</h1>"
7       }
8       return (
9           <div>
10              <p ref={(p)=>demoRef=p}></p>
11              <button onClick={()=>Change()}>Submit</button>
12          </div>
13      );
14  };
15  export default Index;
```

REACT HOOK

useRef() Method Working With Attributes

index.js

```
1   import React, {useRef} from 'react';
2   const Index = () => {
3       let demoRef=useRef(null);
4       const Change=()=>{
5          demoRef.current.src="https://placehold.co/600x400/orange/white"
6          demoRef.current.setAttribute("height", "200px")
7          demoRef.current.setAttribute("width", "200px")
8       }
9       return (
10          <div>
11              <img src="https://placehold.co/600x400/000000/FFF" ref={demoRef}></img>
12              <button onClick={()=>Change()}>Submit</button>
13          </div>
14      );
15  };
16  export default Index;
```

Learn with Rabbil

REACT HOOK

useRef() Method Working

With Input Element

index.js

```
1   import React, {useRef} from 'react';
2   const Index = () => {
3       let demoRef=useRef();
4       const Change=()=>{
5          demoRef.focus();
6          let inputValue= demoRef.value;
7          alert(inputValue);
8          demoRef.value="New Value"
9       }
10      return (
11          <div>
12              <input ref={(input)=>demoRef=input}/>
13              <button onClick={()=>Change()}>Submit</button>
14          </div>
15      );
16  };
17  export default Index;
```

REACT HOOK

useRef() Method Working With Add Remove CSS Class

index.js

```
1   import React, {useRef} from 'react';
2   const Index = () => {
3       let demoRef=useRef();
4       const Change=()=>{
5           demoRef.classList.add('text-primary')
6           demoRef.classList.remove('text-success')
7       }
8       return (
9           <div>
10              <h1 className="text-success" ref={(h1)=>demoRef=h1}>Learn Next JS</h1>
11              <button onClick={()=>Change()}>Change</button>
12          </div>
13      );
14  };
15  export default Index;
```

REACT HOOK

useRef() Method Create Persisted Mutable Values

index.js

```
1   import React, {useRef} from 'react';
2   const Index = () => {
3       let demoRef=useRef(0);
4       const Change=()=>{
5           demoRef.current++
6           console.log(`Clicked ${demoRef.current} times`);
7       }
8       return (
9           <div>
10              <h1></h1>
11              <button onClick={()=>Change()}>Change</button>
12          </div>
13      );
14  };
15  export default Index;
```

Learn with Rabbil

REACT HOOK

useRef() Caching expensive computations

01. When you need to **re-use the result multiple times** within a component, but you **don't want to re-compute** the value **every time the component renders.**

02. Let's say you have a component that **fetches data from an API.** The API call might take a few seconds to complete, so you **don't want to re-fetch the data every time the component renders.** Instead, you can **use useRef() to cache the result** of the API call

REACT HOOK

useRef() Caching expensive computations

App.jsx

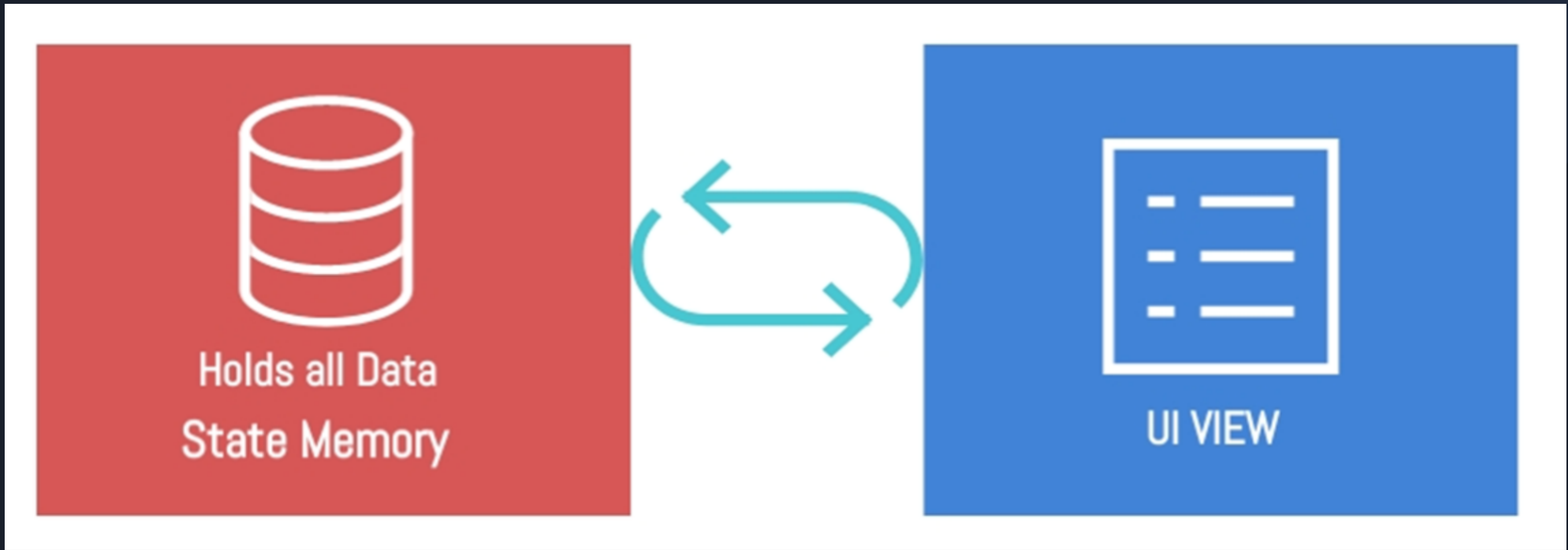
```jsx
const App = () => {
    const expensiveResultRef = useRef(null);
    const myDiv = useRef(null);

