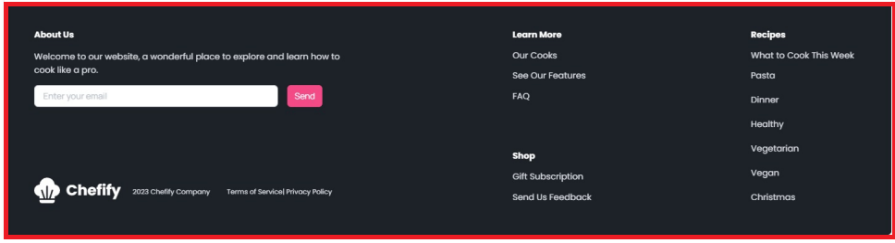
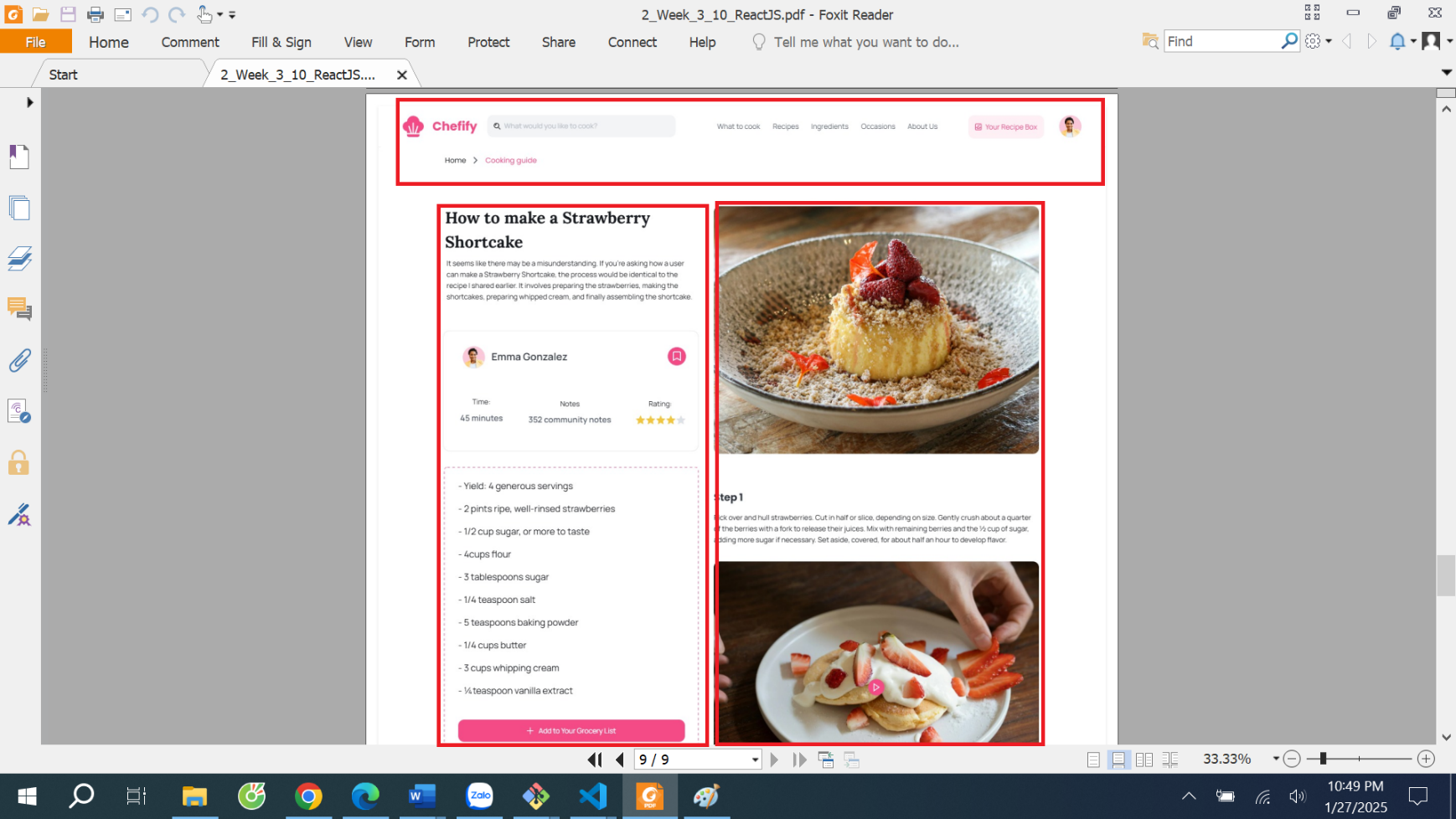
**Tài liệu môn học: Phát triển giao diện ứng dụng - react**

# React Component - Làm Quen với Cách Viết Component



**Component:** Chia bố cục trang, tái sử dụng code.

1. **Class component**

Tạo một thư mục **components** trong **src**

Tạo file **Mycomponent.jsx** //tên class component phải bắt đầu bằng chữ hoa

import React from "react";

class Mycomponent extends React.Component {

    render() {

        return (

            <div>My first component</div>//chỉ trả về 1 phần tử

        );

    }

}

export default Mycomponent;

**Sử dụng class component**

* Trở lại file **App.jsx**
* import Mycomponent from './components/Mycomponent'
* Trong file **App.jsx,** chứa **funtion App()** viết theo kiểu **function component** chứa **class component Mycomponent**

function App() {

  const [count, setCount] = useState(0)

  return (

    <>

      <div>

        Hello world

        <Mycomponent></Mycomponent> //class component

</div>

</>)

}

Đối với **vite**, component App() được gọi trong file main.jsx

import { StrictMode } from 'react'

import { createRoot } from 'react-dom/client'

import './index.css'

import App from './App.jsx'

createRoot(document.getElementById('root')).render(

  <StrictMode>

**<App />**

  </StrictMode>,

)

**Component được nhúng vào div “root” 🡺 hiển thị ngay trên trình duyệt**

**Cách react quản lý state của nó**

* **State: là 1 biến object cùa js, giúp mô tả trạng thái của ứng dung**

class Mycomponent extends React.Component {

    state = {

        Name: 'Eric',

        Age: 28,

        Address: 'HCM'

    }

    render() {

        return (

            <div>

                My name is: {this.state.Name}

                <br />

                Address: {this.state.Address}

            </div>

        );

    }

}

export default Mycomponent;

# State và Props: quản lý data trong react

**Props** và **state** xác định những gì một component hiển thị và nó hoạt động như thế nào

* **Props: truyền dữ liệu từ cha sang con**

**Truyền giá trị  
ví dụ: Mycomponent: (cha)**

import React from "react";

import Childcomponent from "./Chidcomponent"

class Mycomponent extends React.Component {

    render() {

        let parent = "this is my parent";

        return (

            <div>

                <Childcomponent **myProp={parent}**></Childcomponent>

*//truyền giá trị cho component con thông qua prop*

            </div>

        );

    }

}

export default Mycomponent;

----------------

**Childcomponent (component con)**

class Childcomponent extends React.Component {

    render() {

        return (

            <>

                <h1>

I just get from parent value:{this.props.myProp}

                </h1>

            </>

        );

    };

}

export default Childcomponent;

--------------

**Truyền hàm**

**Mycomponent: (cha): chứa hàm sum**

import React from "react";

import Childcomponent from "./Chidcomponent"

class Mycomponent extends React.Component {

    sum = (a, b) => {

        return a + b;

    }

    render() {

       return (

          <div>

<Childcomponent myProp={parent} **sum={this.sum}**></Childcomponent>

*//truyền hàm sum cho component con thông qua prop*

          </div>

        );

    }

}

export default Mycomponent;

**Childcomponent (component con)**

import React from "react";

class Childcomponent extends React.Component {

    render() {

        return (

            <>

                <h1>

**call function Sum:6+7={this.props.sum(6, 7)}**

//gọi hàm sum từ component cha

                </h1>

            </>

        );

    };

}

export default Childcomponent;

*lưu ý: có thể viết gọn hơn:*

**let {myProp , sum}=this.props;**

return (

            <>

                <h1>

**call function sum:6+7={sum(6, 7)}**

//gọi hàm sum từ component cha

                </h1>

            </>

* **State: là 1 biến object của javascript, mộ tả trạng thái của ứng dụng**

Để định nghĩa **state** của một ứng dụng, bắt buộc phải khai báo một **constructor**

Ví dụ: dung state quản lý trạng thái của một input

class Childcomponent extends React.Component {

**constructor(props)** {

**super**(props);

**this.state = {**

**valueInput: 'abc'//giá trị mặc định**

        }

    }

**Tạo một input:**

 <div>

      <input type="text" />

 </div>

**Gắn giá trị mặc định cho input**

 <input value={this.**state**.**valueInput**} type="text" />

**Thay đổi giá trị của input**

**Hàm setState(): cập nhật biến state của react,**

**Merge state: mỗi lần cập nhật một biến , thì các biến còn lại vẫn giữ nguyên**

Ví dụ:

Định nghĩa hàm handleInput(event)

**handleInput** = (event) => {

        //console.log(event.target.value);

        this.**setState**({//hiển thị đúng giá trị được nhập vào

            valueInput: event.target.value

        })

    }

Gán sự kiện **onchange()** cho input gọi hàm handleInput để thay đổi giá trị của input

