**ReactJS**

1. **Cấu trúc thư mục ReactJs**

File src: Thư mục quan trong nhất. Làm việc trên thư mục này.

* App.js : component mặc định hiển thị đầu tiên khi khởi tạo.
* Index.js : kết nối component App.js đổ dữ liệu vào file pulic 🡪 index.html.

Package.json : nơi khai báo tên project, version của project, dependencies chứa các package ( module ).

+ scripts: trong package.json để khai báo tên và cú pháp theo yêu cầu của người dùng.

Node\_modules : nơi chứ thư viện.

Pulic : index.html là file chạy đầu tiên.

* %PUBLIC\_URL% : đi đến thư mục public.

Yarn: quản lý module giống npm.

1. **Ôn lại một số kiến thức:**

**Lưu ý :**

* Kết thúc một lệnh phải có dấu chấm phẩy ;
* Giữa những toán tử , biến phải có một khoảng cách.
* Độ dài tối đa trên một dòng 80 kí tự.
* Code block { } một khối

**ES6 , so sánh var, let và const:**

* **Var :** Có thể khải báo lại cùng tên biến nhiều lần. Có thể khai báo không cần gán giá. Có thể thay đổi biến gốc.
* **Let :** Không thể khai báo tiên biến trùng nhau. Có thể khai báo không cần gán giá trị . Có thể thay đổi biến gốc.
* **Const :** Là hằng số chỉ khai báo 1 lần không thay đổi giá trị biến gốc. Bắt buộc khai báo phải gán trị cho biến.

+ Const trong obj, arr không thay đổi địa chỉ trong ô nhớ, gán bằng array, obj khác.

+ Có thể thay đổi giá trị trọng object và array.

**Scope :** Tầm vực

* **Var :** Tầm vực của var ở trong function.
* **Let :** Tầm vực của let là ở trong block 🡪 { }
* **Const :** Tầm vực của const là ở trong block 🡪 { }

**Hoisting :** var có thể hoisting . let và const thì không hoisting.

**Arrow function :**

* **Argument object:** là những tham số khi truyền vào hàm.

**+** Không bị ràng buộc trong arrow function.

**Con trỏ this trong object :**

* Trong arrow function con trỏ this không được ràng buộc trong object.

**Ý tưởng làm:**

* Giả sử chúng ta có một mảng có nhiều đối tương và có nhu cầu sử dụng lại nhiều lần thì giải pháp là 🡪 chuyển đổi thành một object mới . Có những giá trị của mảng cũ chuyển đổi thành key : value.

1. **Component**

Component : là một thành phần thành phần được tách nhỏ từ giao diện html. Xây dựng lên một trang web cần nhiều component hợp lại. Phân chia ứng dụng thành nhiều component để dễ quản lý.

Hiển thị: <tencompont> hoặc thẻ đóng mở như thẻ html.

* Render() được gọi khi component được gọi khi props hoặc state thay đổi.
* Trong return không được để hai phân tử tồn tại song song. Phải có một div bao ngoài và ( ).

**Component funtion :** không có state, ngắn gọn hơn.

1. **JSX**

JSX : cú pháp viết gần giống html.

Hiển thị biến :để trong cặp { }.

JSX mở rộng: html sử dụng hàm map để lặp mỗi đối tượng phải có key khác nhau.

1. **Props**

Props : là một cái thuộc tính của component.

Truyền tham số props cha sang con thông qua { }

Tạo component cách có 2 cặp thẻ đóng mở nội dung trong cặp thẻ this.props.children.

Sử dụng hàm map để lặp component bắt buốn phải có key khác nhau.

* Các props truyền component con không thể thay đổi được.

1. **Conditional redering**

Falsy: null, undefined, 0, ‘ ’, NaN, fasle.

* True && vephai : nếu vế trái true thì bên kết quả là vế trái.
* False && vephai : nếu trái false thì trả về kết quả là vế trái fase.

