1. **Các extensions + react dev chrome**

**react dev chrome:** cho chrome

Vs code extensions: react code snippet, react redux ES6 snippet, boostrap 4 and fontawesome, prettier

Khởi tạo project:

npm install -g create-react-app(cài 1 lần dùng lun ko cần code lại cái này),

create-react-app [*tênproject*]

npm install react-bs-notifier

1. **Định nghĩa Component**

Cách 1: định nghĩa 1 function bình thường có tên

Function Champ() {

Return ( .....)

}

Note: ko thể return 2 div cùng 1 cấp >>> chỉ 1 div chứa các tag con – only 1 wrapper.

cách 2: anonymous function

var Champ = function(){

return ( ............ )

}

Cách 3: arrow function(ko cần keyword: return)

() => (.............);

Cách 4: Class

Class Champ extends Component {

Render(){

Return ( ......... );

}

}

1. **Props(Properties)**

**Props:** thuộc tính của Component. Là cách truyền input vào Component.

1. Function Champ(props){

Return (

<div className=”test”>{props.*ten*}</div>

);

}

2. Class

class Champ extends Component {

render() {

return ({**this.props.*ten***});

};

}

Note: cách tạo class này thì ko cần truyền ghi input props. Mà mặc định đã có sẵn. Nhớ có keyword: this

1. **JSX**

Cho phép viết biến với {} (code, props của js)

Dùng công cụ: html to JSX

1. **COMPONENT HÓA**

Cắt từng phần của file html sang các component.

1. **SỬ DỤNG PUBLIC\_URL**

SET-UP thư viện, file css, bootstrap, js bằng cách thêm trước các link tiền tố **%PUBLIC\_URL%.** Đây là 1 biến dc định nghĩa sẵn trong ReactJs.

**Href=”%PUBLIC\_URL%/**index.css”

1. **CÁCH PHÍM TẮT TẠO NHANH**

Install React code snippet extension

Rcc: react component

Sau khi set-up component: **export component** trong file component lun

**Import** *tênClass đó* vào file App.js (import *tênClass* from ‘./components/tenfile’)

1. **ĐƯA CÁC COMPONENTS VÀO FILE APP.JS**

**Copy** code html và chuyển sang JSX bằng ‘convert html to JSX’.

Then, đưa vào file component.

Để có thể truy cập các props: khởi tạo constructor(props){super(props)};

Shortkey: ‘con’ > tạo constructor(props){super(props)};

1. **VIẾT ,GỌI FUNCTION CÓ/KHÔNG THAM SỐ TRONG REACTJS**

Ko có tham số: onClick={this.*tênfunc*}; Note: ko có ()

Có thêm tham số:

1. Dùng arrow function trong onCLick luôn. onClick={() => this.*tênfunc*(param)}; (có thể dùng cho hàm ko tham số luôn nhớ có (), phải có this )
2. Dùng bind(): onClick={this.*func.*bind(this, ‘string’);}
3. **STATE TRONG REACTJS**

Là nơi lưu các biến, tham số trong react.

Là từ khóa > có sẵn trong react và ko cần khai báo lại.

Là 1 Obj > lưu ko giới hạn

Cú pháp ghi trong constructor(props){

Super(props);

This.state = { //shortkey: stt

Key: value

}

}

* Để thay đổi state: this.setState({key: value}); // shortkey: sst.

Trong reactJS: khi thay đổi các state bằng setState thì nếu thỏa điều kiện, các hàm có liên quan đến state sẽ tự động gọi. 🡺 khi đó ta ko cần gọi lại hàm nữa. (đây gọi là biến trung gian).

State là biến privacy > chỉ chạy trong 1 component

Khi 1 hàm lấy data thì data chỉ nằm trong hàm đó. Muốn những cái khác lấy dc data thì phải đưa lên State

1. **DEFAULTVALUE**

Có thể dùng trong thẻ component: <Header defaultValue={this.props.title} ........ />

1. **Ref**

Đây là biến trung gian 🡺 để thay đổi thông tin mà ko cần load lại trang.