<input value={this.state.valueInput}

onChange={(event) => {this.handleInput(event) }}

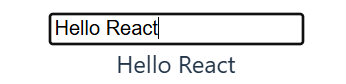
type="text" />

khi nhập giá trị vào input trên trang sẽ hiển thị đúng những gì đã nhập (cập nhật trạng thái của input khi có sự kiện onchange)

<div>

            <span>{this.state.valueInput}</span>

          </div>



Tối ưu code: ta có thể khai báo:

let {valueInput}=this.state;

Khi đó ta viết lại:

<input **value={** ~~this.state.~~valueInput **}**

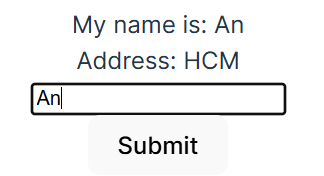
onChange={(event) => {this.handleInput(event) }}

type="text" />

và <span>{~~this.state.~~valueInput}</span>

# Xử lý form trong react

Ví dụ: khi nhập giá trị vào text box, nội dung sẽ được cập nhật ngay trên trang



Sử dụng hàm setState() cập nhật lại giá trị của biến **state**

**handleOnchangeInput** = (event) => {

        this.**setState(**{

            Name: event.target.value

        })

Gọi hàm **handleOnchangeInput tại sự kiện onChange() của text box**

 <form action="">

     <input type="text"

      onChange={(event) => this.handleOnchangeInput(event)} />

     <button>Submit</button>

  </form>

Khi click nút **Submit** thì có thể lấy được giá trị của biến **state**

* Viết hàm **handle**onSubmit

**handleOnSubmit** = (event) => {

        event.preventDefault();//ngăn việc tải lại trang

        console.log(this.state)

    }

Gọi hàm **handleonSubmit** trong sự kiện **onSubmit** của form: khi nhấn enter thì hàm

<form action=""

onSubmit={(event) => this.**handleOnSubmit**(event)}>

      <input type="text"

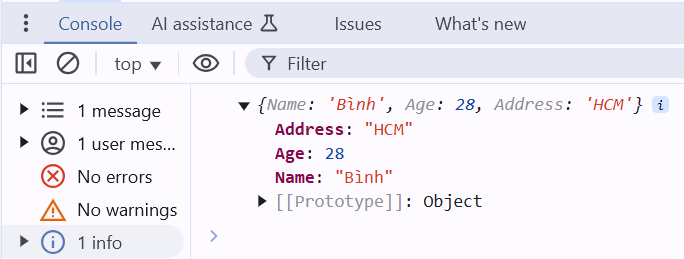
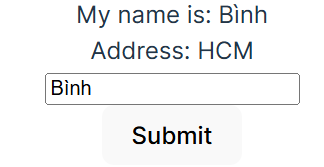
value={this.state.Name}//gán giá trị mặc định

onChange={(event) =>this.handleOnchangeInput(event)} />

    <button>Submit</button>

</form>

Khi nhập dữ liệu vào textbox và nhấn enter hoặc click nút Submit sẽ lấy được giá trị của biết state



**Lưu ý:** *gọi hàm handleOnSubmit cho sự kiện onSubmit của form sẽ bắt 2 sự kiện enter và click nút submit*

**Cài đặt react devtools**

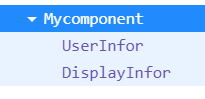
Vào google chrome 🡪nhập react devtools

Trong **Browser extension, chọn** [**Install for Chrome**](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en)

## Cách truyền dữ liệu từ cha sang con

**Hiển thị danh sách Array hoặc Object**

Ví dụ: trong đó **DisplayInfor** và **UserInfor** là con của **Mycomponent**

Tạo danh sách người dung trong **Mycomponent** dạng một Array các Object

class **Mycomponent** extends React.Component {

    state = {

        listUser: [

            {id: 1, Name: "Dung", Age: 49},

            {id: 2, Name: "Hoang", Age: 34},

            {id: 3, Name: "Chien", Age: 32},

        ]

    }

**Trong phần render() của Mycomponent**

render() {

          return (

            <div>

              <UserInfor></UserInfor>

                <hr />

<DisplayInfor listUser={this.state.listUser}></DisplayInfor>

            </div>

        );

    }

Trong component DisplayInfor, hiển thị danh sách user lên trình duyệt

## Dùng phương thức map() của Array

class DisplayInfor extends React.Component {

    render() {

        const {listUser} = this.props;//truyền từ cha sang con

//const listUser=this.props.listUser //tương đương với cách trên

        console.log(listUser)//hiển thị ds trên màn hình console

        return (

            <div>

                {listUser.map((user) => {

                    return (

<div **key={user.id}**>//mỗi phần tử trên trang hải là duy nhất

                            <div>My name is: {user.Name}</div>

                            <div>My Age: {user.Age}</div>

                            <hr />

                     </div>

                    )

                })}

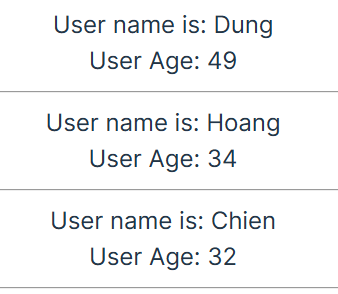
            </div>

        )

    }

}

Kết quả:



## Định dạng dữ liệu có điều kiện

Vào file **App.css**

Tạo style

.red {

  color: red;

}

.blue {

  color: blue;

}

**Vào DisplayInfor**

return (

   <div>

     {listUser.map((user) => {

       return (

       <div key={user.id} **className={user.Age < 18 ? "red": "blue"}>**

       <div>User name is: {user.Name}</div>

       <div>User Age: {user.Age}</div>

      <hr />

    </div>

    )

    })}

  </div>

)

Kết quả:



## Cách truyền hàm từ component cha sang component con

Ví dụ: Khi nhập dữ liệu từ các text box trên form và click nút Submit thì dữ liệu add vào list

Các bước thực hiện:

* Đổi tên component **UserInfor** thành **AddUserInfor**
* Trong Mycomponent, định nghĩa một function **handleAddnewUser**
* **Trong Render() của Mycomponent**
* **Thêm component AddUserInfor**

<AddUserInfor handleAddnewUser={this.handleAddnewUser}>

</AddUserInfor>

***Trong đó:***

**Props** handleAddnewUser tham chiếu đến hàm **handleAddnewUser** (*không có cặp dấu ngoặc ở cuối*)

**Hàm handleAddnewUser: them một object vào array**

**handleAddnewUser** = (userObject) => {

        this.setState({

            listUser: [userObject, ...this.state.listUser]

        })

**Trong component AddUserInfor**:

Cập nhật hàm **handleOnSubmit**

handleOnSubmit = (event) => {

        event.preventDefault();//ngăn việc tải lại trang

        //console.log(this.state)

        this.props.**handleAddnewUser**({

            id: Math.floor((Math.random() \* 100) + 1) + "user",

            Name: this.state.Name,

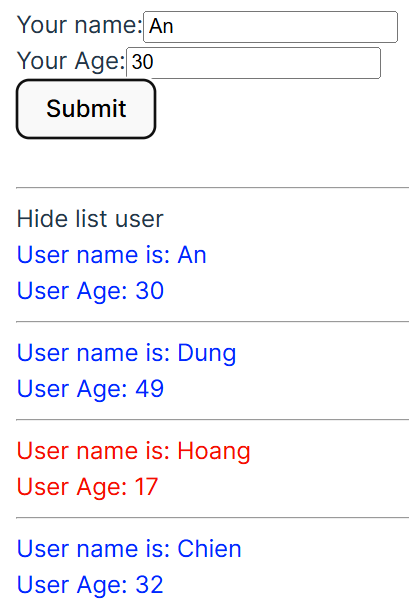
            Age: this.state.Age

        })

    }

Trong đó: gọi hàm **handleAddnewUser() với tham số truyền vào là một object với 3 thuộc tính id, name và Age**

Kết quả:



## Css trong react

* Cách 1: tạo file css
* Cách 2: viết **style inline**

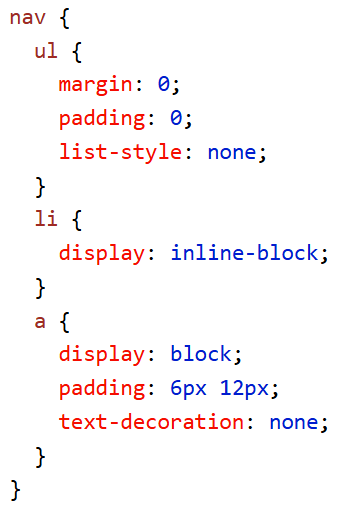
 <div style={{ color: "red", fontWeight: "bold" }}>

Trong thực tế, nếu dự án lớn thì không nên sử dụng cách này

* Cách 3: Cài them gói **sass**

Trong terminal: npm install **sass@latest**

Sass cho phép viết css lồng nhau, làm cho code css gọn hơn



## Sử dụng hình ảnh trong react

Thư mục public: trong thực tế không lưu ảnh trong public do hiệu năng không cao

Có 2 cách lưu ảnh trong react: hoặc là lưu ở DB, hoặc lưu đường link của ảnh

Ví dụ:

Trong **Vite**: logo lưu trong public

Cách đưa ảnh vào trang: dung lệnh import, wepack sẽ chuẩn bị đường link và tìm hình dán vào thuộc tính src.