1. **Handling Events**

**Handling Events :** Chúng ta sử dụng arrow funtion để thao tác với element.

* Hàm luôn chưa đối số là đối tượng event . event này có tham số thông tin liên quan đến sự kiện đấy.
* Hàm khi truyền vào this.props(name): hàm này được gọi tại thời điểm render 🡪 thay vào đó khi khai báo hàm ở trên chúng ta trả về một hàm return ( ) => { } 🡪 hàm này mới là hàm được gọi khi click.

1. **Basic of Form Handling**

Code: this.state = { email: ‘ ’ }

This.changeEmailHandler = ( event ) => {

This.setState({ email: even.target.value })

}

<input type =‘text’ value={this.state.email} onChange={ this.changeEmailHandler } />

* Event.target.value : lấy giá trị.
* Value phải đi kèm với onChange( ): value thay đổi thì sẽ gọi đến.
* Forms & input trong JSX 🡪 key là submitForm 🡪 e.target.element.value( ).

1. **Fragments**

Mỗi khi component return thì chúng ta không phải bọc element HTML trong một cặp thẻ div thay vì thế chúng ta sử dụng cặp thẻ React.Fragment : không hiển thị ngoài DOM.

1. **Pure Component and React.memo**

**PureComponent :** Không còn tình trạng re-render không cần thiết nữa.

* Kiểm tra props và state có thay đổi hay không nếu có thì re-render, không thì không render.
* Nhiều trường hợp state , props dạng complex thay đổi nhưng shallow equality không phát hiện thay đổi đó => Không render khi cần thiết.

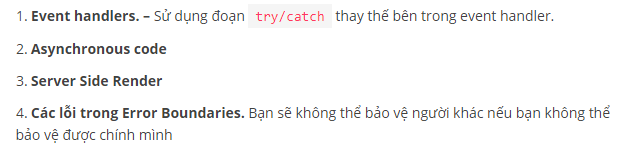
**React.memo:** Nó tương tự như PureComponent nhưng dành cho funtion component thay vì là class.

* Function component render cùng một kết quả được cung cấp cùng một props. Bạn có thể gói nó trong React.memo để tăng hiệu xuất trong một số trường hợp gi nhớ kết quả 🡪 bỏ qua render va sử dụng kết quả được render cuối cùng.
* Bạn không thể dùng nó để ngăn chặn việc render component vì điều này có thể dẫn đến lỗi bất cứ lúc nào.

1. **Error Boundary ( ranh giới )**

**Error Boundary:** là một component con trong React giúp bắt lỗi trong component của nó để hiển thị lên service ghi lại lỗi hoặc hiển thị một fallback UI để xác đinh component con nào bị lỗi trong khi vẫn duy trì ứng dụng hoạt động bình thường.

* **Static getDerivedStateFromError( error ) :** Hiển thị một fallback UI sau khi sảy ra lỗi.
* **componentDidCatch ( error, info )** : Log thông tin lỗi ra.



1. **Immutability : ( bất động )**

Mutate: Bị thay đổi trạng thái bên trong.

* Làm cho chương trình khó đoán được vì nó có thể bị thay đổi trực tiếp, cùng chung vùng nhớ trong bộ nhớ máy tính.

Mutable:Có thể thay đổi trang thái bên trong.

Mutability:Khả năng có thể thay đổi trạng thái bên trong.

Immutability:Không có khả năng thay đổi trạng thái bên trong, không cùng chung vùng nhớ.

* Array: concat(), slice(), toán tử 3 chấm.
* Object:Object.assign({ }, name object copy ), toán tử 3 chấm.

