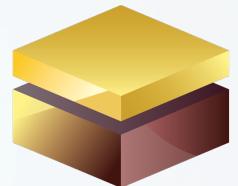
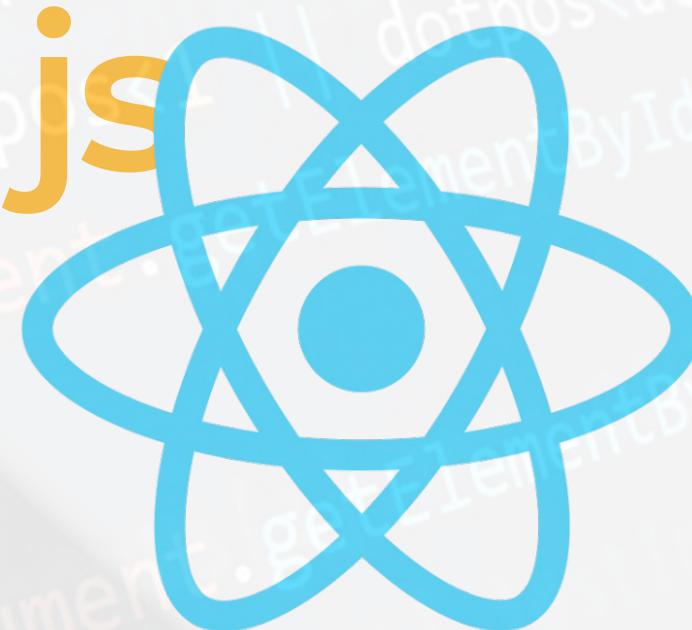
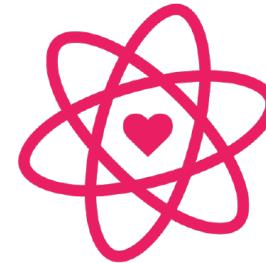


# React



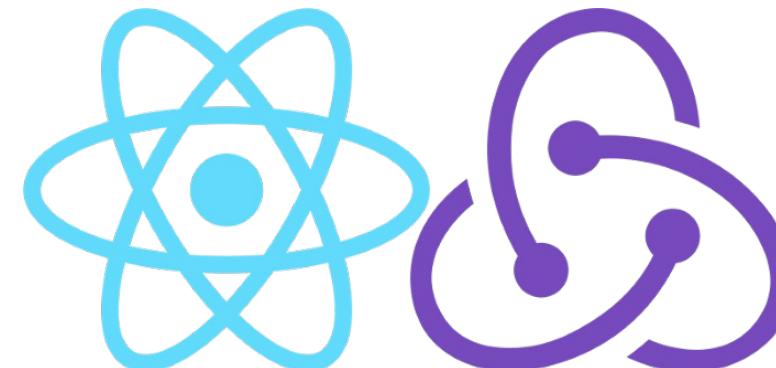
**CYBERLEARN**  
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Hướng dẫn : Trương Tấn Khải



# React JS

Functional Component  
React hooks



## ☐ React hooks là gì? (React version >= 16.8)

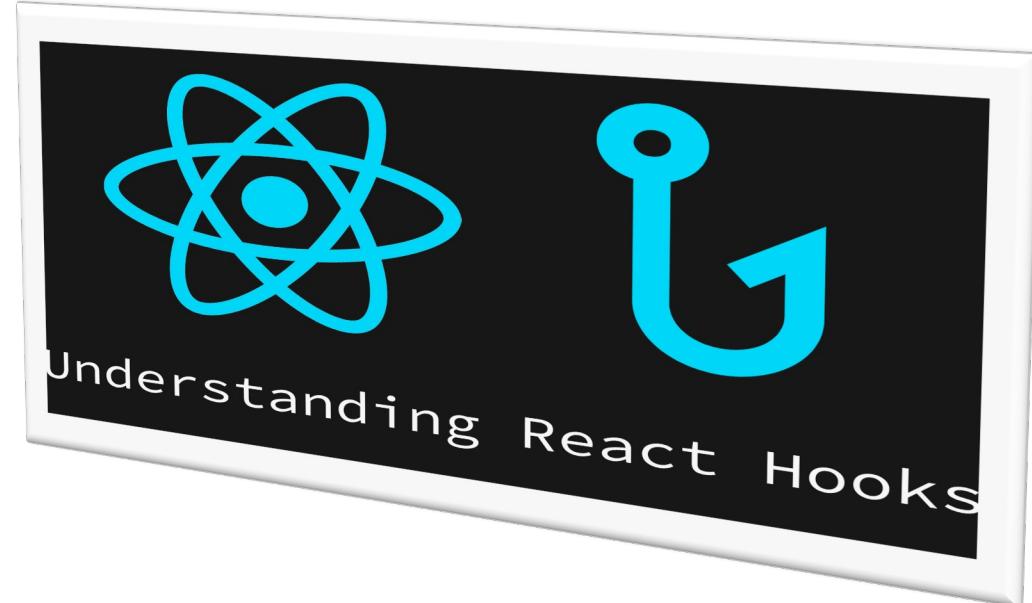
- Như ta đã biết ở các bài trước, state và lifecycle chỉ có thể sử dụng được trong class component
- Hook ra đời, cho phép ta có thể sử dụng 2 khái niệm này trong 1 functional component, giúp code ngắn và clean hơn, đó chính là react hook.
  - Tại sao lại cho ra đời React hook?
- Hướng đối tượng luôn luôn là vấn đề khó khi học lập trình. con trỏ this, thuộc tính , phương thức... đó là rất nhiều logic và gây khó với một số bạn...

➔ **Nắm rõ loại nào thì ta sử dụng loại đó**

- React hook ra đời, cho phép tận dụng được các tính năng của class component trong functional component

**Vậy nên xài hook hay class component ?**

- Không có sự chênh lệch giữa performance giữa 2 loại này, cũng không cần loại bỏ hoàn toàn kiến thức về class component trước đây ta đã học, hook ra đời chỉ là thêm cho chúng ta một option để chọn khi làm mà thôi.



