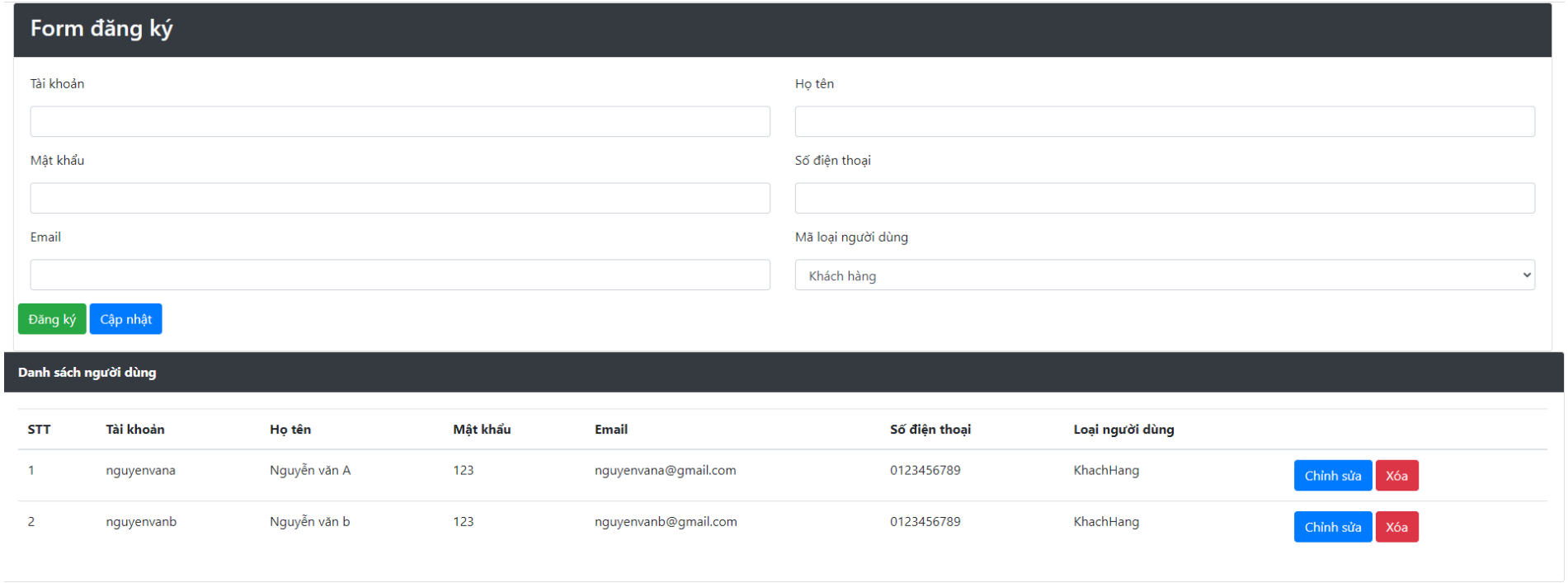
❑ Bài tập quản lý người dùng thực hành lifecycle

- Xây dựng giao diện như hình bằng bootstrap 4, hoặc sử dụng template có sẵn (tìm trên mạng), hoặc định nghĩa = styledComponent
- Yêu cầu:
  + Xây dựng component

```
<BaiTapQuanLyNguoiDung>
        <FormDangKy/>
        <TableDanhSachNguoiDung />
</BaiTapQuanLyNguoiDung>
```

  + Xác định các giá trị thay đổi, tổ chức lưu trữ redux
  + Hoàn thiện các chức năng đăng ký, xóa, chỉnh sửa, cập nhật người dung (Lưu ý: Viết theo action creator). Lưu ý: Validation input