    const fetchData = async () => {
        const response = await fetch('https://dummyjson.com/products');
        expensiveResultRef.current = await response.json();
    }
    const ShowData = () => {
        myDiv.current.innerHTML = JSON.stringify(expensiveResultRef.current);
    }
    return (
        <div>
            <div ref={myDiv}></div>;
            <button onClick={ShowData}>Show Data</button>
            <button onClick={fetchData}>Call API</button>
        </div>
    );
};
```

Learn with Rabbil

UNDER STANDING STATE

- In React, state refers to an object that holds data of your component
- When data changed component refresh automatically to reflect the changes



Learn with Rabbil

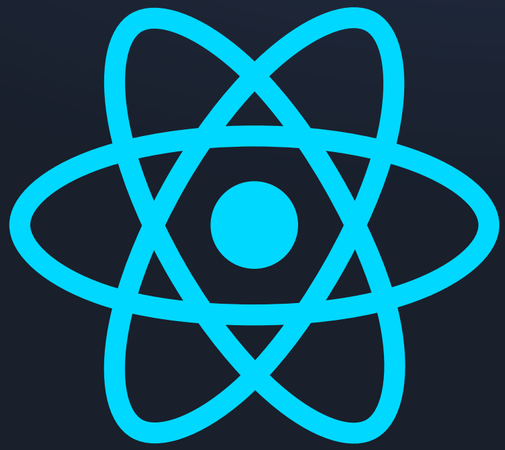
REACT HOOK

useState() Method

01. The state is a built-in React object that is used to contain data or information about the component.

02. A state can be modified based on user action or network changes

03. Every time the state of an object changes, React re-renders the component to the browser



Learn with Rabbil

REACT HOOK

useState() Method Counter Example

index.js

```
1   import React, {useState} from 'react';
2   const Index = () => {
3       const [number,setNumber]=useState(0);
4       return (
5           <div>
6               <h1>{number}</h1>
7               <button onClick={()=>setNumber(number+1)}>Click</button>
8           </div>
9       );
10  };
11  export default Index;
```

Learn with Rabbil

REACT HOOK

useState() Method

Working With Object

App.jsx

```jsx
const App = () => {
    const [myObject, setMyObject] = useState({
        key1: 'value1',
        key2: 'value2',
        key3: 'value3'
    });

    const updateObject = () => {
        setMyObject(prevObject => ({
            ...prevObject,
            kye1: 'new value'
        }));
    };
    return (
        <div>
            <div ref={myObject.key1}></div>;
            <button onClick={updateObject}>Change</button>
        </div>
    );
};
```

REACT HOOK

useState() Method Todo Example

index.js

```
1  import React, {useState} from 'react';
2  const Index = () => {
3      let [list,setList]=useState([]);
4      let [item,setItem]=useState("");
5      const AddToList=()=>{
6          list.push(item)
7          setList([...list]);
8      }
9      const RemoveFromList=(index)=>{
10         list.splice(index,1)
11         setList([...list]);
12     }
13     return (
14         <div>
15             <input onChange={(e)=>setItem(e.target.value)}/>
16             <button onClick={()=>AddToList()}>Click</button>
17             <table>
18                     <tbody>
19                     {
20                         list.length!==0?(
21                             list.map((element,i)=>{
22                                 return(
23                                     <tr key={i.toString()}>
24                                         <td>{element}</td>
25                                         <td><button onClick={()=>{RemoveFromList(i)}}>Remove</button></td>
26                                     </tr>
27                                 )
28                             })
29                         ):(<tr></tr>)
30                     }
31                     </tbody>
32             </table>
33         </div>
34     );
35 };
36 export default Index;
```

Learn with Rabbil

WHY WE ARE USING

Spread Operator In State Object

Step: 01 In React, the **state object is intended to be immutable**

Step: 02 React encourages developers to follow the **principle of immutability** when working with state.

Step: 03 Which means that you should not directly **mutate the state object.** Instead, you **create a new object with the desired changes** and **update the state with the new object.**

Step: 04 By following immutability, React can efficiently c**ompare previous and current state objects** to determine **if a re-render is necessary**

Step: 05 When you **mutate the state object directly**, React may not detect the changes correctly, **leading to unexpected behavior**.

Step: 06 Using the spread operator technique **we are creating new object** that maintains the previous state's values while making the necessary modifications.

Step: 07 This ensures that the state **object remains immutable**

Step: 08 So, remember to always treat the state object as immutable and create a new object when updating state values in React.

REACT HOOK

useState() Method Manage Form

index.js