# ☐ React hooks là gì?

Link: <https://reactjs.org/docs/hooks-intro.html>

The screenshot shows the React.js documentation website. At the top, there's a navigation bar with links for 'React', 'Docs' (which is underlined), 'Tutorial', 'Blog', and 'Community'. A search bar is also present. Below the navigation, a banner reads 'Black Lives Matter. Support the Equal Justice Initiative.' On the left, a section titled 'Introducing Hooks' contains a code example:

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

To the right of the code, a sidebar menu is open, showing categories like 'INSTALLATION', 'MAIN CONCEPTS', 'ADVANCED GUIDES', 'API REFERENCE', and 'HOOKS'. The 'HOOKS' category is expanded, and its sub-items are listed within a red box:

- 1. Introducing Hooks
- 2. Hooks at a Glance
- 3. Using the State Hook
- 4. Using the Effect Hook
- 5. Rules of Hooks
- 6. Building Your Own Hooks
- 7. Hooks API Reference
- 8. Hooks FAQ

Below the 'HOOKS' section is a 'TESTING' category.

## React hook cơ bản

- + use State
- + use Effect
- + use memo
- + use callback
- + use ref
- + use Reducer
- + use Dispatch
- + use Selector
- + ...

## Motivation

Hooks solve a wide variety of seemingly unconnected problems in React that we've encountered over five years of writing and maintaining tens of thousands of components. Whether you're learning React, use it daily, or even prefer a different library with a similar component model, you might recognize some of these problems.

# ❑ useState Hook

- useState nhận vào input là stateDefault
- Kết quả trả về là 1 tuple 2 tham số là state, và hàm setState tương tự react LifeCycle (ReactClassComponent).
- Trong 1 component có thể khai báo nhiều hàm useState.
- useState dùng để thay thế thuộc tính this.state tương đương react class component.

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

export default function HookuseState(props) {
    const [state, setState] = useState({ like: 0 }); //useState nhận vào input là stateDefault
                                                    // Kết quả trả về là 1 tuple 2 tham số là state,
                                                    // và hàm setState tương tự react LifeCycle
                                                    // (ReactClassComponent)

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

    return (
        <div className="m-5" style={{width:200,height:200}}>
            <div className="card text-left" >
                
                <div className="card-body">
                    <h4 className="card-title">Picture</h4>
                    <p style={{ color: 'red' }}> {state.like} ❤</p>
                </div>
            </div>
            <span className="display-4" style={{ color: 'pink', cursor:'pointer' }} onClick={() => {
                setState({ like: state.like + 1 })
            }}>❤</span>
        </div>
    );
}
```

# ❑ useEffect Hook

- useEffect cho phép thực hiện side Effect (thay đổi các state ngoài hàm) bên trong các function component.
- useEffect chạy sau khi Dom được cập nhật (Có thể hiểu là chạy sau render).
- Có thể gọi useEffect nhiều lần trong 1 component
- Vì vậy useEffect có thể ứng với 3 lifecycle chạy sau render đó là didMount, componentDidUpdate, willUnmount.

➔ Xem demo

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

export default function HookUseEffect(props) {
  let [number, setNumber] = useState(1);
  useEffect(() => {
    console.log('Hàm được thực thi sau mỗi lần render');
    console.log('Cách viết này ứng với cả 2 lifeCycle didMount và componentDidUpdate');
  })
  useEffect(() => {
    console.log('Hàm gọi sau lần render đầu tiên');

    return () => {
      console.log('Hàm định nghĩa bên trong đây sẽ được gọi cuối cùng thay willUnmount');
    }
  }, [])
  useEffect(() => {
    console.log('Hàm gọi mỗi khi number (state) thay đổi sau khi render ! thay componentDidUpdate ')
  }, [number])

  return (
    <div className="container">
      Number: {number}
      <button className="btn btn-success" onClick={() => {
        setNumber(number + 1);
      }}>+</button>
    </div>
  )
}
```

```
import React, { Component } from 'react'

export default class LifeCycleDemo extends Component {
  constructor(props) {
    super(props);
    this.state = {
      number: 1
    }
  }
  render() {
    return (
      <div className="container">
        <h3>{this.state.number}</h3>
        <button className="btn btn-success" onClick={() => {
          this.setState({ number: this.state.number + 1 })
        }}>+</button>
      </div>
    )
  }
  componentDidMount() {
    console.log('didMount');
  }

  componentDidUpdate(prevProp,prevState) {
    console.log('didUpdate');
  }

  componentWillUnmount() {
    console.log('willUnmount');
  }
}
```

## ❑ Ví dụ về useEffect (ComponentDidMount)

- Trường hợp 1 thường dùng component tự động load dữ liệu từ api backend trả về sau khi html vừa load xong. Hoặc các api thư viện được gọi sau khi giao diện html được load hoàn tất.

Demo: <https://codesandbox.io/s/useeffect-componentdidmount-f3qq4q?file=/src/App.js>

```
1 import React from 'react'  6.9k (gzipped: 2.7k)
2 import { useState } from 'react'  4.1k (gzipped: 1.8k)
3 import { useEffect } from 'react'  4.1k (gzipped: 1.8k)
4 import axios from 'axios'  38.7k (gzipped: 13.2k)
5 export default function DemoUseEffectD() {
6
7     const [arrProd, setArrProd] = useState([]);
8
9     const getApi = async () => {
10        try {
11            let result = await axios({
12                url: 'https://shop.cyberlearn.vn/api/Product',
13                method: 'GET'
14            });
15            setArrProd(result.data.content);
16        } catch (err) {
17            console.log(err)
18        }
19    }
20
21    useEffect(() => {
22        //Tự động kích hoạt và thực thi ngay sau khi component render lần đầu tiên
23        getApi();
24    }, [])
25
26 }
```

## ☐ Ví dụ về useEffect (ComponentDidUpdate)

- Trường hợp 2 Khi bạn muốn 1 giá trị state nào thay đổi thì 1 khối lệnh nào khác thực thi tự đồng kích hoạt thì ta sử dụng `useEffect` có dependency.
- Demo: <https://codesandbox.io/s/useeffect-dependency-22rjgl>

Profile

user Name

Province

Tp. Hồ Chí Minh

District

Quận 1

Submit

(1) `setProvinceId(value);`

(2) `useEffect(() => {`

(3) `if(provinceId !== '') {`

(4) `}, [provinceId])`

```
export default function DemoUseEffect() {
  const [provinceId, setProvinceId] = useState('');
  const [arrDistrict, setArrDistrict] = useState([]);
  const [districtId, setDistrictId] = useState(districts[0].id);

  const handleChange = (e) => {
    let {value} = e.target;
    setProvinceId(value);
  }

  useEffect(() => {
    if(provinceId !== '') {
      setArrDistrict(districts.filter(item => item.provinceId === provinceId));
    }
  }, [provinceId])

  const handleSubmit = (e) => {
    e.preventDefault();
  }
}
```