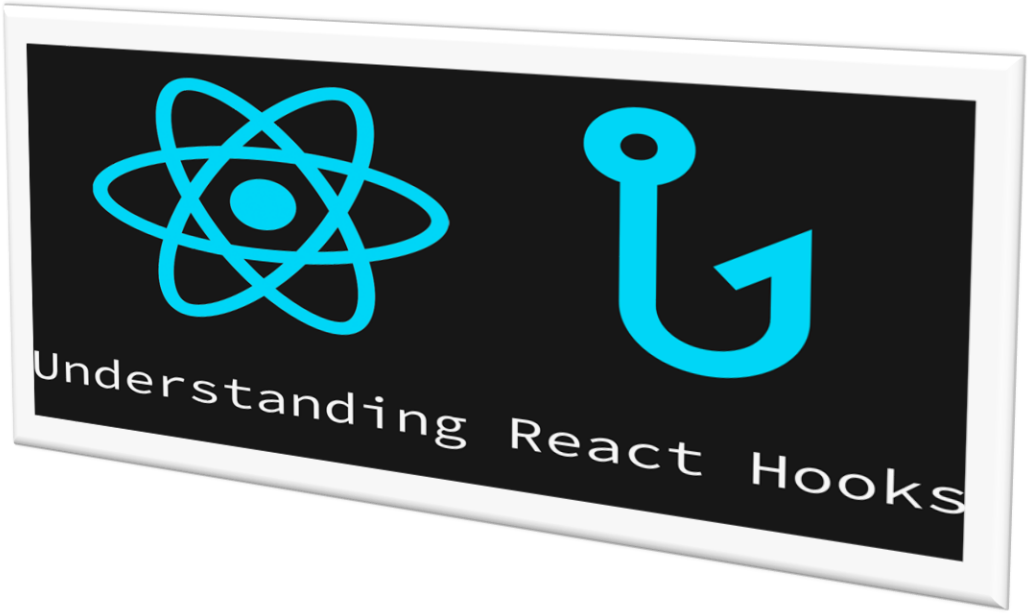
❑ React hooks là gì? (React version >= 16.8)

■ 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

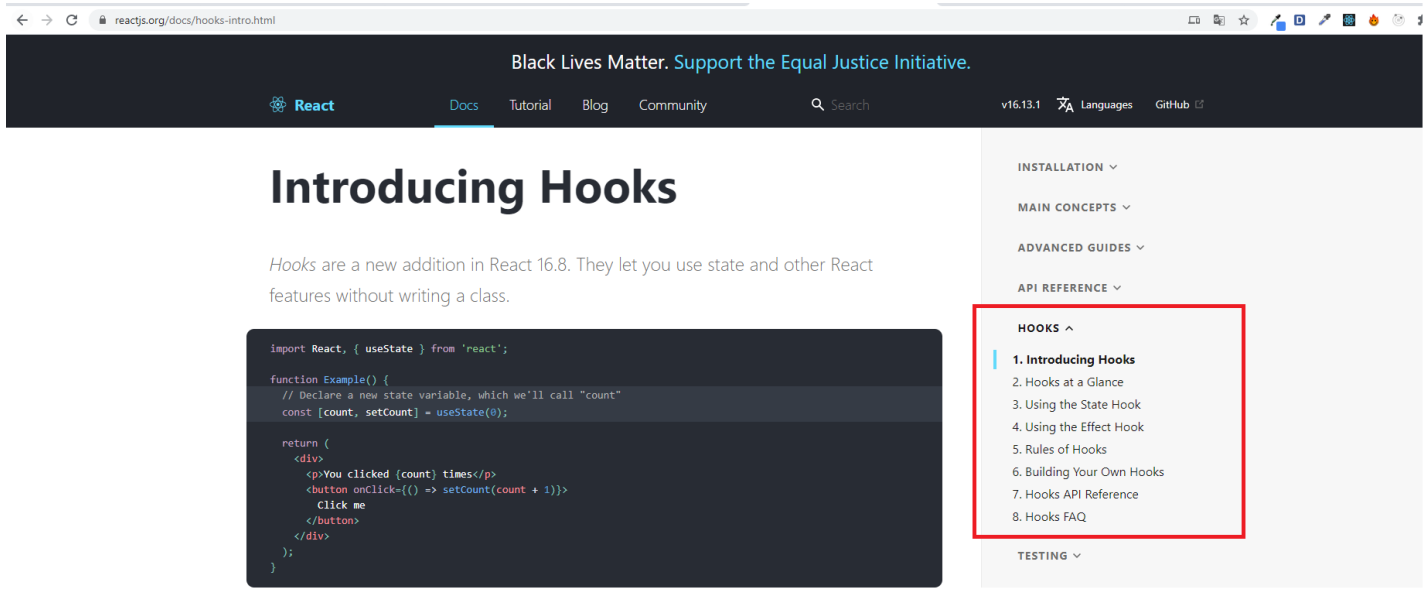
■ React hook ra đời, cho phép tận dụng được các tính năng của class component trong functional component

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



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.

React hook cơ bản

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

❑ 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"  >
                    <img  className="card-img-top" src="https://picsum.photos/200/200" alt="picture" />
                    <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, didUpdate, willUnmount.