```
1   import React, {useState} from 'react';
2   const Index = () => {
3       let [FormValue,SetFormValue]=useState({fname:"", lname:"", city:"", gender:""})
4       const InputOnChange=(InputName,InputValue)=>{
5           SetFormValue(FormValue => ({
6               ...FormValue,
7               [InputName]: InputValue
8           }));
9       }
10      const FormSubmit=(e)=>{
11          e.preventDefault();
12          alert(JSON.stringify(FormValue))
13      }
14      return (
15          <form onSubmit={FormSubmit}>
16               <input placeholder="First Name" value={FormValue.fname} onChange={(e)=>InputOnChange('fname',e.target.value)} />
17               <input placeholder="Last Name"  value={FormValue.lname} onChange={(e)=>InputOnChange('lname',e.target.value)} />
18               <select value={FormValue.city} onChange={(e)=>InputOnChange('city',e.target.value)} >
19                   <option value="">Select City</option>
20                   <option value="Dhaka">Dhaka</option>
21                   <option value="Rangpur">Rangpur</option>
22               </select>
23              <input checked={FormValue.gender === "Male"} onChange={(e)=>{InputOnChange('gender','Male')} } type="radio" name="gender"/> Male
24              <input checked={FormValue.gender === "Female"} onChange={(e)=>{InputOnChange('gender','Female')} } type="radio" value="Female" name="gender"/> Female
25              <br/>
26              <button type="submit">Submit</button>
27          </form>
28      );
29  };
30  export default Index;
```

Learn with Rabbil

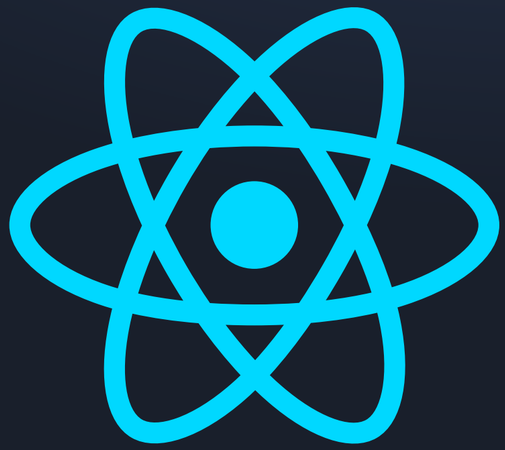
REACT HOOK

useEffect() Method

01. The useEffect Hook allows you to perform side effects in your components.

02. useEffect accepts two arguments. The second argument is optional dependency array

03. Mostly used for Fetching data



REACT HOOK

Learn with Rabbil

01. **First Argument:** executed after the component has rendered and the DOM has been updated. This function can perform various side effects such as fetching data, subscribing to events, or manipulating the DOM.

App.jsx

```
6       useEffect(() => {
7           // This effect runs only once on component mount
8         }, []);
```

02. **Second Argument:** React will re-run the effect only if any of the values in the dependencies array have changed since the last render. If you want the effect to run only once, you can pass an empty array as the dependencies, indicating that the effect has no dependencies.

App.jsx

```
5       const [count, setCount] = useState(0);
6
7       useEffect(() => {
8         // This effect runs whenever the 'count' state value changes
9         console.log("Count changed:", count);
10      }, [count]);
```

Learn with Rabbil

REACT HOOK

useEffect() Method Fetch Example

index.js

```
1   import React, {useEffect, useState} from 'react';
2
3   const Index = () => {
4       const [Data,SetData]=useState([]);
5
6       useEffect(()=>{
7           fetch('https://dummyjson.com/products/1')
8               .then(res => res.json())
9               .then(json => SetData(json))
10      },[])
11
12      return (
13          <div>
14              {JSON.stringify(Data)}
15          </div>
16      );
17  };
18
19  export default Index;
```

Learn with Rabbil

REACT HOOK

useEffect() Method Fetch

Async Await Example

index.js