Khi người dùng chọn provincId. Thì district sẽ load lại array mới

- Ngoài ra các state khác nếu có thay đổi thì useEffect này cũng không hoạt động. Chỉ hoạt động khi dependency provincId thay đổi

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

const arrProvince = [
  { id: 'HCM', name: 'Tp. Hồ Chí Minh' },
  { id: 'HN', name: 'Hà Nội' },
]

const districts = [
  { id: 'Q1', name: 'Quận 1', provinceId: 'HCM' },
  { id: 'Q3', name: 'Quận 3', provinceId: 'HCM' },
  { id: 'Q7', name: 'Quận 7', provinceId: 'HCM' },
  { id: 'BD', name: 'Ba Đình', provinceId: 'HN' },
  { id: 'TL', name: 'Từ Liêm', provinceId: 'HN' },
  { id: 'TH', name: 'Đống Đa', provinceId: 'HN' },
]
```

## ☐ Ví dụ về useEffect (ComponentWillUnmount)

- Dùng để clear các script của file component đó khi giao diện không được load trên ReactDOM

```
import React from 'react'
import { useEffect } from 'react';
import { useRef } from 'react';
import { useState } from 'react'
import { NavLink } from 'react-router-dom';

export default function DemoUseEffectWillUnmount() {

let [countDown, setCountDow] = useState(60);
let timeout = {};

const handleCountDown = () => {
    timeout = setInterval(() => {
        countDown--;
        setCountDow(countDown);
        console.log(countDown); //Khi component chuyển trang nếu không clear thì hàm này vẫn sẽ chạy ngầm
    }, 1000);
}

useEffect(() => {
    handleCountDown();
    return () => {
        clearInterval(timeout); //Clear timeout tại đây khi component mất khỏi React dom hàm này sẽ tự xoá
    }
}, [])

return (
    <div className='container'>
        <h3>Count down: {countDown}</h3>
        <NavLink to={''}>back to home</NavLink>
    </div>
)
}
```

Count down: 54

[back to home](#)

> 17 messages	58
> 17 user messages	57
✖ No errors	56
⚠ No warnings	55
> 17 info	54
✖ No verbose	53
	52
	51
	50
	49
	48
	47
	46
	45
	44
	43
	42

# ☐ Quy tắc Hook

Hook là các function JavaScript, có những quy luật bạn cần phải tuân theo khi sử dụng. Chúng tôi có một plugin linter để đảm bảo các luật này luôn luôn được áp dụng đúng:

Lưu ý:

- Chỉ gọi Hook ở trên cùng.
- Không gọi hook bên trong loop, câu điều kiện, hay các function lồng với nhau.
- Chỉ dùng hook với react function component.

Thay vì đó, luôn sử dụng Hook ở phần trên cùng của function. Với cách này, bạn đảm bảo các Hook được gọi theo đúng thứ tự trong các lần render. Nó cho phép React có được đúng state giữa nhiều lần gọi useState và useEffect. dưới.)

Lý do: <https://vi.reactjs.org/docs/hooks-rules.html>

Ngoài ra react cho phép chúng ta custom hook (Tạo ra các hook của riêng ta dựa trên các hook cơ bản của react ➔ Xây dựng trên nguyên tắc kế thừa không phải thay đổi)

Link tham khảo: <https://vi.reactjs.org/docs/hooks-custom.html>

## ❑ useCallback hook (Hook mở rộng)

Trước khi thảo luận tiếp về 2 hook còn lại là useCallback và useMemo, ta hãy cùng điểm qua một khái niệm mới , gọi là Memo (tương ứng với PureComponent trong class component).

Ví dụ bên dưới, nếu như ta dùng hook useState khi ta setState thì component bên trong mặc định sẽ được load lại, vì vậy ta dùng memo để bọc Component Comment lại => Khi nào có sự thay đổi prop hoặc state lại Comment thì comment mới loại lại. (Thay đổi ở đây cũng giống như PureComponent so sánh shallow)

```
import React,{useState,memo} from 'react'
import Comment from './Comment';

export default function HookUseCallBack() {

  let [like,setLike] = useState(1);

  return (
    <div className="m-5">
      Like: {like} ♥
      <br />
      <span style={{cursor:'pointer',color:'red',fontSize:35}} onClick={()=>{
        setLike(like+1)
      }}>♥</span>
      <br />
      <br />
      <Comment like={like} />
    </div>
  )
}
```

```
import React, { memo } from 'react'

const Comment = (props) => {
  console.log('comment');

  return (
    <div>
      <textarea></textarea> <br />
      <button>Gửi</button>
    </div>
  )
}

export default memo(Comment);
```

## ❑ useCallback hook (Hook mở rộng)

Vì memo chỉ hỗ trợ so sánh shallow (là so sánh các kiểu dữ liệu cơ sở như number, string, Boolean). Nên khi ta truyền vào là 1 function thì mặc dù function không hề thay đổi giá trị tuy nhiên khi setState (sử dụng hook useState) thì function được khai báo lại .

```
let renderNotify = () => {
    return `Bạn đã thả ${like} ♥ !`}
```



Mỗi lần render function được khai báo lại với từ khóa let.

```
import React,{useState,memo} from 'react'
import Comment from './Comment';

export default function HookUseCallBack() {

    let [like,setLike] = useState(1);

    let renderNotify = () => {
        return `Bạn đã thả ${like} ♥ !`
    }

    return (
        <div className="m-5">
            Like: {like} ♥
            <br />
            <span style={{cursor:'pointer',color:'red',fontSize:35}} onClick={()=>{
                setLike(like+1)
            }}>♥</span>
            <br />
            <br />
            <Comment renderNotify={renderNotify}/>
        </div>
    )
}
```