Vào DisplayInfor, import hình logo:

import logo from '../../public/vite.svg'

trong phần render(), dung thẻ img

<img src={logo} alt="" />

Logo là tên của hình

Vào component.scss viết style định dạng hình

img {

        width: 200px;

        height: 200px;

    }

Một số lưu ý khi sử dụng biến trong react

* **In string và number: có thể in trực tiếp**

Ví dụ:

Trong phần **render()**

const test = 'Hello React'

Trong phần **return()**

 {test}

* **Biến kiểu object:**

const test = { Name: 'Henry', Age: 30 }

Trong phần **return()**

{console.log('check test', test)}

Hoặc

{JSON.stringify(test)}

## Delete data với Filter

Trong **Mycomponent**, tao một function **handleDeleteUser** với tham số đầu vào là **userID**

**handleDeleteUser** = (userID) => {

        let listUserClone = this.state.listUser;

// let listUserClone=[...this.state.listUser]//có thể viết theo cách này

listUserClone = listUserClone.filter(item => item.id !== userID)

        this.setState({

            listUser: listUserClone

        })

    }

Trong đó: listUserClone là danh sách sau khi xoá

Cập nhật lại thuộc tính của component DisplayInfor

<DisplayInfor listUser={this.state.listUser}

              handleDeleteUser={this.handleDeleteUser}

                ></DisplayInfor>

Trong component DisplayInfor

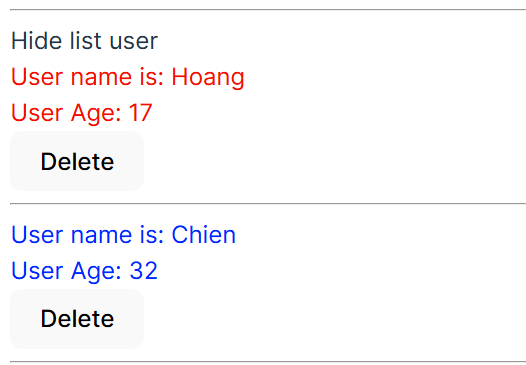
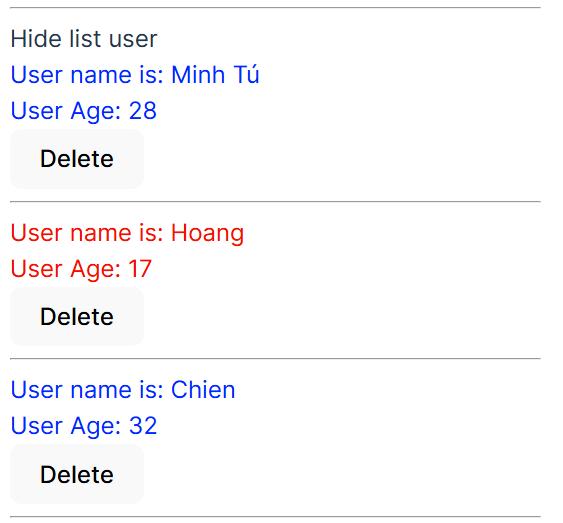
Gọi hàm handleDeleteUser tại sự kiện onclick nút Button Delete

<button onClick={() => {

        this.props.handleDeleteUser(user.id)

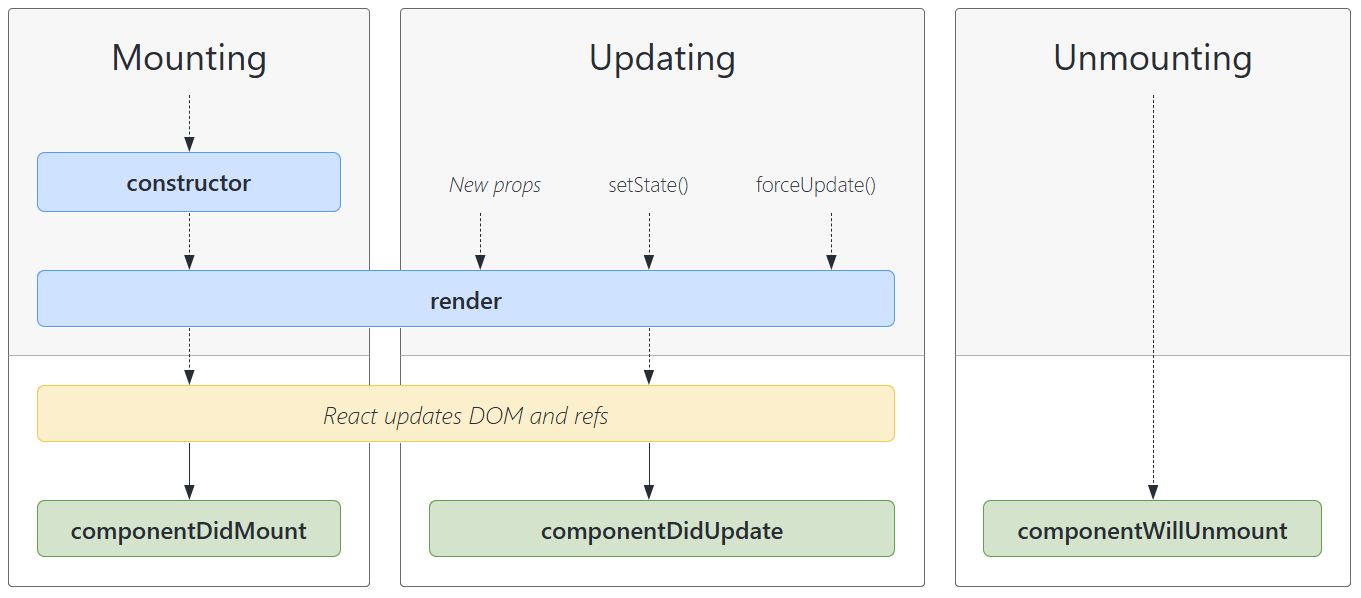
        }}>Delete</button>

Kết quả

nhấn Delete Minh Tú

# React Hiện Đại với Hook – function component

## Life cycle react diagram



## React lifecycle - Vòng đời của Component

**Ví dụ: vòng đời của game: nhấn nút Play-> chơi->dừng hoặc resum-> kết thúc**

**Component React có 3 giai đoạn:**

[**https://legacy.reactjs.org/docs/react-component.html#componentdidupdate**](https://legacy.reactjs.org/docs/react-component.html#componentdidupdate)

1. **Giai đoạn Mounting:** khi vào website thì tất cả dữ liệu được chèn vào cây DOM, Thứ tự chạy các function trong Component lần lượt là:
   * constructor():
   * getDerivedStateFromProps()
   * render()
   * componentDidMount()

Thứ tự thực hiện các hàm trong component

* **hàm constructor():**được chạy đầu tiên, có nhiệm vụ:
  + khởi tạo state của component
  + cho phép component kế thừa các props từ component cha
* getDerivedStateFromProps()
* **render():** chứa code HTML và javascript, react phải cập nhật cây DOM và hiển thị lên màn hình
* **componentDidMount():** sau khi component đã được chèn vào cây DOM, react sẽ chạy hàm componentDidMount

The render() method is required and will always be called, the others are optional and will be called if you define them**.**

**Minh hoạ thứ tự chạy các hàm trong giai đoạn mounting**

* **Vào component DisplayInfor**

Viết lại hàm **componentDidMount**

**componentDidMount()** {

        console.log(">>> Call me component did mount")

    }

**Trong constructor**

constructor(props) {

        console.log(">>> call constructor: 1")

**}**

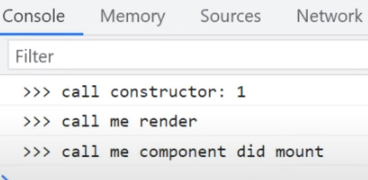
**Trong hàm render()**

render() {

        console.log(">>> Call me render");

}

**Kết quả:**

****

<https://legacy.reactjs.org/docs/react-component.html>

Cập nhật:

Ví dụ: sau khi website chạy, ta thay đổi tiêu đề của trang (cập nhật tiêu đề trang sau 3s)

 componentDidMount() {

        console.log("Call component did mount")

        setTimeout(() => {

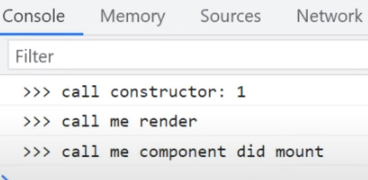
            document.title = "Hello Vite+React"

        }, 3000)

    }

Sau khi chạy các hàm constructor(), render(), sẽ chạy componentDidMount()

Nghĩa là kết quả

**** sau đó tiêu đề sẽ được cập nhật

1. **Giai đoạn Updating:**

**Giai đoạn Updating - Component được cập nhật (dữ liệu bị thay đổi)** Khi nào giai đoạn Updating xảy ra:

* Khi có sự thay đổi props từ Cha truyền xuống con
* Khi có sự cập nhật **state**, thông qua hàm **setState**

🡺 khi props thay đổi, hoặc gọi setstate thì hàm render() chạy lại, react sẽ thực hiện hàm **componentDidUpdate()**

Hàm componentDidUpdate() chỉ thực thi khi có sự thay đổi trên trang, nghĩa là phải truyền tham số cho hàm: componentDidUpdate(prevProps, prevState, snapshot)

**Thứ tự chạy:**

* Render()
* componentDidUpdate(prevProps, prevState, snapshot)