Ko cần render lại thông số khi **sửa sẽ lưu vào biến ref**. Dùng để lấy value trong input

Khác với State: state là biến luôn có. Dùng để phân bổ thông tin cho các hàm.

1. **LifeCycle**

Vòng đời của component.

lifeCycle là tập hợp các hàm dựng sẵn(build-in) trong reactJS.

Có thể can thiệp vào trước, sau khi render()

Gồm 4 quá trình:

1. init – khởi tạo
2. mounting – gắn những hàm kết nối component.
3. updation – props, states.
4. Dismount – hủy gắn kết

* Note: đây là những hàm build-in 🡺 keyword phải viết đúng và có viết hoa.

Shortkey: cwm > **componentWillMount()**

**Cdm > componentDidMount()**

**Cwr >componentWillReceiveProps()**

**Scup > shouldComponentUpdate()**

**Cwup >** componentWillUpdate()

**Cdup >** componentDidUpdate()

**Cwun > componentWillUnmount()**

1. **Các hàm build-in của lifecycle**

**componentWillMount**(): hàm này sẽ chạy **trước** hàm render();

**componentDidMount():** hàm này sẽ chạy **sau** hàm render();

* **Updatation**:

State: **shouldComponentUpdate**(nextProps, nextState){} \\ hàm này gọi khi ta gọi đến 1 method làm thay đổi state, hay props.

Khi hàm **shouldComponentUpdate** return **true** thì sẽ chạy tiếp 3 hàm : componentWillUpdate(), render(), componentDidUpdate(). **False** thì sẽ ko chạy 3 hàm này.

Props: **componentWillReceiveProps():** hàm này gọi khi prop của component bị thay đổi.(thường là 1 event trong parent Component – App.js)

Còn lại 3 hàm kia giông như State.

1. **REACT ROUTER**

Khi click vào menu thì ko đổi trang, chuyển hướng web mà ko load lại.

1. Tạo giao diện
2. Điều hướng SPA
3. Truyền tham số
4. Dựng Slug cho url = router
5. redirect
6. **CÁCH BƯỚC CHUYỂN HTML SANG COMPONENTS**
7. Copy mã HTML vào các components. (note: dùng công cụ Html to JSX)
8. Import vào file App.js(đây coi như file root của Reactjs)
9. Đặt tenclass vào thẻ div của App.js
10. **REACT ROUTER**

**Keyword: react-router-dom**

**Property:** activeClassName

1. **Dùng Array map để đưa dữ liệu ra từ .JSON**

**Tạo** 1 File .JSON gồm 1 mảng chứa các Objects(id,name,img,excerpt,...)

**Dùng Array map** để render ra giao diện

Render(){

Data.**map((value, index)** =>

Return(

<Com

key={index}

name={value.name}

img={value.img}

excerpt={value.excerpt}

></Com>

)

)

}

1. **Đường dẫn thân thiện với SEO**

**Keyword:**  slug javascript.

1. **Làm giao diện cho backend với reactJS**

Dùng để viết các chức năng tương tác với back-end: thêm, sửa, xóa, tìm kiếm dữ liệu dạng Single Page Application(SPA).

1. tạo user – sdt – quyền quản trị
2. chuyển giao diện sang ReactJS Components
3. tạo dự liệu .json
4. đẩy data vào Components
5. viết chức năng thêm sửa xóa.

Chuyển các html sang Component(1st: link các đường dẫn bootstrap, jquery cho đúng)

Note: onClick={this.funcName()} : đây là hàm chạy lun. Ko cần click cũng chạy. (ko tốt)

onClick={() => this.funcName()}: click vào mới chạy.

Define 1 func

Dùng if để check điều kiện để chạy/gọi hàm đó.(thông qua State. True or False)

{this.funcName()}: lúc này thì sẽ ko call exceeded nữa. Mà đã có dk để call.

Dùng khi 2 nút nằm cùng chung 1 Component thì thay đổi State thôi.