```
import React, { memo } from 'react'

const Comment = (props) => {

    console.log('comment');

    return (
        <div>
            {props.renderNotify()}
            <br />
            <textarea></textarea> <br />
            <button>Gửi</button>
        </div>
    )
}

export default memo(Comment);
```

Mỗi lần hàm setLike được gọi cả function chạy lại vì vậy hàm renderNotify được khai báo lại => xem như là giá trị mới

## ❑ useCallback hook (Hook mở rộng)

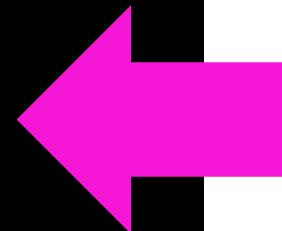
```
import React, { useState, memo, useCallback } from 'react'
import Comment from './Comment';

export default function HookUseCallBack() {
  let [like, setLike] = useState(1);

  let renderNotify = () => {
    return `Bạn đã thả ${like} ❤ !`
  }

  const callbackRenderNotify = useCallback(renderNotify,
    [like],
  )

  return (
    <div className="m-5">
      Like: {like} ❤
      <br />
      <span style={{ cursor: 'pointer', color: 'red', fontSize:
35 }} onClick={() => {
        setLike(like + 1)
      }}>❤</span>
      <br />
      <br />
      <Comment renderNotify={callbackRenderNotify} />
    </div>
  )
}
```



Do đó ở đây ta cần useCallback, showNumberCallback ở đây là một bản snapshot của renderNotify, và nó chỉ được khai báo lại khi like thay đổi, nếu để mảng rỗng, có nghĩa là sẽ khai báo 1 lần duy nhất.

➤ Dẫn tới , khi component cha thay đổi state và render lại, nếu like không đổi, thì renderNotify sẽ không đổi, dẫn tới component con sẽ ko render lại

## ❑ useMemo hook (Hook mở rộng)

```
import React, { useMemo, useState } from 'react'
import Cart from './Cart';

export default function HookMemo(props) {
  let [like, setLike] = useState(1);
  let cart = [
    { id: 1, name: 'iphone', price: 1000 },
    { id: 2, name: 'htc phone', price: 2000 },
    { id: 3, name: 'lg phone', price: 3000 }
  ];
  const cartMemo = useMemo(() => cart, []);
  return (
    <div className="m-5">
      Like: {like} ❤
      <br />
      <span style={{ cursor: 'pointer', color: 'red', fontSize: 35 }} onClick={() => {
        setLike(like + 1);
      }}>❤</span>
      <br />
      <br />
      <Cart cart={cartMemo} />
    </div>
  )
}
```

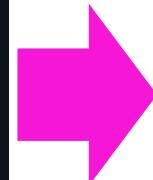


```
import React,{memo} from 'react'
function Cart(props) {
  console.log('cart')
  return (
    <div>
      <table class="table">
        <thead>
          <tr>
            <th>id</th>
            <th>name</th>
            <th>price</th>
          </tr>
        </thead>
        <tbody>
        {
          props.cart.map((item,index)=>{
            return <tr key={index}>
              <td>{item.id}</td>
              <td>{item.name}</td>
              <td>{item.price}</td>
            </tr>
          })
        }
      </tbody>
    </div>
  )
}
export default memo(Cart);
```

- Nhìn vào ví dụ trên ta thấy khi ta setLike không ảnh hưởng đến props (cart) của component Cart mặc dù vậy nhưng component Cart vẫn load lại → Điều này là không cần thiết (Component Cart không cần thiết phải load lại), cũng như useCallback useMemo dùng cho các biến là object thay vì useCallback là hàm.

## ❑ useRef (Hook mở rộng)

```
JS HookRef.js
src > hookUseState > JS HookRef.js > ...
1  import React, { useState, useRef } from 'react'  8.3K (gzipped: 3.3K)
2  export default function HookRef(props) {
3      const inputUserName = useRef(null);
4      const inputPassword = useRef(null);
5
6      let [user, setUser] = useState({
7          userName: '',
8          passWord: ''
9      });
10     let handleSubmit = ()=>{
11         console.log(inputUserName.current.name,inputUserName.current.value);
12         console.log(inputPassword.current.name,inputPassword.current.value);
13     }
14
15     return (
16         <div className="container">
17             <h3>Login</h3>
18             <div className="form-group">
19                 <p>UserName</p>
20                 <input ref={inputUserName} className="form-control" name="userName" />
21             </div>
22             <div className="form-group">
23                 <p>Password</p>
24                 <input ref={inputPassword} className="form-control" name="password" />
25             </div>
26             <div className="form-group">
27                 <button onClick={handleSubmit} type="button" className="btn btn-success">Login</button>
28             </div>
29         </div>
30     )
}
```



The screenshot shows a browser window titled "Trương Tân Khải" with the URL "localhost:3000". The page displays a "Login" form with fields for "UserName" (containing "taikhoan") and "Password" (containing "matkhau"). A green "Login" button is visible. Below the form, the browser's developer tools are open, specifically the "Console" tab. It shows two entries: "userName taikhoan" at line 11 and "password matkhau" at line 12, both originating from "HookRef.js".

- Ứng với thuộc tính `this.ref` sử dụng `createRef` ở khóa 2. React cũng cung cấp cho ta 1 hook là `useRef` làm công việc tương tự.

- Giá trị `useRef` không thay đổi sau mỗi lần render. Ngoài việc sử dụng như DOM `useRef` còn dùng trong việc lưu trữ các biến, hàm, mảng, object sau mỗi lần render

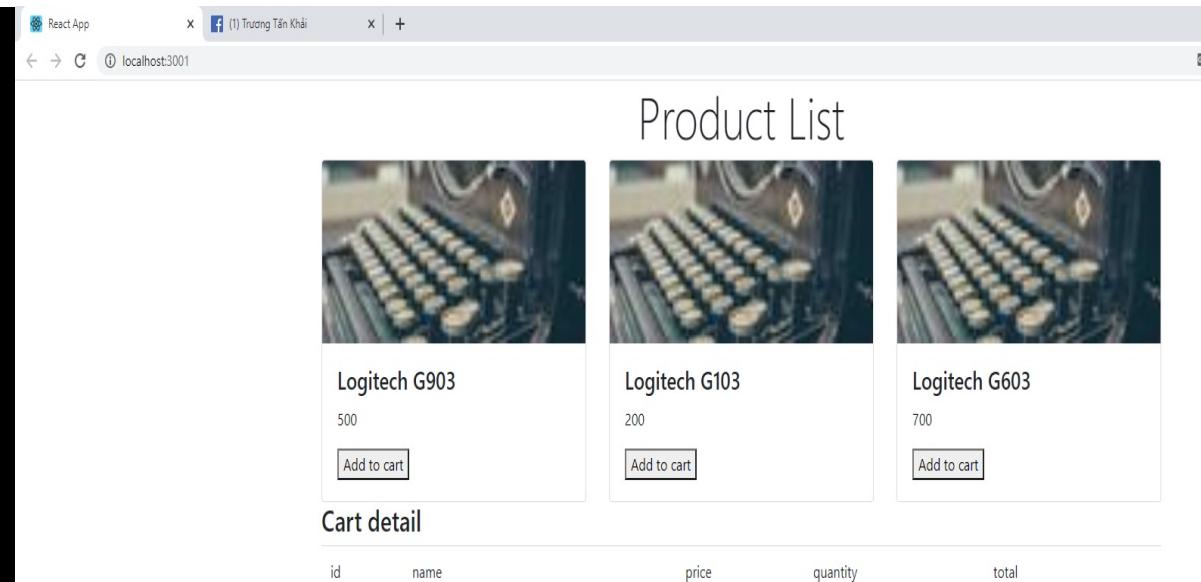
## ❑ useReducer hook (Hook mở rộng)

useReducer cũng giống như 1 phiên bản nâng cấp của useState. Dùng để quản lý những state của giao diện. Thay vì các bạn dùng nhiều useState hoặc useState với value là nested object/array và viết nhiều function để thay đổi state thì bây giờ các bạn có thể tổ chức state và các action làm thay đổi state đó 1 cách logic nhờ useReducer.