<https://legacy.reactjs.org/docs/react-component.html>

 componentDidUpdate(prevProps, prevState, snapshot) {

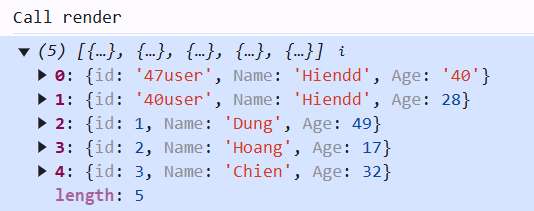
        // console.log(">>> Call me did Update");

        console.log(">>> Call me did Update", this.props, prevProps);

        //so sánh props hiện tại và quá khứ của listUser

    }

Hiện tại:



Quá khứ



Lưu ý: hàm **Render( )** và **componentDidupdate()** có thể được chạy nhiều lần, chỉ cần có sự thay đổi của **props/state** thì nó sẽ chạy lại (giống hệt một vòng lặp, cứ thay đổi là chạy lại)

1. **giai đoạn Unmounting**: khi xoá component ra khỏi cây DOM, ví dụ khi chuyển trang, được sử dụng khi webste có nhiều trang, cần điều hướng trang. (minh hoạ trong react Hook)

## Stateless/StateFull Components

* **StateFull**: (có state) Stateful component là class component, các **stateful components** có một **state** được khởi tạo trong **constructor**.

constructor(props) {

super(props);

this.state = {

count: 0

};

}

Các React component cung cấp phương thức **setState** dùng để cập nhật state.

handleCount(value) {

this.setState({count: this.state.count + value});

}

* **Stateless**: (không có state)

Một **stateless component**, còn được gọi là **functional componen**t, biểu diễn một thành phần UI mà không có **state** bên trong. một stateless component là một hàm lấy props làm đầu vào và trả về JSX (JavaScript XML) để mô tả UI.

Ví dụ:

class **DisplayInfor** extends React.Component {

    render() {

**const{ listUser } = this.props**;//nhận dữ liệu từ cha thông qua props

console.log(listUser)

return (…)

}

* Với những component stateless có cách viết thứ hai dạng function component (không có state).
* Nếu cần quản lý dữ liệu thì dung class component (statefull)

## Cách chuyển đổi từ Stateful sang Stateless Component

* Sử dụng Arrow function
* Không sử dụng hàm constructor và keyword this
* Không sử dụng hàm render, thay vào đấy là keyword **return**
* Props được truyền tự động từ cha xuống con

# React Hook

Tài liệu tham khảo hook: <https://legacy.reactjs.org/docs/hooks-intro.html>

## useState()

Khi chuyển từ class component (statefull) sang function component (stateless), nhược điểm là trong function component không chứa state, Hook sẽ giải quyết vấn đề này.

Function component không sử dụng state và từ khoá **this**, nhưng react hook cung cấp một hook đặc biệt là ***usestate***

### **useState giúp sử dụng được State cho Function Component**

**Cú pháp**: **const [ state, setState ] = useState(initValue);**

*(sử dụng cú pháp Array Destructuring - Giản lược hóa cấu trúc Object/Array)*

### **hàm useState ( )** là 1 array, trả về 2 tham số.

* + Tham số thứ nhất là ‘**tên của State’**
  + Tham số thứ 2 chính là ‘**hàm có thể cập nhật giá trị của state’** => ***dùng array destructuring để lấy ra 2 tham số này.***
* Ví dụ: **const [ name, setName ] = useState(‘Eric’).**

Ví dụ: thực hiện chức năng **Show** và **Hide** UserList sử dụng hook **usestate**

Mở component **DisplayInfor**, viết **function** **DisplayInfor** thay thế cho **class DisplayInfor**

const **DisplayInfor** = (props) => {

    const { listUser } = props; //trong function component không dùng this

    const [isShowHideListUser, setShowHideListUser] = useState(true);

    // this.state{

    //     isShowHideListUser:true//gán giá trị cho biến

    // }

    const handleShowHideListUser = () => {

        setShowHideListUser(!isShowHideListUser);

    }

    return (…)//phần này tương tự class component

}

Giải thích: Lệnh gọi hook **usestate**

const [isShowHideListUser, setShowHideListUser] = **useState**(true);

* Gọi hook ***usestate*** để có thể sử dụng state đối với component function
* Sử dụng dấu [] vì ***usestate*** trả về 2 tham số:
  + Tham số thứ 1: biến state isShowHideListUser do ta khai báo
  + Tham số thứ 2: hàm setShowHideListUser() là công cụ cập nhật giá trị cho biến **state**
* Khai báo **const**: biến state chỉ thay đổi khi gọi hàm thông qua Hàm của react setShowHideListUser()

Gọi hàm thông qua sự kiện **onclick** của thẻ span

<div>

    <span **onClick={() => handleShowHideListUser()}>**

    {isShowHideListUser === true? "Hide list User":"Show list User"}

</span>

</div>

<https://legacy.reactjs.org/docs/hooks-state.html>

Bài tập: Yêu cầu chuyển các class sang function

* **Component Mycomponent**

// class Mycomponent extends React.Component {

//     state = {

//         listUser: [

//             { id: 1, Name: "Dung", Age: 49 },

//             { id: 2, Name: "Hoang", Age: 17 },

//             { id: 3, Name: "Chien", Age: 32 },

//         ]

//     }

**Thay bằng function**

const Mycomponent = (props) => {

    const [listUser, setlistUser] = useState(

        [

            { id: 1, Name: "Dung", Age: 49 },

            { id: 2, Name: "Hoang", Age: 17 },

            { id: 3, Name: "Chien", Age: 32 },

        ]

    )

**Bài Tập: Sử Dụng useState Hook**

**Yêu cầu**: Chuyển đổi Class Component sang sử dụng Function Component với useState Hook

Tài liệu về useState Hook: <https://reactjs.org/docs/hooks-state.html>

**Giải Bài Tập useState Hook**

**Đối với React Hook:**

* Sử dụng **Arrow Function** (Function component)
* **Không sử dụng keyword this** và hàm **setState**
* Mỗi 1 state Hook sẽ là 1 biến độc lập, như vậy, sẽ chia nhỏ các state của component ra, ngược hoàn toàn với React Class (biến state là object chứa toàn bộ state của component)

**Các bước thực hiện:**

Hiệu chỉnh các hàm handleAddnewUser, handleDeleteUser theo cách viết Hook

const handleAddnewUser = (userObject) => {

        setlistUser([userObject, ...listUser])

const handleDeleteUser = (userID) => {

        let listUserClone = listUser;

listUserClone = listUserClone.filter(item => item.id!== userID)

setlistUser(listUserClone);

    }

Xoá từ khoá this trong lệnh gọi component trong phần return

<AddUserInfor handleAddnewUser={handleAddnewUser}></AddUserInfor>

                <hr />

<DisplayInfor listUser={listUser}

              handleDeleteUser={handleDeleteUser}></DisplayInfor>

* **Component AddUserInfor**

const AddUserInfor = (props) => {

    // state = {

    //     Name: 'Hien',

    //     Age: 28,

    //     Address: 'HCM'

    // }

    //chuyển sang hook

    const [Name, setName] = useState('');

    const [Age, setAge] = useState('');

    const [Address, setAddress] = useState('IUH');

const handleOnchangeInput = (event) => {

          setName(event.target.value)

    }

    const handleOnchangeAge = (event) => {

        setAge(event.target.value)

    }

    const handleOnSubmit = (event) => {

        event.preventDefault();//ngăn việc tải lại trang

        props.handleAddnewUser({

            id: Math.floor((Math.random() \* 100) + 1) + "user",

            Name: Name,

            Age: Age

        })

    }

Xoá tất cả các từ khoá this.state trong form

 <form action="" onSubmit={(event) => **handleOnSubmit(event)}>**

                <label htmlFor="">Your name:</label>

                <input type="text" value={Name}

                 onChange={(event) => **handleOnchangeInput(event)}** />

                <br />

                <label htmlFor="">Your Age:</label>

                <input type="text"

                    onChange={(event) => **handleOnchangeAge(event)**}

                    value={Age} />

                <button>Submit</button>

 </form>

## useEffect hook

[**https://legacy.reactjs.org/docs/hooks-effect.html**](https://legacy.reactjs.org/docs/hooks-effect.html)

* + Sử dụng life cycle với **function compoment**
  + Chuyển đổi các phương thức trong vòng đời của react sang sử dụng hook

*Làm thế nào để sử dụng componentDidMount và componentDidUpdate khi không kế thừa các lớp của react component?*

**Khi dung** react Hook thì **useEffect Hook** bao gồm **componentDidMount** của react class.

**Ví dụ: sử dụng useEffect thay đổi tiêu đề của trang**

**Trong component DisplayInfor**

**useEffect**(() => {

        setTimeout(() => {

            document.title = "Hello React hook";

        }, 3000);

       console.log("call useeffect");

    });

Mỗi lần tương tác với giao diện (render) thì hàm useEffect luôn được chạy, có 2 cách để render lại giao diện. **Nếu muốn chạy 1 lần**: trong hàm userEffect có tham số thứ 2 là **[]**, thì hàm chỉ chạy 1 lần

useEffect(

        () => {

            setTimeout(() => {

                document.title = "Hello React hook";

            }, 3000);

            console.log("call useeffect");

        }, **[]**);

### ***Lưu ý 1:***

useEffect(

    () => {//code logic here},

**[]**

    );

Khi truyền vào mảng rỗng (tham số thứ 2 của hàm useEffect), thì hàm **useEffect** sẽ **chạy đúng 1 lần**, sẽ có tác dụng như hàm **ComponentDidMount** của React Class.