1. **propTypes**

**PropTypes**: npm tải về là thuộc tính của các component. Mô tả chức năng, thuộc tính của component.

1. **Refs**

Refs dùng để truy cập trực tiếp đến DOM thực trong React. Nó rất hữu dụng

* ví dụ trong trường hợp chúng ta muốn thay đổi input mà không muốn sử dụng props hoặc re-render lại toàn bộ component.
* Khai báo thuộc tính refs trong ô mình muốn lấy giá trị 🡪 gọi ra sử dụng.

 this.inputElement = React.createRef();// ref object

**Refs with class component :**

* B1: class con tạo biến createRef 🡪 tạo phương thức focus.
* B2: truyền class input con vào nhận vào ref mới tạo createRef 🡪 button truyền hàm, khởi tạo hàm, bên trong hàm truyền hàm focus của con vào.

**Forwarding ( chuyển ) Refs :**

Dùng forwardRef( (props, ref) =>{ code here } ) khi component cha cần truy cập đến component con

1. **Portals**

Yêu cầu : ta có component B trong component A nhưng lại hiển thị trong component C.

**Sử dụng : ReactDom.createPotal( child, container )**

1. **State**

**State**: Trạng thái của một component khai báo trong constructor của class.

1. **Component Lifecycles**

**Lifecycles** : vòng đời của một component.

* Những method sẽ được gọi trong quá trình component khởi tạo tới khi kết thúc.
* React có 2 DOM chạy song song với nhau 1 DOM thực và 1 DOM ảo

+ virtual DOM sẽ tạo ra trước 🡪 dùng các DOM API thông qua thư viện rout 🡪 tính toán ra các DOM element cần thiết hiển thị lên trên trình duyệt.

+ mout : Gọi các DOM API để thêm component vào trong trang.

+ unmout : component không dùng nữa. Ví dụ như chuyển trang thì component ở trang cũ sẽ mất đi.

Ví dụ : component cha sẽ gọi component con thì :

Gọi lần đầu tiên:

* **B1** : Hàm render( ) của cha được gọi.
* **B2** : Gọi đến con thì hàm contructor sẽ chạy đầu tiền vì ở đâu đó trong code react của chúng ta sẽ có new name con.
* **B3** : Hàm render( ) của con sẽ được gọi.
* **B4** : Hàm componentDidMout( ) sẽ được gọi

\* componentDidMout( ) : Được gọi sau render( ) lần đầu tiên trong một component . Nếu lấy dữ liệu từ trên sever về thì nên gọi trong này hoặc componentDidUpdate( ) tùy hoàn cảnh.

\* **contructor** : khơi tạo state, có thể gội context method sẽ được dùng ở đây.

Gọi lần thứ hai trở đi thay đổi props hoặc state:

* **B5** : Hàm render( ) của con sẽ được gọi lại.
* **B6** : Hàm componentDidUpdate( ) sẽ được gọi.

**\*** componentDidUpdate( ) : Được gọi khi props hoặc state thay đổi.

* **B7** : Trước khi xóa component con thì componentWillUnmout( ) sẽ được gọi.

Gọi thay đổi state của cha :

* **B1**: Hàm contructor của cha sẽ gọi đầu tiên.
* **B2** : Hàm render( ) của cha được gọi 🡪 đến con được gọi.

\* shouldComponentUpdate( nextProps, nextStates ) : mặc đinh là true để cho việc update được thực hiện. Tối ưu react app.

+ React app như một cái cây có 2 nhánh: một nhánh update và một nhánh không update

Giữ nhánh không update lại để tái sử dụng 🡪 khỏi phải gọi render nhiều lần giúp app chay nhanh hơn.

1. **React router**

Dùng thư viện react router : dùng routing phía bên client.

Sử dụng axios để ajax get dữ liệu về.

1. **Higher-Order Component**

**Higher-Order Component:** truyền vào 1 wrapped function hàm được bao bởi hàm khác.

* Một hàm không tên bao component ( có thể thêm tham số nữa )🡪 trả về một component không tên khác.
* Tái sử dụng code.
* Code một App lớn tốt hơn tạo ra nhiều component nhỏ lắp ráp lại với nhau.