```
let initialCart = [];
let cartReducer = (state, action) => {
  switch (action.type) {
    case 'addToCart':
      let index = state.findIndex(item => item.id === action.product.id);
      if (index != -1) {
        state[index].quantity += 1;
        return [...state];
      }
      return [...state, { ...action.product, quantity: 1 }];
    default: return state;
  }
}

let arrProduct = [
  { id: 1, name: 'Logitech G903', price: 500 },
  { id: 2, name: 'Logitech G103', price: 200 },
  { id: 3, name: 'Logitech G603', price: 700 },
]
```

Ví dụ ta có 1 state là giỏ hàng, ta sẽ viết 1 reducer giống hệt như redux



Ví dụ demo về giỏ hàng

# ❑ useReducer hook (Hook mở rộng)

```
export default function HookUseReducer() {
  const [cart, dispatch] = useReducer(cartReducer, initialCart)

  return (
    <div className="container">
      <h3 className="display-4 text-center">Product List</h3>
      <div className="row">
        {
          arrProduct.map((item, index) => {
            return <div className="col-4" key={index}>
              <div className="card text-left">
                
                <div className="card-body">
                  <h4 className="card-title">{item.name}</h4>
                  <p className="card-text">{item.price}</p>
                  <button onClick={() => {
                    dispatch({
                      type: 'addToCart',
                      product: item
                    })
                  }}>Add to cart</button>
                </div>
              </div>
            </div>
          )}
        )
      </div>
      <h3>Cart detail</h3>
      <table className="table">
        <thead>
          <tr>
            <td>id</td>
            <td>name</td>
            <td>price</td>
            <td>quantity</td>
            <td>total</td>
            <td></td>
          </tr>
        </thead>
        <tbody>
          {cart.map((item, index) => {
            return <tr key={index}>
              <td>{item.id}</td>
              <td>{item.name}</td>
              <td>{item.price}</td>
              <td>{item.quantity}</td>
              <td>{item.price * item.quantity}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  )
}
```

State chúng ta là 1 giỏ hàng với các action tương ứng là thêm xóa sửa sản phẩm

# ❑ useContext hook (Hook mở rộng)

```
import React, { useReducer } from 'react'

export const storeContext = React.createContext();
let initialCart = [];

let cartReducer = (state, action) => {
  switch (action.type) {
    case 'addToCart':
      let index = state.findIndex(item => item.id === action.item.id);
      if (index != -1) {
        state[index].quantity += 1;
        return [...state];
      }
      return [...state, { ...action.item, quantity: 1 }]
    default: return state;
  }
}

export default function Context(props) {

  let [cart, dispatch] = useReducer(cartReducer, initialCart);
  //Có thể kết hợp useState hoặc useReducer

  const store = {
    cartReducer: [cart, dispatch], //Tạo ra store giống như rootReducer
  };