### ***Lưu ý 2:***

useEffect(

    () => {//code logic here},

        [tên\_biến]

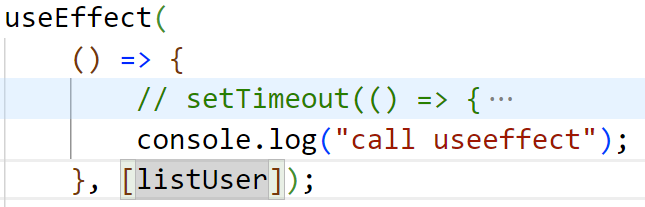
    );

* Khi truyền vào tên biến (đối với tham số thứ 2), thì hàm **useEffect** sẽ được chạy ‘bất cứ khi nào giá trị của ‘biến quan sát’ bị thay đổi.
* Như vậy, lúc này, hàm **useEffect** sẽ có tác dụng như hàm **ComponentDidUpdate,** giúp chúng ta có thể so sánh giá trị quá khứ và giá trị hiện tại của biến (props/state).
* Nếu giá trị của biến thay đổi, khối code logic của hàm useEffect sẽ được chạy.

Hàm **componentDidUpdate**

* React hook không lấy được giá trị quá khứ của một biến, trừ khi phải gán lại.
* Nếu muốn so sánh sự thay đổi của một biến, thì truyền trực tiếp vào mảng dependency **[]**

Ví dụ: khi xoá hết danh sách, trang sẽ hiển thị thông báo lỗi

Ta có một props ListUser, cần theo dõi sự thay đổi của biến này, ta đưa nó vào mảng dependency **[]**

useEffect(

        () => {

            if (listUser.length === 0)

                alert("You deleted all users")

            console.log("call useeffect");

        }, [listUser]);

* Có thể định nghĩa nhiều **useEffect** trong một component
* Khi dùng **useEffect** tương đương với 2 hàm componentDidMount và componentDidUpdate trong class component

## Sử dụng hook trong React

* React Hook ra đời vào năm 2018, giúp **Function Component** có đầy đủ tính năng như **Class Component**.
* **Hook** = Cái móc câu. Nếu như **Function Component** là ‘con cá’, thì Hook sẽ ‘móc vào’ Component, giúp component có thêm cái gì đấy (có thêm state :v)
* React Hook chỉ có thể sử dụng với React version >= 16.8

**React Class tồn tại vấn đề sau:**

* Viết code theo OOP (lập trình hướng đối tượng). Cần hàm tạo (constructor), khai báo tate, khai báo lifecycle…
* Sử dụng keyword **this**
* Với 1 component có nhiều giá trị **state**, việc tách riêng ‘một vài state’ để tái sử dụng là điều không thể.

**React Hook**

* giúp giải quyết các vấn đề trên,
* giúp component của React, chỉ đơn giản là Function.
* Dùng Hook, thay vì dùng Class Component vì:
  + Hook ra đời sau Class, và đang là xu hướng.
  + Facebook chuyển dịch thì chúng ta chuyển dịch theo.
  + Các dự án mới đa phần phát triển với Hook, vì đây là xu hướng chuyển dịch.

Tài liệu tham khảo về React Hook: <https://reactjs.org/docs/hooks-intro.html>

* + Hook chia nhỏ dữ liệu để có thể tái sử dụng

Ví dụ:

    // state = {

    //     Name: 'Hien',

    //     Age: 28,

    //     Address: 'HCM'

    // }

    //chuyển sang hook

    const [Name, setName] = useState('');

    const [Age, setAge] = useState('');

    const [Address, setAddress] = useState('IUH');

# Setup Dự Án Backend (Dùng với Docker)

Hướng Dẫn Cài Đặt Dự Án Backend (Dùng Docker)

Download file cài đặt Docker: [https://drive.google.com/file/d/1a92o\_EJa1h5Ciz99lK1borpPqz-](https://drive.google.com/file/d/1a92o_EJa1h5Ciz99lK1borpPqz-XStvL/view?usp=sharing)

[XStvL/view?usp=sharing](https://drive.google.com/file/d/1a92o_EJa1h5Ciz99lK1borpPqz-XStvL/view?usp=sharing)

## Về Cách Học Docker Cho Beginners

Học nhanh docker dành cho beginners tại đây:

<https://www.youtube.com/playlist?list=PLncHg6Kn2JT4EcU8eTdmojGO1StFDILe2>

## Tạo tài khoản Docker Hub

* Docker giúp chúng ta ‘đóng gói ứng dụng’, tạo môi trường để chạy project, và hạn chế tối đa việc phải cài đặt công cụ.
* Docker Hub là nơi lưu trữ file chạy của Docker (images), tương tự như Github để lưu code.

Tìm hiểu về Docker cơ bản, các bạn có thể xem tại đây:

<https://www.youtube.com/playlist?list=PLncHg6Kn2JT4kLKJ_7uy0x4AdNrCHbe0n>

* **Các bước tạo tài khoản trên Docker Hub:**

1. Truy cập <https://hub.docker.com/signup> để tạo tài khoản
2. Docker Id chính là tên username dùng để hiển thị. Sử dụng Gmail để đăng ký
3. Khi xác nhận Email, nếu không thấy Email xác nhận ở Inbox (Hộp thư đến), các bạn nhớ check trong thư mục Spam nhé ^^

## Cài đặt Docker Desktop

* Video hướng dẫn cài đặt: [https://www.youtube.com/watch?v=Ic48-](https://www.youtube.com/watch?v=Ic48-WLhtHg&list=PLncHg6Kn2JT4kLKJ_7uy0x4AdNrCHbe0n&index=2)
* [WLhtHg&list=PLncHg6Kn2JT4kLKJ\_7uy0x4AdNrCHbe0n&index=2](https://www.youtube.com/watch?v=Ic48-WLhtHg&list=PLncHg6Kn2JT4kLKJ_7uy0x4AdNrCHbe0n&index=2)
* Download Docker Desktop và VM Linux mình sử dụng trong video tại đây:
* [https://drive.google.com/drive/folders/1i2tzQRlTZ7shibrAXzvWAfhoSatvqxH8?usp=sh](https://drive.google.com/drive/folders/1i2tzQRlTZ7shibrAXzvWAfhoSatvqxH8?usp=sharing)
* [aring](https://drive.google.com/drive/folders/1i2tzQRlTZ7shibrAXzvWAfhoSatvqxH8?usp=sharing)

## Hướng Dẫn Cài Đặt Dự Án Backend (Dùng Docker)

Download file cài đặt Docker: [https://drive.google.com/file/d/1a92o\_EJa1h5Ciz99lK1borpPqz-](https://drive.google.com/file/d/1a92o_EJa1h5Ciz99lK1borpPqz-XStvL/view?usp=sharing)

[XStvL/view?usp=sharing](https://drive.google.com/file/d/1a92o_EJa1h5Ciz99lK1borpPqz-XStvL/view?usp=sharing)

## DBeaver - Kết Nối Database Postgres

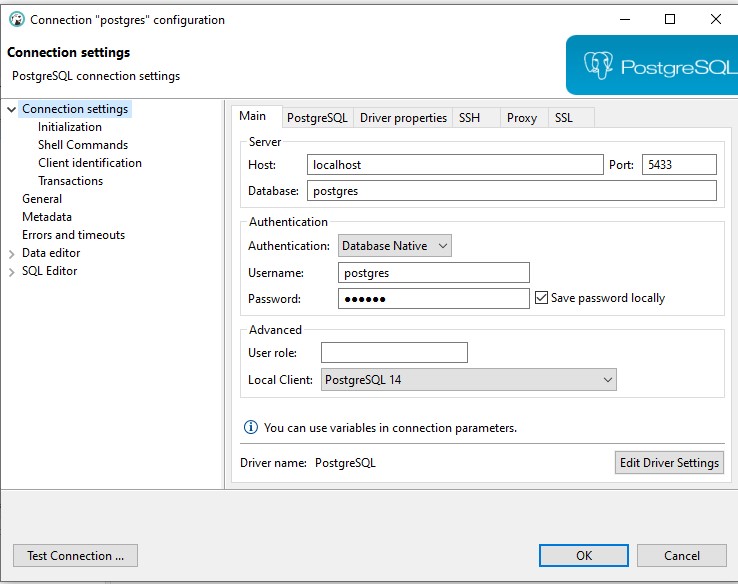
## Download DBeaver: (dành cho Windows)

[https://drive.google.com/file/d/1Tf1wIf-G6gdVu5wWMze1ULNJbht-](https://drive.google.com/file/d/1Tf1wIf-G6gdVu5wWMze1ULNJbht-JlJF/view?usp=sharing)

[JlJF/view?usp=sharing](https://drive.google.com/file/d/1Tf1wIf-G6gdVu5wWMze1ULNJbht-JlJF/view?usp=sharing)

## Kết nối Database Postgres:

* Các tham số mặc định của database, khai báo tại file docker-compose khi cài đặt với Docker.
* Mặc định:
  + Port: 5433
  + Database: postgres
  + Username: root
  + Password: 123456
* Lưu ý: trong lần đầu tiên sử dụng, có thể cần download Driver, các bạn cứ nhấn ‘Yes/Ok’ (nếu có) để phần mềm tự động download nhé



## Phân tích Models Database:

Models:

* Participant: quản lý users sử dụng hệ thống
* Quiz: quản lý các bài thi
* Question: quản lý câu hỏi của bài Quiz
* Answer: quản lý câu trả lời cho từng câu hỏi (Question)

## PostMan - Test APIs Backend

Download Postman: <https://www.postman.com/downloads/>

Link download for Windows: [https://drive.google.com/file/d/1MTndb2SdL9J73TFchSYXF89Xm8yNmzK/view?usp=sharing](https://drive.google.com/file/d/1MTndb2SdL9J73TFc-hSYXF89Xm8yNmzK/view?usp=sharing)

# Điều Hướng Trang với React Router v6

*Điều hướng trang trong ứng dụng React sử dụng React-router*

## Setup Bootstrap 5 & React Router Dom v6

### **Cài đặt React Router Dom :**

**npm install --save-exact react-router-dom@6.3.0**

Tài liệu chi tiết: <https://reactrouter.com/>

* **Cài đặt Bootstrap 5 sử dụng với React**

**npm install --save-exact react-bootstrap@2.4.0 bootstrap@5.2.0**

Sau khi cài đặt thư viện, cần import CSS của Bootstrap vào file index.js

import 'bootstrap/dist/css/bootstrap.min.css';

Tài liệu chi tiết: <https://react-bootstrap.github.io/>

Tại sao lại cài đặt thư viện **react-bootstrap** để dùng bootstrap, mà không đơn giản chỉ cần cài đặt mình thư viện bootstrap ?