**Kết luận:** component giống nhau thì nên viết theo kiểu này HOC truyền prop về cho componet con in ra.

* Sử dụng toán tử { ...this.props } HOC : nếu muốn sử dụng props của riêng con.

**Higher-Order Function** : nhận vào function và trả về function.

1. **Render props : Đừng tự lập lại chính mình.**

Component : cũng chỉ là một function có đầu vào, đầu ra.

* Truyền data và function làm tham số của component 🡪data và function trong có thể làm gì tùy yêu cầu.
* Cung cấp dữ liệu 🡪 ở file hiển thị sẽ sử dụng tùy vào mục đích.

**Kết luận :**

* Cho chúng ta sử dụng lại logic của một component khác.
* Render props được sử dụng như một props với giá trị là một function.

**Ví dụ:** Tạo một component cha chung chứa tất cả những gì component khác đều muốn có 🡪 truyền thông qua prop render( biến hoặc hàm ) của cha sang con. 🡪 Hiển thị con được bọc trong compponent cha với props là hàm render.

* Có thể thay this.props.render() === this.props.children().

1. **Context API : truyền state qua nhiều cấp thích hợp sử dụng Context API**

**Provider :** cung cấp dữ liệu **và consumer :** cần truy xuất vào dữ liệu vào.

Khởi tạo biến nameContext = React.creactContext( ).

* Tạo thẻ nameContext.Provider tham số là value.
* Tiếp theo tạo thẻ nameContext.Consumer.
* funtion gọi đến component khác 🡪 Bên trong cùng này chứa giá trị của tham số value Provider.
* **Static contextType = nameContext :** Truy xuất value Provider trong componet con **this.context.**

1. **Function state**

Khi thực hiện setState thì chưa chắc state của chúng ta đã được update. Nó không phải đồng bộ mà là bất đồng bộ.

Khi setState của component thì component đấy phải render lại các bước:

* Tính toán lại virtual DOM 🡪 update DOM thực tế.

Nếu app của chúng ta làm việc setState nhiều lần cùng một lúc bằng đồng bộ như vậy 🡪 nó sẽ update DOM nhiều lần và không cần thiết.

Mục đích của React là tạo ra virtual DOM nó sẽ tính toán các thay đổi, các phần khác biệt. Vì vậy mỗi lần gọi setState cho giá trị vào một hàng đợi nào đấy.

Trong trường hợp nhiều State được gọi liên tiếp nhau, thì nó sẽ được gom lại rồi nó mới thực hiện cùng một lúc, tính toán ra virtual DOM 🡪 chỉ update một DOM thật là xong.

1. **ReactJS – HTTP and React**

**HTTP trong React :** client thao tác với sever thông API.

* Cài đặt Axios để call API từ phía sever.
* Cài đặt JSONPlaceholder : Tạo API online giả để test và tạo mẫu.

**HTTP GET Request :** gọi trong componentDidMout hoặc componentDidUpdate

**HTTP POST Request :** <form onSubmit={this.submitHandler}> : phương thức gửi liệu đi.

1. **React hooks useState, hooks useContext**

**Hooks**: là tính năng cho phép chúng ta sử dụng state mà không cần tạo class.

* **Hooks** : không được sử dụng trong class.
* **Hooks** : liên quan đến Javascript nhiều hơn là viết bằng class
* **Hooks** : chỉ gọi hook trên cùng. Không gọi hook bên trong vòng lặp, điều kiện hoặc các hàm lồng nhau.
* **Hooks** : cho phép tái sử dụng lại logic component. Chia một component thành các hàm nhỏ hơn.