  return (
    <storeContext.Provider value={store}>
      {props.children}
    </storeContext.Provider>
  )
}
```

Thiết kế component ContextProvider

Sử dụng trong app

Việc sử dụng useContext tương đương với việc sử dụng contextApi trong react class component không có quá nhiều sự khác biệt

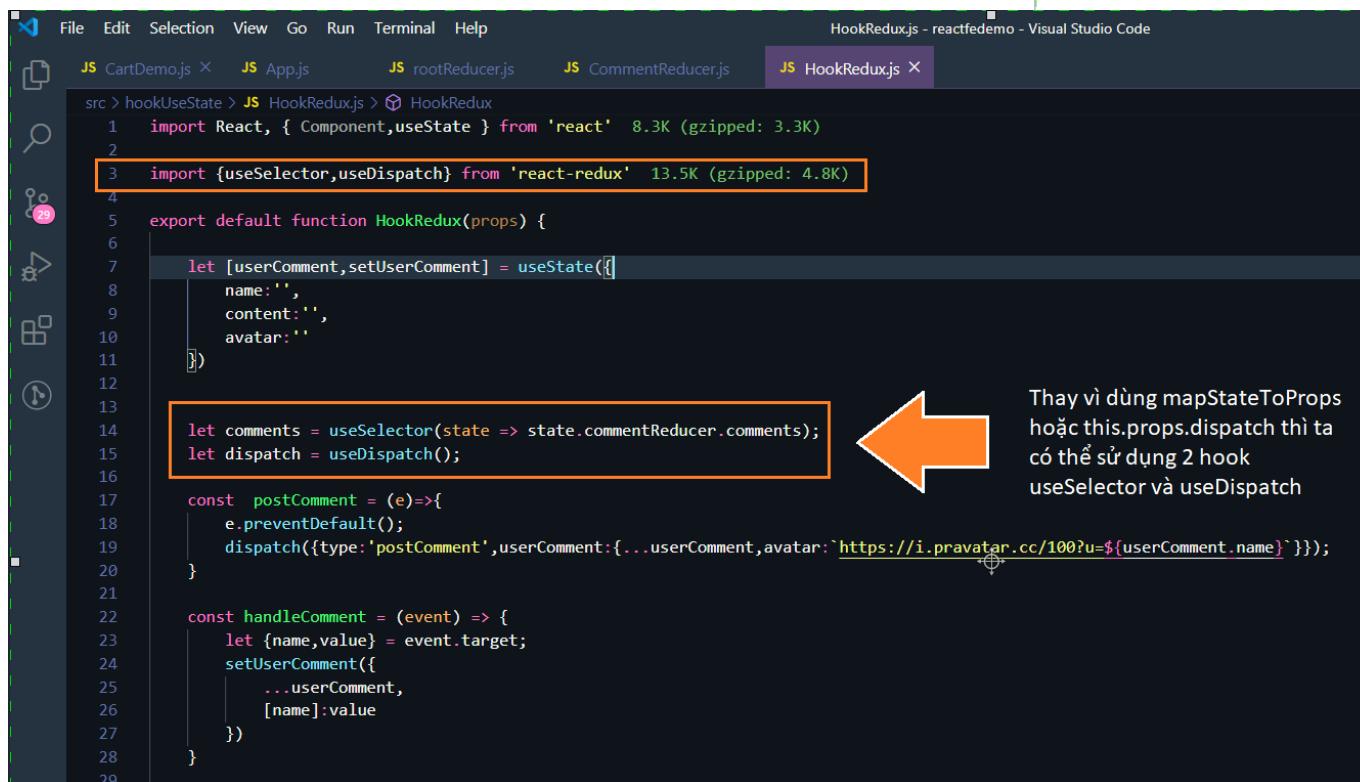
```
JS CartDemo.js X
src > hookUseState > JS CartDemo.js > arrProduct
1 import React,{useContext} from 'react' 8.3K (gzipped: 3.3K)
2 import {storeContext}from './Context';
3
4 let arrProduct = []
5   { id: 1, name: 'Logitech G903', price: 500 },
6   { id: 2, name: 'Logitech G103', price: 200 },
7   { id: 3, name: 'Logitech G603', price: 700 },
8 ]
9
10 export default function CartDemo(props) {
11   const {cartReducer} = useContext(storeContext);
12   let [cart,dispatch] = cartReducer;
13   // console.log('context',context)
14
15   return (
16     <div className="container">
17       <h3 className="display-4 text-center">Product List</h3>
18       <div className="row">
19         {
20           arrProduct.map((item, index) => {
21             return <div className="col-4" key={index}>
22               <div className="card text-left">
23                 
24                 <div className="card-body">
25                   <h4 className="card-title">{item.name}</h4>
26                   <p className="card-text">{item.price}</p>
27                   <button onClick={() => {
28                     dispatch({
29                       type: 'addToCart',
30                       product: item
31                     })
32                   }}>Add to cart</button>
33                 </div>
34               </div>
35             </div>
36           )
37         </div>
38         <h3>Cart detail</h3>
39       )
40     )
41   export default App;
```

Sử dụng hook useContext bắt cứ đâu trong Context.Provider để thay đổi state

Các component bên trong có thể truy xuất đến store thông qua useContext.

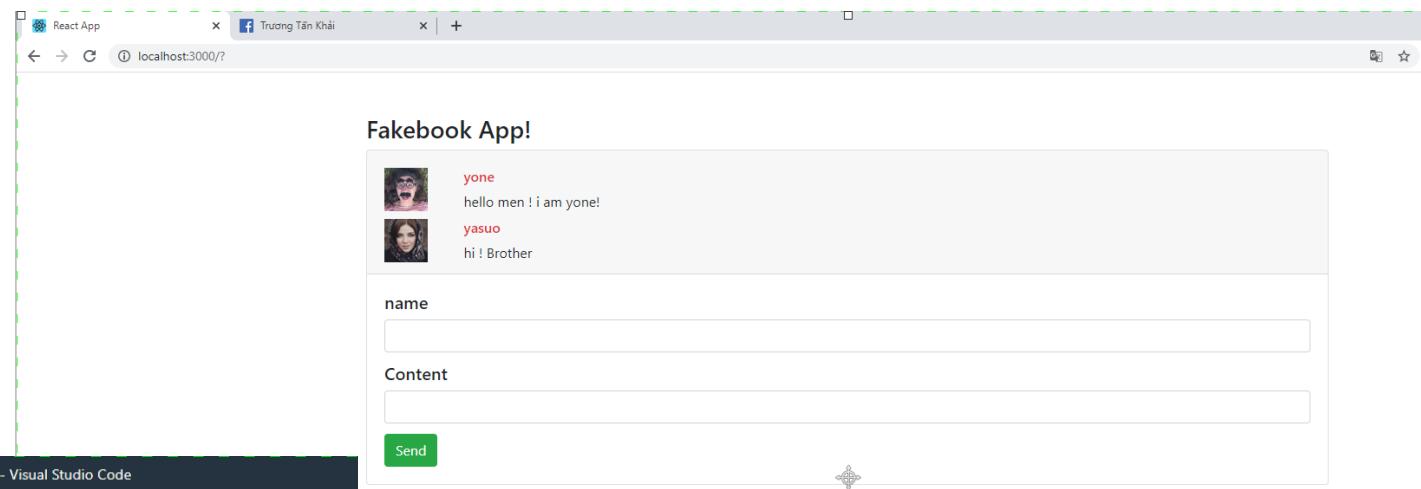
## ❑ useSelector - useDispatch hook (Hook mở rộng)

- 2 Thư viện hook useSelector và useDispatch không phải của react core cung cấp, mà 2 thư viện này thuộc thư viện react-redux cung cấp cho phép ta truy cập đến reducer một cách dễ dàng và ngắn gọn hơn.