* Nếu chỉ cài thư viện Bootstrap, thì với các Component cần hiệu ứng, ví dụ như Modal, Dropdown… sẽ không chạy được, vì phần hiệu ứng do JQuery và Javascript phụ trách
* Thư viện **React-Bootstrap** giúp giải quyết vấn đề trên, đồng thời, tối ưu code để dùng tốt với React.

## Design Header với Bootstrap Navigation

* Tài liệu Bootstrap NavBar:

<https://react-bootstrap.netlify.app/docs/components/navbar/>

## Điều Hướng Trang với Links

* Tài Liệu: <https://reactrouter.com/docs/en/v6/getting-started/tutorial#connect-the-url>

## Nested Routes

* Tài liệu:<https://reactrouter.com/docs/en/v6/getting-started/tutorial#nested-routes>
* Để share ‘dữ liệu HTML’ (phần dùng chung) giữa các **Routes**, chúng ta sử dụng **component Outlet**.
* Outlet cần được định nghĩa ở component Cha. Từ đấy, khi render giao diện giữa các route con, phần component con sẽ được render vào phần Outlet đã định nghĩa.

## Active Link - NavLink

* Tài liệu:<https://reactrouter.com/docs/en/v6/getting-started/tutorial#active-links>
* **Component NavLink khác gì Link ?**
* **NavLink** có chức năng ‘giống hệt’ Link (giúp điều hướng trang), tuy nhiên, cung cấp thêm công cụ để chúng ta CSS cho **‘active link’** - link chúng ta ‘đã click trước đó’.

**NavLink = Link + CSS**

* *Mặc định, khi user click vào 1* ***NavLink****, thì ngay lập tức* ***NavLink đó sẽ được ‘add - thêm vào’ class có tên là ‘active’***

## Index Routes

* Tài *liệu*:<https://reactrouter.com/docs/en/v6/getting-started/tutorial#index-routes>
* **Index routes** không có thuộc tính ‘path’, vì đơn giản, nó dùng lại path của parent. Thay vào đó, nó cần **props ‘index’**

<Route

    index element={

        <main style={{ padding: "1rem" }}>

            <p>Select an invoice</p>

        </main>

    }

/>

* **Index route**s sẽ được gọi **tới**, khi tất cả ‘route con’ còn lại không **match**, thì nó sẽ được dùng.

## Design Homepage

### **Nguồn tài liệu tham khảo**

* Link website clone: <https://www.typeform.com/>
* Link download video: [https://drive.google.com/file/d/1APqoC\_9VKHnHqmIEwImkqjedgKv-](https://drive.google.com/file/d/1APqoC_9VKHnHqmIEwImkqjedgKv-S1Bs/view?usp=sharing)
* [S1Bs/view?usp=sharing](https://drive.google.com/file/d/1APqoC_9VKHnHqmIEwImkqjedgKv-S1Bs/view?usp=sharing)

### **Lưu ý về sử dụng Video với React**

* Import video vào Component như cách sử dụng với image
* Sử dụng html tag video theo cú pháp:

<video autoPlay muted loop>

    <source src= ‘... ‘  type = ‘...’/>

</video>

**Trong đó:**

* **loop (vòng lặp) :** giúp video chạy xong, nó sẽ tự động chạy lại (lặp đi lặp lại, vô hạn).
* **autoPlay(tự động chạy) :** giúp video tự động phát khi load vào website. Ở đây, video sẽ được phát mà không cần nhấn nút ‘run’.
* **muted (câm) :** giúp tắt tiếng (audio) của video. Bắt buộc phải sử dụng thuộc tính này kèm với autoPlay để cho tính năng autoPlay có thể hoạt động.

### **Lý do phải dùng song song muted + autoPlay là vì:**

* trước 2018, các website thường tự động chạy video (có âm thanh) và tự động chạy file nhạc khi người dùng vào website, đặc biệt là các website quảng cáo.
* Như vậy, tạo sự khó chịu cho người dùng khi không muốn xem/nghe file mà vẫn bị bắt phải làm.
* Vì vậy, sau tháng 4/2018, với các version mới của Google Chrome, tính năng **autoPlay file nhạc (audio) bị bỏ.** Tương tự với video có audio (video không có tiếng vẫn dùng được autoPlay). Đấy chính là lý do tại sao cần dùng autoPlay + muted.
* Nếu muốn chạy file nhạc (audio), hoặc video có tiếng, bắt buộc người dùng phải nhấn nút ‘run/play’ để thực hiện, không thể dùng javascript/html để tự động chạy 2 loại file trên.
* Chi tiết xem tại: <https://developer.chrome.com/blog/autoplay/>

## Design New Header

### **Tạo Header với background transparent (trong suốt):**

**background-color: rgba(0, 0 , 0, 0 )**

### Về rgba và rgb xem tại đây: <https://www.w3schools.com/css/css_colors_rgb.asp>

## Câu lệnh git checkout

### <https://git-scm.com/docs/git-checkout>

**Check code tại 1 branch (đã có sẵn)**

**git checkout branch\_name**

**Check code sang 1 branch hoàn toàn mới (chưa có)**

**git checkout -b branch\_name**

**Check code tại 1 commit (1 mốc thời gian trong quá khứ)**

**git checkout commit\_hash**

## Design Admin SideBar

### **Tích hợp thư viện**

**npm install --save-exact react-pro-sidebar@0.7.1**

**Nếu cài đặt bị lỗi, thử câu lệnh này:**

**npm install --save-exact react-pro-sidebar@0.7.1 --legacy-peer-deps**

Link github: <https://github.com/azouaoui-med/react-pro-sidebar>

Link demo (preview): <https://azouaoui-med.github.io/react-pro-sidebar/>

### **Để sử dụng icon, cài đặt thêm thư viện sau:**

**npm install --save-exact react-icons@4.4.0**

# Setup Axios, React Toastify, React Paginate

## Tài liệu về React-Icons:

Github: <https://github.com/react-icons/react-icons>

Trang chủ: <https://react-icons.github.io/react-icons/>

Download Sidebar (code của mình): [https://drive.google.com/file/d/1JkqrgmBERd8s9E44eUYYsymtUQK3Ld1D/view?usp=s haring](https://drive.google.com/file/d/1JkqrgmBERd8s9E44eUYYsymtUQK3Ld1D/view?usp=sharing)

## Cài đặt thư viện cần thiết:

**npm install --save-exact axios@0.27.2 react-toastify@9.0.7 react-paginate@8.1.3**

# CRUD Users - Thêm/Hiển Thị/Cập Nhật/Xóa Người Dùng

*Luyện tập chức năng thêm, sửa, xóa người dùng với React và APIs của Backend*

## Modal Thêm Mới Người Dùng

* Tài liệu:
* Modal Reactrap: [https://react-bootstrap.github.io/components/modal/#livedemo](https://react-bootstrap.github.io/components/modal/#live-demo)
* Form Bootstrap:
* <https://getbootstrap.com/docs/5.0/forms/layout/#gutters>

## State Hóa Modal Add New User

### **Lưu ý khi CSS cho Modal:**

* Mặc định, Modal không được render vào div ‘root’, mà thường chèn vào trước tag </body>
  + khi CSS, nếu CSS theo kiểu cha lồng con của app React, nó sẽ không hoạt động.
  + cách dễ nhất để fix bugs trên là đặt className cho Modal, sau đấy CSS global.
* Phải đặt className, tránh việc CSS trực tiếp vì nó có thể làm vỡ giao diện, do bị trùng (ghi đè) CSS của nhau.