```
1   import React, {useEffect, useState} from 'react';
2
3   const Index = () => {
4       const [Data,SetData]=useState([]);
5
6       useEffect(()=>{
7
8           (async () => {
9              let response= await  fetch('https://dummyjson.com/products/1')
10             let result = await response.json();
11             SetData(result);
12          })()
13
14      },[])
15
16      return (
17          <div>
18              {JSON.stringify(Data)}
19          </div>
20      );
21  };
22
23  export default Index;
```

Learn with Rabbil

REACT ROUTER DOM

```
npm i react-router-dom
```

01. The react-router package is the heart of React Router and provides all the core functionality for both react-router-dom and react-router-native

02. The react-router-dom package contains bindings for using React Router in web applications.

03. The react-router-native package contains bindings for using React Router in React Native applications

Learn with Rabbil

REACT ROUTER DOM

01. **<BrowserRouter>** stores the **current location in the browser's address** bar using clean URLs and navigates using the **browser's built-in history stack.**

02. **<Routes>** **renders a route exclusively** as it displays the first child route that matches the current URL

03. **<Route>** is the child component that renders a **specific UI component when the URL matches the specified path.**

04. **<Link>** is an element that lets the **user navigate to another page by clicking** or tapping on it

Menu.jsx

```jsx
const Menu = () => {
    return (
        <div>
            <ul>
                <li><Link to="/">Page1</Link></li>
                <li><Link to="/page2">Page2</Link></li>
                <li><Link to="/page3">Page3</Link></li>
            </ul>
        </div>
    );
};
```

App.jsx

```jsx
const App = () => {
    return (
        <div>
            <BrowserRouter>
                <Routes>
                    <Route path="/" element={<Page1/>}/>
                    <Route path="/page2" element={<Page2/>}/>
                    <Route path="/page3" element={<Page3/>}/>
                    <Route path="*" element={<NotFound/>}/>
                </Routes>
            </BrowserRouter>
        </div>
    );
};
```

Learn with Rabbil

REACT ROUTER DOM

01. **<HashRouter>** is for **use in web browsers** when the URL should not (or cannot) be sent to the server for some reason. This may happen in some shared hosting scenarios where you **do not have full control over the server**

02. <Routes> renders a route exclusively as it displays the first child route that matches the current URL

03. <Route> is the child component that renders a specific UI component when the URL matches the specified path.

04. <NavLink> is a special kind of <Link> that knows whether or not it is "active" or "pending"

App.jsx

```jsx
const App = () => {
    return (
        <div>
            <HashRouter>
                <Routes>
                    <Route path="/" element={<Page1/>}/>
                    <Route path="/page2" element={<Page2/>}/>
                    <Route path="/page3" element={<Page3/>}/>
                    <Route path="*" element={<NotFound/>}/>
                </Routes>
            </HashRouter>
        </div>
    );
};
```

Menu.jsx

```jsx
const Menu = () => {
    return (
        <div>
            <ul>
                <li>
                    <NavLink className={({ isActive, isPending }) =>
                            isPending ? "pending" : isActive ? "active" : ""
                    } to="/">Page1</NavLink>
                </li>
                <li><NavLink to="/page2">Page2</NavLink></li>
                <li><NavLink to="/page3">Page3</NavLink></li>
            </ul>
        </div>
    );
};
```

Learn with Rabbil

01. **URL appearance:** HashRouter will include a **hash (#) i**n the URL, while BrowserRouter will provide c**leaner URLs without the hash.**

02. **Compatibility:** HashRouter **works on all servers** since the hash portion of the URL is not sent to the server. **BrowserRouter relies on the server to handle routing** for all URLs, so **it requires server configuration** to redirect requests to the main React app.

03. **History API:** HashRouter **does not use the HTML5 History API** and instead relies on the hash part of the URL to handle routing. **BrowserRouter** uses the HTML5 History API for navigation

04. **Server-side rendering (SSR):** BrowserRouter may require **additional server-side configuration** to handle routing correctly, while HashRouter can be used more easily for server-side rendering.

Learn with Rabbil

REACT ROUTER DOM

Passing Parameter

```jsx
<BrowserRouter>
    <Routes>
        <Route path="/" element={<Page1/>}/>
        <Route path="/page2/:userId" element={<Page2/>}/>
        <Route path="/page3" element={<Page3/>}/>
        <Route path="*" element={<NotFound/>}/>
    </Routes>
</BrowserRouter>
```

```jsx
<li><Link to="/">Page1</Link></li>
<li><Link to="/page2/123">Page2</Link></li>
<li><Link to="/page3">Page3</Link></li>
```

Page2.jsx

```jsx
const Page2 = () => {
    let { userId } = useParams();
    return (
        <div>
            <Menu/>
            <h1>Page 02</h1>
            <h1>{userId}</h1>
        </div>
    );
};
```

BASIC BLOG PRACTICE PROJECT

BASIC BLOG PRACTICE PROJECT