```
File Edit Selection View Go Run Terminal Help
src > hookUseState > JS HookRedux.js > HookRedux
1 import React, { Component, useState } from 'react' 8.3K (gzipped: 3.3K)
2
3 import {useSelector, useDispatch} from 'react-redux' 13.5K (gzipped: 4.8K)
4
5 export default function HookRedux(props) {
6
7   let [userComment, setUserComment] = useState({
8     name:'',
9     content:'',
10    avatar:''
11  })
12
13
14   let comments = useSelector(state => state.commentReducer.comments);
15   let dispatch = useDispatch();
16
17   const postComment = (e)=>{
18     e.preventDefault();
19     dispatch({type:'postComment', userComment:{...userComment, avatar:`https://i.pravatar.cc/100?u=${userComment.name}`}});
20   }
21
22   const handleComment = (event) => {
23     let {name,value} = event.target;
24     setUserComment({
25       ...userComment,
26       [name]:value
27     })
28   }
29 }
```

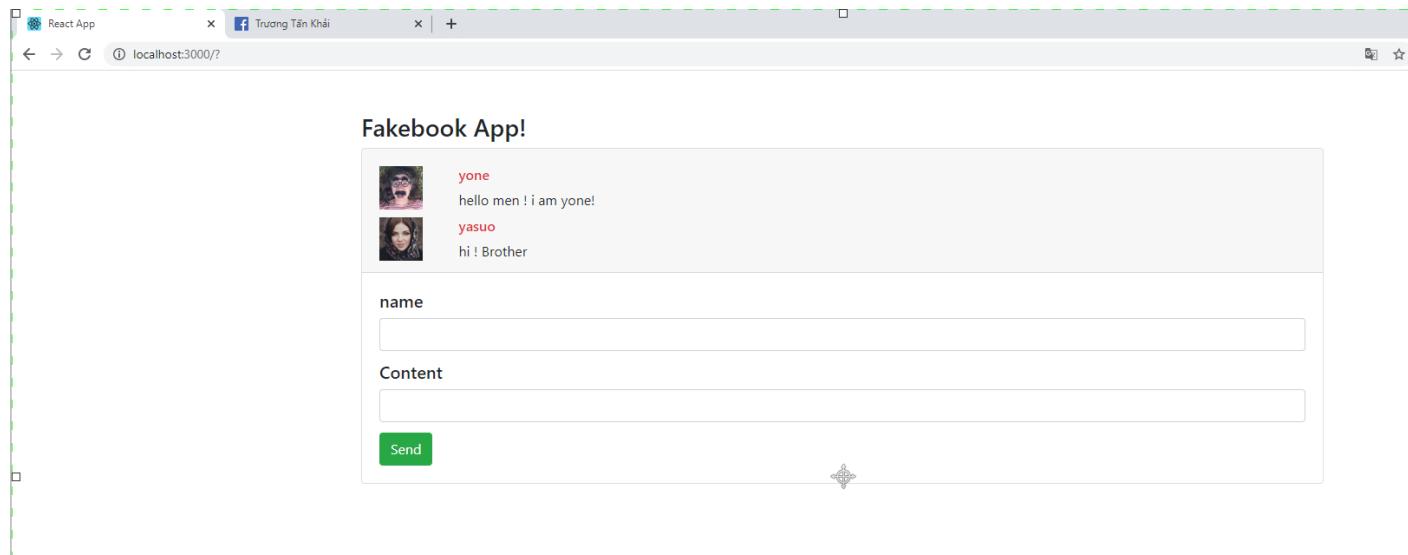
Thay vì dùng mapStateToProps  
hoặc this.props.dispatch thì ta  
có thể sử dụng 2 hook  
useSelector và useDispatch



Xây dựng ứng dụng chat app đơn giản

# ☐ Bài tập

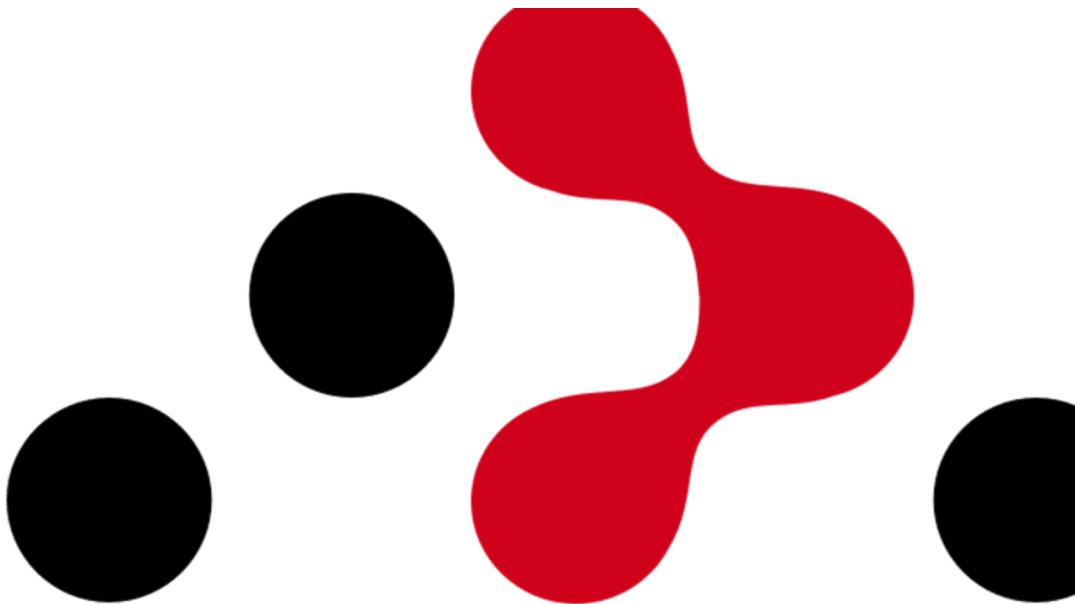
Sử dụng hook xây dựng tính năng xóa và sửa comment



## ☐ Các hook routing thông dụng

### React hook routing cơ bản

- + use Navigate
- + use Params
- + use History
- + useSearchParams
- + ....



## ❑ useNavigate (React router dom – hook)

- Giúp ta chuyển hướng trang sau khi xử lý 1 hành động (handleSubmit api, handleChange ...)
- Ví dụ bên dưới với chức năng giả lập đăng nhập api sau thời gian 3s xác thực rồi mới chuyển hướng trang.
- Cú pháp:

```
const navigate = useNavigate();

//Chuyển hướng

navigate('path')
```

```
import React, { useRef } from 'react'
import { history } from '../../App';
import { useNavigate } from "react-router-dom";

export default function ReactForm(props) {
  const navigate = useNavigate();
  // console.log(props)
  const userLoginRef = useRef({
    userName:'',
    passWord:''
  });
  const handleChange = (e) => {
    const {value,id} = e.target;
    userLoginRef.current[id] = value;
    console.log(userLoginRef.current);
  }

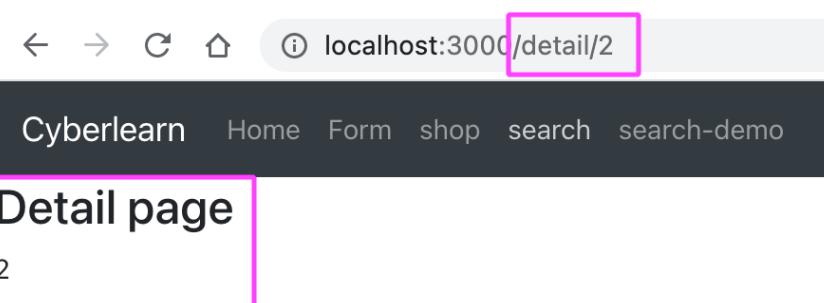
  const handleSubmit = async (e) => {
    e.preventDefault(); //Chặn reload browser
    console.log(userLoginRef.current);
    let promise = new Promise (resolve => {
      setTimeout(()=> {
        console.log('Đăng nhập api')
        resolve('Đăng nhập thành công');
      }, 3000);
    })
    let result = await promise;
    console.log(result);
    navigate('home');