➔ Xem demo

```jsx
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 didUpdate');
    })
    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 didUpdate  ')
    }, [number])

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

```jsx
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');
    }
}
```

❑ 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 tắt 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 hổ 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>
    )
}
```

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

```
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);
```

❑ useCallback hook (Hook mở rộng)

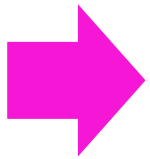
```
import React, { useState, memo,useCallback } from 'react'
import Comment from './Comment';

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)

```jsx
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>
    )
}
```



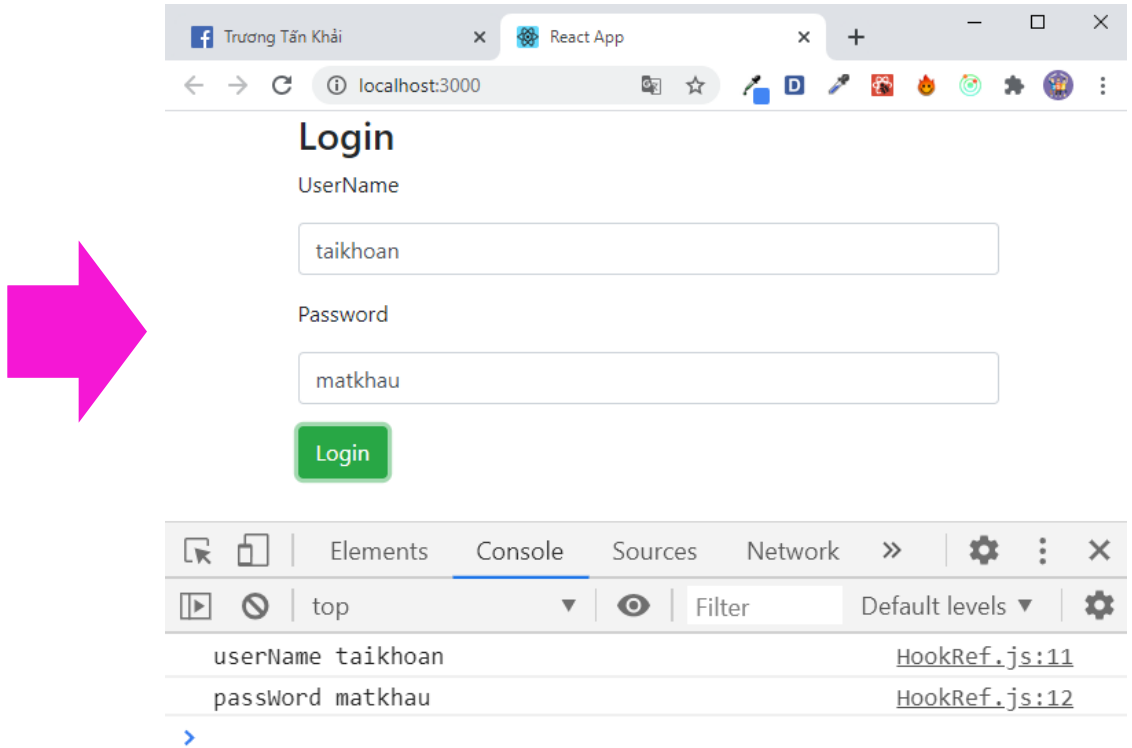
```jsx
import React,{memo} from 'react'
function Cart(props) {
    console.log('cart')
    return (
        <div>
            <table class="table">
                <thead>
                    <tr>
                        <th>id</th>
                        <th>name</th>
                        <th>price</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>
            </table>
        </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)