**Hook useState** : khởi tạo và thay đổi state trong function. Biến giá trị đầu tiên là getter, biến thứ hai là setter.

 với object

**Hook useContex :** Code không bị lồng nhau.

**Hook useEffect :( side effect )** Như sự kết hợp của componentDidMount, componentDidUpdate và componentWillUnmount.

* Effect không cần CleanUp : chạy một vài đoạn code sau khi React đã cập nhật DOM. Có thể chạy chúng và quên ngay lập tức. Ví dụ : Network request, tự ý thay đổi DOM và logging.
* Effect cần CleanUp : componentWillUnmount được sử dụng trong effect 🡪 trong return của hàm useEffect.

Run effect only one : Hook useEffect : sẽ có tham số thứ hai là một mảng [ ], để so sánh nếu thông số thay đổi thì được bỏ qua effect.

useEffect with incorrect dependency : Khai báo và sử dụng hàm bên trong effect.

**Router**

1. **React-router: sử dụng**

**Link:** đường dẫn web app.

**NavLink:** đường dẫn có thêm class active.

**Custom Link:** dựa vào đối tương children.

**Switch:** như câu điều kiện rẽ nhánh switch key.

**Tách file cấu hình Router:** Tách menu để trong component, cấu hình trong file routes.

**Đối tượng Match:**

* B1: trong file routes thêm tham số đầu vào là một { match };

**Đối tượng Prompt:** thông báo khi người dùng đang nhập liệu mà thoát sang trang khác.



**Đối tượng Redireact:** 

Điều chinh đường dẫn cho phép đang nhập mới xem.

Truy xuất thông tin trước đó thông qua đối tượng localtion

* location khai báo thêm vào file routes để lấy thông tin componet trước đó.

**Reder:** custom từng cái route.

import { withRouter } from 'react-router'

truyền các **props history, location, match.**

**Router đối tượng history:**

* History.goBack( ): quay về trang trước đó. Hoặc History.push(‘/’).

**Json Web Token**

**1 Bổ trợ: Authentication.**

Là xác thực: quá trình định danh, xác định tài khoản, ai đang vào hệ thống website của chúng ta.

Có rất nhiều phương thức: tài khoản , mật khẩu, vân tay, khuôn mặt, móng mặt

Ta dùng mật khẩu: cơ chế **TOKEN BASED AUTHENTIATION**.

**2 Bổ trợ: Token.**

Là chuỗi đã được mã hóa ( như một chìa khóa ). Lưu thông tin người dùng.

Được dùng hầu hết các **restful webservice**. Được sử dụng hầu hết ở các web backend

Khi **webservice** cung cấp **API** để truy xuất dữ liệu. **Webservice** đã dùng **token** để xác thực, phân biệt, kiểm tra quyền hạn.

**3 Bổ trợ: Token Authentication.**

Khi đăng nhập thành công: hệ thống trả về **token, token** lưu ở **client** ( **localStorage, session, storage, cookie ). Mỗi request sau đó** ( call api sau đó ) ta sẽ gửi kèm **token** vào phần **header của HTTP**

**Server** kiểm tra **token** hợp lệ hay không.

**Điểm mạnh :**

* Dễ maintain, mở rộng ( vì server không lưu trữ ).
* Bảo mật.
* Uỷ quyền: web game, đăng nhập bằng facebook, google.
* Đa nền tảng: web, mobile,...

**JWT** Là chuỗi gôm ba thành phần : **header, payload, signature**.

Trong **payload**: có thể lấy được thông tin cần thiết do server được đính kèm trong token. VD: fullname, email,...



**Token :** chuỗi ngắn thay đổi liên tục.

**Connect API**

1. **Cấu trúc thư mục trong một dự án Reactjs.**

* **Action:** chứa những hành động của redux 🡪 khi chúng gọi dispath thì tham số là hàm hành động đó.
* **Component:** chứa những component.
* **Constanst:** chứa những biến const.
* **Container:** chứa những component chung gian giao tiếp giữa component con và redux.
* **Reducer:** chứa những file redux.
* **Page:** chứa những component hiển thị trong app.
* **Utils:** chứa các file kết nối với server.