    //Có replace : là k lưu lại lịch sử khi back trang
    //Không có replace là lưu lại lịch sử khi back trang
  }

  return (
    <form className='container' onSubmit={handleSubmit}>
      <h3>Login</h3>
      <div className='form-group'>
        <p>username</p>
        <input className='form-control' id="userName" onChange={handleChange} />
      </div>
      <div className='form-group'>
        <p>password</p>
        <input className='form-control' id="passWord" onChange={handleChange} />
      </div>
      <div className='form-group'>
        <button className='btn btn-success'>Login</button>
      </div>
    </form>
  )
}
```

## ❑ useParams (React router dom – hook)

- Giúp ta lấy được các tham số trên url của trình duyệt.
- Để định cài đặt param trên url như hình thì ta setup như code demo bên phải.



```
38   <Route path='detail'>
39     <Route path=':id' element={<Detail />}></Route>
40   </Route>
```

Định nghĩa <Route />

```
import React from 'react'
import { useLocation, useMatch, useParams } from 'react-router-dom'

export default function Detail(props) {
  const params = useParams(); //Dùng useParams để lấy param
  const match = useLocation();

  console.log('id', params.id)
  console.log(match);

  return (
    <div>
      <h3>Detail page</h3>
      <p>{params.id}</p>
    </div>
  )
}
```

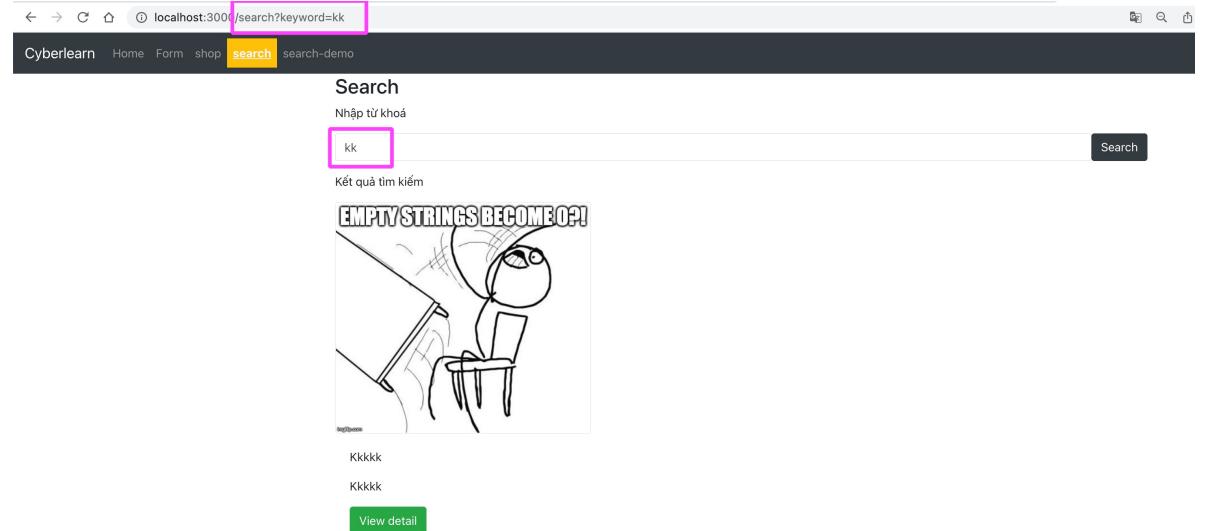
Sử dụng useParams để lấy id

Ngoài ra còn có thể sử dụng useLocation để lấy thêm thông tin

## ❑ useSearchParams (React router dom – hook)

- Tương tự như cách truyền tham số qua url thì react-router-dom cho phép ta định nghĩa tham số trên url của trình duyệt thông qua `?QueryParam`.
- Điều này thường được áp dụng trên các trang tìm kiếm để tạo link động dựa trên kết quả người dùng nhập và có thể lưu trữ lại link đó để share hoặc gửi cho người khác.

Demo: <https://codesandbox.io/s/usesearchparams-2f8trk?file=/src/App.js>



## ❑ Custom hook

- Ở hook có 1 số nguyên tắc cơ bản (Ví dụ: cannot be called inside a callback...). Vì vậy để sử dụng được HOC hoặc kế thừa như class component đôi lúc chúng ta sẽ lỗi vì phạm nguyên tắc. Vì vậy để tiện lợi cho lập trình viên thì muốn tái sử dụng lại các logic mà ta đã xây dựng cho 1 Functional component ta có thể tận dụng Custom hook.
- Cách khai báo cũng tương tự react component tuy nhiên kết quả trả về của function custom hook sẽ thay vì jsx thì custom hook trả về giá trị.
- Bên phải là demo về hook Route.

+ Thay vì ở component nào chúng ta cũng phải import các hook như useParams, useLocation,... Thị ta có thể tách logic này ra 1 custom hook để component nào cần sử dụng các hook này ta chỉ việc khai báo và lấy ra sử dụng .

+ Một số thư viện custom hook tham khảo

<https://github.com/streamich/react-use?fbclid=IwAR3R1tQBNTKowvRT60CDIjnIstcfGUnMewyi7R3yP2QSLBUJH1EgTg5hzxE>

```
import React from 'react'
import { useLocation, useNavigate, useParams, useSearchParams } from 'react-router-dom'

export default function useRoute() {
  const params = useParams();
  const match = useLocation();
  const search = useSearchParams();
  const navigate = useNavigate();
  return { params, match, search, navigate };
}
```

```
git clone https://github.com/streamich/react-use.git
cd react-use
git checkout custom-hooks
cd custom-hooks
git checkout useRoute
cd ..
cd examples
cd routes
cd custom-hook
cd DemoCustomHook
node index.js
1  import React from 'react' 6.9k (gzipped: 2.7k)
2  import useRoute from './useRoute'
3
4  export default function DemoCustomHook(props) {
5
6    const { match, params, search, navigate } = useRoute();
7    console.log({ match, params, search, navigate })
8
9    return (
10      <div>
11        <h3>DemoCustomHook</h3>
12      </div>
13    )
14  }
15
16
17
18
```