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      return (
15          <div className="container">
16              <h3>Login</h3>
17              <div  className="form-group">
18                  <p>UserName</p>
19                  <input ref={inputUserName}  className="form-control"  name="userName" />
20              </div>
21              <div className="form-group">
22                  <p>Password</p>
23                  <input  ref={inputPassword}  className="form-control" name="passWord" />
24              </div>
25              <div className="form-group">
26                  <button onClick={handleSubmit} type="button" className="btn btn-success">Login</button>
27              </div>
28          </div>
29      )
30  }
```



- 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.

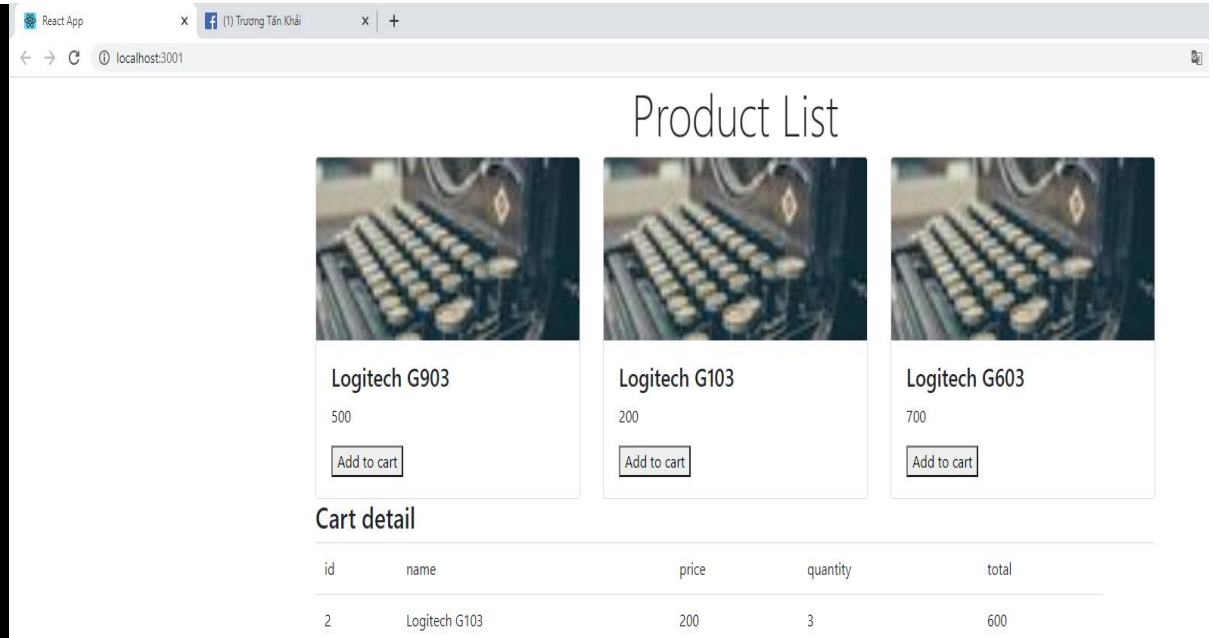
- Giá trị useReft không thay đổi sau mỗi lần render. Ngoài việc sử dụng như DOM useRef còn dung trong việc lưu trữ các biến, hàm, mảng, object sau mỗi lần render

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.pro
duct.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)

```jsx
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">
                                <img className="card-img-top" src="http://picsum.photos/100/50" alt={index} />
                                <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

```
import ContextProvider from './hookUseState/Context';
import CartDemo from './hookUseState/CartDemo';

function App() {
  return (
    <ContextProvider>
    <div className="App">

      <CartDemo />
      </div>
    </ContextProvider>
  );
}

export default App;
```

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

CartDemo.js

src > hookUseState > CartDemo.js > arrProduct

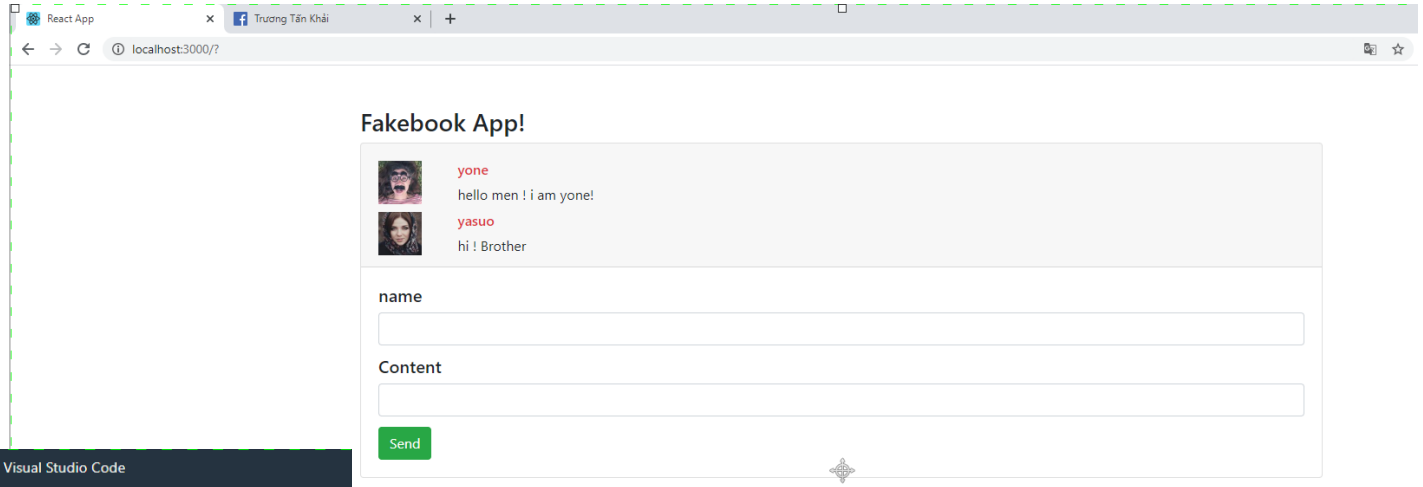
```
import React,{useContext} from 'react'  8.3K (gzipped: 3.3K)
import {storeContext}from './Context';

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

export default function CartDemo(props) {
    const {cartReducer} = useContext(storeContext);
    let [cart,dispatch] = cartReducer;
    // console.log('context',context)

    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">
                                <img className="card-img-top" src="http://picsum.photos/100/50" alt={index} />
                                <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>
```