## CSS Image Fit Div:

<https://stackoverflow.com/a/3029434>

### **Customize Upload File button**

Sử dụng:

<label htmlFor=”your-input-id” > </label>

<input hidden id=”your-input-id” />

### **Upload File với React**

* Để lấy file thông qua event, sử dụng event.target.files[0]
* khi <input type=’file’/>, sử dụng event.target sẽ trả ra FileList (một array chứa File), thành ra khi upload 1 file (single), chúng ta cần lấy phần tử đầu tiên của FileList (files[0])

**Lưu ý: File là 1 loại dữ liệu đặc biệt của javascript**

* **Tài liệu về File với javascript:**
* <https://developer.mozilla.org/en>[US/docs/Web/API/File\_API/Using\_files\_from\_web\_applications](https://developer.mozilla.org/en-US/docs/Web/API/File_API/Using_files_from_web_applications)

### **Preview Image với React**

* File ảnh upload được lấy dưới dạng File object => không thể hiển thị hình ảnh dưới dạng File object, cần phải sử dụng URL.createObjectURL( )
* Tài liệu:
* <https://developer.mozilla.org/en-US/docs/Web/API/URL/createObjectURL>

# API Thêm Mới Người Dùng (CREATE)

## API thêm mới user:

* POST <http://localhost:8081/api/v1/participant>
* Tài liệu Axios với Form Data: [https://github.com/axios/axios#using-multipartform-dataformat](https://github.com/axios/axios#using-multipartform-data-format)

## Validate data và React Toastify

* Validate Email:<https://stackoverflow.com/a/46181>
* Tài liệu React Toastify: <https://fkhadra.github.io/react-toastify/introduction/>

toast.info("Lorem ipsum dolor"); // same as toast(message, {type: "info"});

toast.error("Lorem ipsum dolor") toast.success("Lorem ipsum dolor") toast.success("Lorem ipsum dolor", { theme: "colored" })

toast.warn("Lorem ipsum dolor") toast.warn("Lorem ipsum dolor", { theme: "dark"

})

## API Services

* **Cách đặt tên thư mục:**
* Thông thường, trong các dự án, thư mục **Services** thường dùng để ám chỉ các logic liên quan tới gọi APIs. Nên tách riêng phần này khỏi component để dễ quản lý code.
* Thư mục **Utils** (là viết tắt của utility - hữu ích), nơi chứa những function dùng chung (helper)

## Axios Customize

* **Tạo Instance Axios:**

<https://github.com/axios/axios#creating-an-instance>

* **Tạo customize của Axios**, để trước khi gọi API, chúng ta sẽ chỉnh sửa thư viện axios cho phù hợp với nhu cầu. Hiểu nôm na, những phần nào dùng chung giữa các lần gọi **API với Axios**, chúng ta sẽ ‘cấu hình’ tại 1 nơi duy nhất.

Ví dụ:

* mỗi lần gọi API, chúng ta cần truyền **access\_token** vào APIs. Thay vì làm thủ công, là **khi nào gọi APIs thì thêm trường access\_token**, chúng ta sẽ cấu hình ở file **axios\_customize, và thêm access\_token** ở đó (1 nơi duy nhất).
* Như vậy, mặc định, chúng ta sẽ không cần truyền access\_token khi gọi APIs nữa.
* **Instance**, hiểu đơn giản là 1 object (thực thể/đối **tượng**). Giống như khi học lập trình hướng đối tượng, chúng ta có:

**const car = new Car (...) thì ở đây là const instance = axios.create(...)**

* **Tạo Interceptor:**

<https://github.com/axios/axios#interceptors>

* You can intercept requests or responses before they are handled by then or catch. *(chúng ta có thể can* thiệp *vào request và response trước khi client nhận được phản hồi)*
* Mã phản hồi của backend (http status code) : [https://developer.mozilla.org/en-](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)
* [US/docs/Web/HTTP/Status](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)
* Interceptor (người can thiệp) được sử dụng để customize request/response.
* Trước khi request (từ client gửi lên server), Interceptor có thể can thiệp (ví dụ như việc thêm acceess\_token ở trên)
* Trước khi response (từ server gửi về client), Interceptor cũng có thể can thiệp.
* Mặc định, khi sử dụng axios, axios sẽ trả về response theo dạng sau:
* <https://github.com/axios/axios#response-schema>
  + response có nhiều hơn những cái chúng ta quan tâm => dùng interceptor để can thiệp và lấy cái chúng ta muốn

# Hiển Thị Danh Sách Users (READ)

## APIs lấy danh sách người dùng

GET <http://localhost:8081/api/v1/participant/all>

### **Tài liệu**

* Về cách sử dụng hàm map
* Table Bootstrap 5: <https://getbootstrap.com/docs/5.0/content/tables/>
* Button Bootstrap 5: <https://getbootstrap.com/docs/5.0/components/buttons/>

## Cập Nhật Danh Sách Users Khi Thêm Mới Thành Công

### **Lift-up state**

### **Tài liệu**: <https://reactjs.org/docs/lifting-state-up.html>

* Lift-up (nâng lên), ở đây chúng ta hay gọi là ‘lifting state up’, một kỹ thuật hay được dùng để ***chia sẻ data trong ứng dụng React***.

* Cụ thể, chúng ta sẽ đưa ‘***state của component con’ sang cho ‘component cha’*** quản lý. ***Component con sẽ dùng data thông qua ‘props truyền từ cha xuống’***

* Như vậy, ‘lift-up state’ ám chỉ việc đưa state của ‘component con’ lên ‘component cha’. ***Do React thiết kế theo mô hình cha-con, data chảy từ trên (cha) xuống dưới (con), nên gọi là lift-up (đẩy lên/nâng lên).***

### **Khi nào cần lift-up state ?**

* Chúng ta cần ***lift-up state*** khi ‘**component cha có nhiều con’,** và ***các con của nó có quan hệ với nhau***.
* Nghĩa là: chúng ta muốn 1 hành động xảy ra ở ‘1 đứa con này’, sẽ cần làm gì đấy ‘ở 1 đứa con khác’, do đó, sẽ nhờ ‘cha của các con’ làm (thông qua việc *lift-up state*). Cha có thể nói chuyện với con thông qua ‘**props’**, còn các con không thể nói chuyện với nhau.

Ví dụ:

<Parent>

<Child1>...</Child1>

<Child2> …</Child2>

s</Parent>

## Design Modal Update User

* Cài đặt thư viện lodash: **npm install --save-exact lodash@4.17.21**
* Hiển thị Image base64: <https://stackoverflow.com/a/42399865>

### **Lưu ý về code:**

* **Component** sẽ được re-render (vẽ lại giao diện) khi có sự thay đổi của state hoặc props
* **Hàm useEffect của react Hook sẽ bằng các hàm sau của React Class**:

1. **ComponentDidMoun**t (component đã được render và chèn vào DOM)
2. **ComponentDidUpdate** (có component đã được cập nhật, thông qua sự thay đổi của props/state)
3. **ComponentWillUnmoun**t của React Class

* Khi useEffect === ComponentDidMount, chúng ta sẽ dùng nơi đây để gọi API. Lý do chúng ta gọi API nơi **này**, vì lúc này DOM đã sẵn sàng (đã có HTML), nên việc thao tác với DOM sẽ không có lỗi. (tránh bugs không cần thiết).
* Lúc **này**, chúng ta sẽ dùng cú pháp:
* **useEffect**( ( ) => {....}, [ ] ); để ý là hàm useEffect chúng ta truyền vào 1 mảng rỗng [ ] , mục đích để cho chạy duy nhất 1 lần.

* Khi **useEffect** === ComponentDidUpdate, chúng ta dùng nơi này để xử lý logic khi có bất kỳ sự thay đổi nào của ‘biến quan sát - state/props’.

Cú pháp:

* **useEffect**( ( ) => {....}, [biến\_quan\_sát ] ); biến quan sát có thể là state của Component hiện tại, hoặc là props truyền từ cha xuống.

* Ở **đây**, mỗi khi ‘biến quan sát’ được thay đổi giá trị, ngay lập tức hàm useEffect sẽ được
* chạy, như vậy hàm useEffect lúc này sẽ được chạy nhiều lần (> 1)

## API Cập Nhật User (UPDATE)

### **APIs cập nhật người dùng**

PUT [http://localhost:8081/api/v1/participant](http://localhost:8081/api/v1/participant/all)

## Bài tập: Chức Năng Xem Chi Tiết User

Yêu cầu:

* Nhấn vào button View, mở Modal Xem Chi Tiết Người Dùng
* Thông tin hiển thị giống hệt như Modal Update User, tuy nhiên, tất cả các trường sẽ disabled (không cho sửa)
* Modal này không cần nút Save, chỉ cần nút Close (mục đích chỉ cho user xem thông tin, không thể thao tác sửa đổi cập nhật)

## Design Modal Delete User

Modal Bootstrap: <https://react-bootstrap.github.io/components/modal/#live-demo>

## APIs Delete User (DELETE)

APIs xóa người dùng

DELETE [http://localhost:8081/api/v1/participant](http://localhost:8081/api/v1/participant/all)

# Hiển Thị Danh Sách User Phân Trang (Paginate)

* Thư viện React-paginate: <https://www.npmjs.com/package/react-paginate>
* Github: <https://github.com/AdeleD/react-paginate>
* Link ví dụ: <https://codepen.io/monsieurv/pen/abyJQWQ>

## APIs lấy danh sách người dùng phân trang:

* GET <http://localhost:8081/api/v1/participant?page=1&limit=2>
* (Truyền động tham số page và limit)
* Về dấu nháy chéo backticks `, xem tại video #6 Template literals (Template strings) -
* Dấu nháy chéo ` backticks

# Design Login

* Về thuộc tính ‘forcePage’ xem tại đây: <https://github.com/AdeleD/react-paginate#props>
* useNavigate Hook : <https://reactrouter.com/docs/en/v6/hooks/use-navigate>

## API Login - Đăng nhập APIs login:

* POST <http://localhost:8081/api/v1/login>
* HTML entity: <https://www.w3schools.com/html/html_entities.asp>