1. **Giới thiệu sơ lược**

API: hiểu đơn giản nó là các url.

**Một số API:**

* mockapi : giả lập API data online.
* Json server : sử dụng off.

**Kết nối với API:**

* Fetch API: trong javascript.
* Axios: Thư viện HTTP Client dựa trên Promise.

**Muốn sử dụng confirm hiển thị thông báo :**



1. **Redux & callAPI**

Lưu giữ liệu kết nối với server trên store 🡪 gọi store sử dụng dữ liệu.

* Viết một action lưu giữ liệu vào Store.
* Viết một action connect với server.

**Middleware & async actions:**

Async actions: là những action chưa sẵn sàng ngay khi gọi.

Sử dung redux-thunk để trì hoãn việc dispatch 🡪 Fetch dữ liệu xong mới dispatch.( phải có dữ liệu thì mới dispatch ).

**Middleware:**

Là một cấu trúc ứng dụng. Lớp nằm ở giữa **reducer** và **dispatch actions**.

Vị trí hoạt động:

* Trước khi reducer nhận được action.
* Sau khi action đã được dispatch.

Ứng dụng: phân chia ứng dụng thành các lớp khác nhau.

Một số middleware: logging, Crash Report.

Một số middleware giải quyết:

* Redux-thunk.
* Redux-saga.
* Redux-obserable.
* Redux-promise.

**Redux-saga**

1. **Tích hợp Material-UI.**

**Xây dựng bộ theme trong Material-UI:** - CSS in JS

* Tạo một đối tượng css bằng js 🡪 gọi sử dụng tự đặt tên class trong DOM. Giải quyết vấn đề trùng tên class khi chúng ta code.
* Theme : Sài chung cho toàn bộ project khi nào thay đổi thì chỉ cần thay đổi ở một chỗ.

**Webpack**

1. **Giới thiệu: là module loader.**

* Giúp cho website tải nhanh hơn.
* Sẽ biết khi nào cần thì mới gọi để sử dụng. Kiểm soát các module.
* Dùng cho dự án lớn.
* Phân chia module và chỉ load khi thật sự cần.
* Đóng gói tất cả file nguồn thành 1 file duy nhất. Nhờ vào Loader mà nó có thể biên dịch các file lại với nhau.
* Biến tài nguyên tĩnh ( hình ảnh , css ) thành 1 module.

1. **Webpack : ReactJS**

* Đóng gói thành các module => giải quyết vấn đề đường dẫn.

1. **Phân tách module**

* Chạy webpack : **npx webpack.** : Đóng gói thành 1 file.

1. **Apply Babel cho project**

* Cho phép xử lý 1 file trước khi require ( import ) hoặc loader vào.
* Giúp chuyển đổi thành javascript thuần.
* Sử dụng các loader đã được xây dựng sẵn.

1. **Babel**

* Biến đổi ES6 thành ES5.
* 1 số module:

**Babel-loader** : Hướng dẫn babel làm việc với webpack.

**Babel-core :** Lấy và phân tính mã, output ra 1 số file.

**Babel-preset-env:** Bộ quy tắc để nói, giải thích với babel về cú pháp ES6, ES7 cần tìm và làm thế nào chuyển đổi thành ES5 code.

**Babel-preset-reactjs :** chuyển đổi code JSX ( ReactJS ) thành JS.

* Cài đặt : npm install –save-dev ...
* **Babel-preset-stage-2 –save:** chuyển đổi code js cho trình duyệt hiểu.

1. **CSS loader**

* 2 module: style-loader và css-loader.

**Css-loader:** Cho biết làm thến nào để hiệu được cú pháp CSS imports.

**Style-loader:** Nhận các css import và thêm vào html.

* Mở rộng:

Plugin: **extract-text-webpack-plugin**.

Mục đích : tách css ra thành file riêng sau khi build, thay vì nằm trong bundle.

**Project webpack :**