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

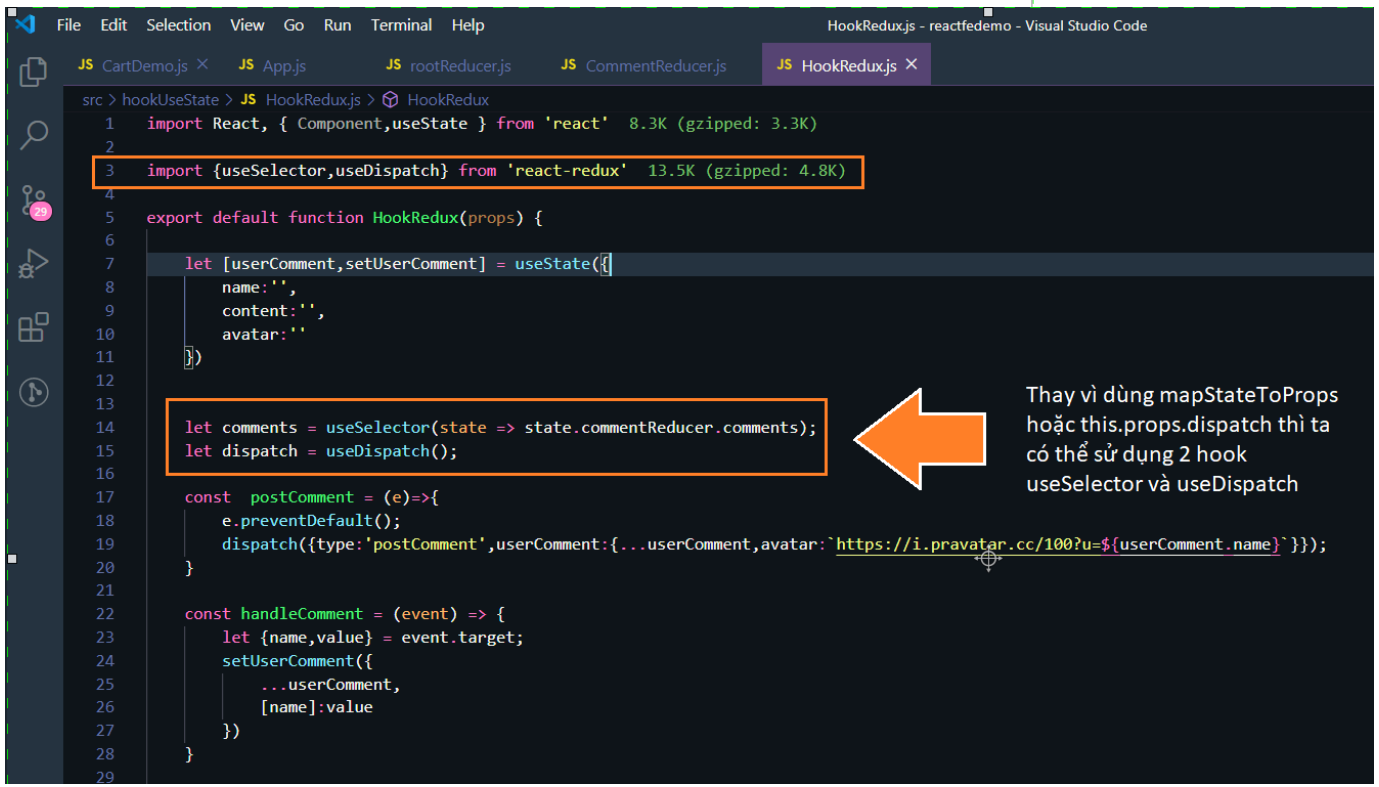
Các component bên trong có thể truy xuất đến store thông qua useContext.

❑ 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.



Xây dựng ứng dụng chat app đơn giãn



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

Bài tập

Sử dụng hook xây dựng tính năng xóa và sửa comment