## Bài tập: Chức năng Register - Đăng Ký Người Dùng APIs register:

POST <http://localhost:8081/api/v1/register>

# Chức Năng Register

* File Register (component + css) : [https://drive.google.com/file/d/1WM\_bA-](https://drive.google.com/file/d/1WM_bA-BIEbY18wNtBIka_yuh9GnsA-y3/view?usp=sharing)
* [BIEbY18wNtBIka\_yuh9GnsA-y3/view?usp=sharing](https://drive.google.com/file/d/1WM_bA-BIEbY18wNtBIka_yuh9GnsA-y3/view?usp=sharing)
* CSS position: <https://www.w3schools.com/css/css_positioning.asp>

# Quản Lý State Application với Redux

*Tìm hiểu các quản lý global state của ứng dụng React với thư viện Redux*

# Why Redux ? Tại Sao Lại Cần Redux

## Vấn đề gặp phải với React (không dùng Redux)

* Nguyên tắc chia sẻ data: Truyền từ cha (parent) xuống con (child) thông qua props. Truyền từ trên xuống dưới (top to bottom), thành ra component cha sẽ nằm ngoài bọc component con.
* Muốn chia sẻ data giữa 2 component cùng cấp, cách đơn giản nhất là lift-up state, đưa data cho component cha quản lý (xem video #62 về lift-up state)
* Chỉ có thể dùng props khi và chỉ khi 2 component cùng tồn tại trên màn hình, tức là có tồn tại component cha và con trên DOM

## Tại sao dùng Redux

* Redux giúp quản lý trạng thái ứng dụng React (managing state) và không phụ thuộc vào nguyên tắc chia sẻ data (từ cha xuống con)
* Redux khác gì so với Context API: <https://reactjs.org/docs/context.html>
* Về chức năng, Redux không khác gì Context API, giúp chia sẻ data giữa các component không có quan hệ với nhau.
* Về hiệu năng, Redux vượt trội, vì 2 lý do chính.
  + 1.Redux hỗ trợ devtool dành cho lập trình viên, giúp việc kiểm tra data lưu trong redux cực kỳ thuận tiện
  + 2.Với context API, mỗi lần nó thay đổi, thì giao diện thay đổi. Không thể pause (dừng lại việc re-render). Với Redux, bạn có thể làm được, có thể quyết định xem có cho component re-render khi redux thay đổi hay không ?
  + 3.Thư viện sinh ra là có lý do của nó, nếu không, nó không tồn tại. Context API là level thô sơ, còn Redux là level đã cài tiến từ Context API :v

## Lưu ý

* Với redux, hiện nay có 2 cách sử dung là: sử dụng redux thuần hoặc redux/toolkit Trong khóa học này, mình sử dụng redux thuần.

* Về redux/toolkit, cũng như chuyên sâu về Redux, mình đã có 1 khóa học riêng về nó: <https://www.udemy.com/course/hoidanit-redux-and-redux-toolkit-ultimate/>

## #74 Store - Lưu Trữ Data Redux

* Redux diagram: [https://redux.js.org/tutorials/fundamentals/part-2-concepts-dataflow#redux-application-data-flow](https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow#redux-application-data-flow)

* Store: (nơi lưu trữ) data Redux . Data sẽ được lưu bên trong thư viện Redux (với
* google Chrome thì sử dụng Redux devtool để xem data Redux - video #18)
* Khi chạy ứng dụng React, thì Redux đã được cấu hình để chạy song song với ứng dụng React, cụ thể như sau:
* Tất cả logic liên quan về Redux được viết trong thư mục src/redux.
* Khi chạy ứng dụng React lên (file index.js), đã nhúng Store của Redux vào, thông qua thư viện react-redux.

File index.js:

<Provider store={reduxStore} > <= nạp data Redux tại đây

<App/>

</Provider>

* File store: quy định các thông tin sẽ lưu (thông qua reducer) và cấu hình cho redux (middleware như redux devtool, redux thunk…)

# Actions/Dispatch

* Download project thực hành Redux:
* [https://drive.google.com/file/d/17u\_Q40NVP95YIPIhtW6JqnXjkHQZ18Bz/view?usp=sh aring](https://drive.google.com/file/d/17u_Q40NVP95YIPIhtW6JqnXjkHQZ18Bz/view?usp=sharing)

## 1.Actions

* Tài liệu: [https://redux.js.org/tutorials/fundamentals/part-3-state-actionsreducers#designing-actions](https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers#designing-actions)
* Actions (tên hành động) là cách ứng dụng React và Redux nói chuyện với nhau. Ứng dụng React sẽ yêu cầu Redux làm việc cho mình, thông qua việc sử dụng actions.
  + Cú pháp: Actions là 1 javascript object, giống với state/props của React.
* Để 1 object javascript là 1 actions, thì phải khai báo theo cấu trúc:

const your\_action = { type: ‘action\_name’, payload }

* + Type: thuộc tính này bắt buộc, dùng để miêu tả hành động (action), nó giống như id để phân biệt các action với nhau.
  + Payload: data muốn truyền theo hành động, thuộc tính này KHÔNG bắt buộc phải có với action

Ví dụ về actions:

* Actions có payload (truyền data cùng actions)
  + { type: 'todos/todoAdded', payload: todoText }
  + { type: 'todos/colorSelected, payload: { todoId, color } }
* Action không truyền payload
  + { type: 'todos/allCompleted' }
  + { type: 'todos/completedCleared' }

## Dispatch

* Dùng keyword ‘dispatch’ để fire (thực hiện) 1 action.
* React => dispatch(action) => Redux
* Dispatch là 1 keyword đặc biệt, chỉ có thể hoạt động trong môi trường react-redux, cũng như nhờ có dispatch, chúng ta từ React, bắn actions vào Redux.
* Redux sau khi nhận được actions, sẽ cần dùng Reducer để xử lý tiếp.

# Reducer

* **Reducer** (công nhân của Redux): sau khi nhận được actions gửi từ React, Redux sẽ gọi tới Reducer để xử lý.
* Reducer sẽ dựa vào type (tên của actions) để biết nó cần làm gì.
* Đầu vào của Reducer là actions (gồm tên (type) hoặc có thêm data (payload))

Đầu ra của Reducer là cập nhật ‘state của Redux - thứ lưu trong Store Redux’

* Reducer sẽ được cấu hình (theo template của Redux) gồm 2 tham số: initState (state mà Reducer quản lý) và actions đầu vào.

* Reducer giống hệt như vòng switch - case (if/else), nó sẽ quét theo ‘tên action’ để biết cần xử lý như thế nào.

* Tại sao khi khai báo action trong Reducer code lại chạy được ?
* Vì tất cả reducer đã được khai báo ở file store.js, khi ứng dụng React chạy lên, thì nó đã biết có bao nhiêu actions cần xử lý rồi :)

## useSelector, useDispatch Hook

* Đối với Redux/toolkit, thay vì dùng Reducer, thì sử dụng Slide. Tuy nhiên, tác dụng của 2 cái này là giống nhau (khác nhau về cú pháp viết)
* 1.useSelector Hook useSelector cho phép chúng ta truy cập ‘state của Redux’, bằng cách cung cấp ‘state’ như là tham số đầu vào, cụ thể:
* const count = useSelector( state => state.counter.count) state: chính là state của Redux, được thư viện cho như là tham số đầu vào.
* couter: tên gọi vắn tắt của Reducer (khai báo trong file store.js) count: tên của state-redux được lấy ra từ counter Reducer
* Cách viết trên là cách viết tắt của arrow function, cách viết đầy đủ là: const count = useSelector( (state) => { return state.counter.count } )
* Như vậy, khối code này sẽ trả ra giá trị, (state) => { return state.counter.count }, sau đấy gán ngược vào biến const count
* Mỗi một khi giá trị lấy từ useSelector() thay đổi, thì ngay lập tức giao diện React bị rerender (vẽ lại). Amazing :)

## useDispatch Hook

Công cụ để lấy ra keyword ‘dispatch’, dùng để fire actions từ React tới Redux

# Sử Dụng Redux Lưu Thông Tin Người Dùng 3 bước thực hiện:

* Step 1: Khai báo dispatch + actions (tại component của React)
* Step 2: Khai báo reducer + logic xử lý của nó (tại thư mục reducer)
* Step 3: Sử dụng state của Redux (tại component React)

## #79 Loading Bar - Hiển Thị Thanh Loading Khi Gọi APIs

* Tài liệu:
  + React Icons: <https://react-icons.github.io/react-icons/search?q=spinner>
  + CSS Make an icon spins: [https://stackoverflow.com/questions/65298589/howto-make-an-icon-spin-in-react](https://stackoverflow.com/questions/65298589/how-to-make-an-icon-spin-in-react)
  + CSS Disabled button:
  + <https://www.w3schools.com/cssref/sel_disabled.asp>
  + Cài đặt thư viện: **npm install --save-exact nprogress@0.2.0**
  + Link github: <https://github.com/rstacruz/nprogress>
  + Customize thư viện: <https://learnjsx.com/category/4/posts/nextjs-nprogress>

## Redux persist - Xử lý Data Khi F5 Refresh

* Thư viện redux-persist giúp ghi data Redux vào local Storage, đồng thời, khi F5 lại trang, nó sẽ tự động nạp data từ local Storage vào ứng dụng Redux.
* Tài liệu:
  + Cài đặt thư viện: **npm install --save-exact redux-persist@6.0.0**
  + Github: <https://github.com/rt2zz/redux-persist>