1. **Cách kết nối component con sang component cha**

Dùng **props** : define 1 func trong App.js(component Cha). Then, dùng props này để kiểm tra **State** của component Cha.( chứ ko phải là **State** của Con) hoặc dùng Props này để connect 1 func của Cha cho Con.

<[ComName] connect={() => this.*funcName*()} />

Trong *ComName:(*component con*)*

<div onClick={() => this.props.connect()}>

Sơ đồ: comA changes state comCha > comB changes

Muốn thay đổi thuộc tính của State Cha thì dùng 1 hàm của ComCha. Và ComCha truyền cho conCon hàm đó

Các hàm xử lý liên quan đến các ComCon thì đặt ở trong App.js-comCha

Truyền Con > Cha: phải truyền vào **cái tham số** trong hàm của Cha.

Cách truyền từ Cha > Con/Chau:

1. comCha: truyền 1 prop cho comCon
2. comCon: truyền 1 prop cho conChau với prop từ conCha ( propCon = {this.props.propCha})
3. comChau: tạo 1 func để hứng propCon. Rồi gọi hàm đó trong comChau

Cách truyền từ Con/Chau> Cha:

1. comCha:phải tạo 1 func có input ( con/chau-cha)
2. comCon/Chau: hàm gọi đến func đó thì ko cần tham số - input
3. **Kết nối các component con với nhau**

**Com**Cha truyền comCon 1 prop.

comCon check prop đó true or false > hiện thị nội dung.

1. **Lấy dữ liệu từ .json**

**Import** data from [datafile.json]; 1 mảng chứa các Obj{“name”:..., “phone”:..., “address”:...}

Lưu vào **State** của comCha.

comCha truyền props cho comCon

In comCon: this.props.*propName*

Dùng Arr map method để lặp dữ liệu ra.

Tạo 1 comChau

Trong conCon: define 1 func: dùng .map((value,index) => <conChau userName={value.name} **key**={index} id={index}/>) **Note: key ko dc truyền như 1 prop.**

Trong comChau: hứng prop từ comCon

Nếu trong .json có 1, 2, 3 mà tương ứng với các cấp độ: 1 = Admin, 2=Mod, 3=User. Thì dùng câu điều kiện (prefer Switch case) trong ComChau để in ra chữ. Chứ ko in ra số.

* Lấy nội dung từ các field > gửi lên comCha > đống gói thành obj{}

1. **Xóa trắng field sau khi submit**

Cách 1: viết hàm chuyển value của các field về = “ “

Cách 2: dùng nút type=”reset” (Note: các fields và nút reset phải nằm trong Tag <form> thì mới reset được, để hiểu là reset những field trong <Form > đó)

* Cách để show/hide 1 Component:
  + Dùng 1 if với state ở ComCha(cái mà thằng comCon nhận dc từ comCha thông qua props). Nếu true: show return( thằng ComChau), nếu ko thì hide

1. **Sửa nội dung**

**Tìm đ**ến id cần sửa. Rồi cập nhật state của App

1. **Xóa nội dung**

**T**ìm đến id cần xóa trong Component chứa thông tin

Đưa id lên App.js

App.js tìm đến id đó và xóa trong state.

* Dùng filter method để lọc ra các item !== [id cần xóa]

1. **localStorage**

**gồm 3 hàm cơ bản:** localStorage.setItem, localStorage.removeItem, localStorage.getItem.

đưa dữ liệu vào localStorage: lần đầu tiên thì lấy lun cái file Json luôn.

Khi làm việc với database: sau khi xử lý xong thì gọi API ra và set lại cho database

1. **xử lý với JSON**

JSON.stringify(‘mảng đối tượng’): biến file .json **từ array sang 1 String** (làm lúc đưa từ Client lên database/localStorage)

JSON.parse(‘mảng đối tượng’): biến file .json **từ** **String thành array**( làm lúc lấy từ database xuống Client)

1. **Đóng gói và up lên host**

Npm run build: đóng gọi project

Cài đặt Xampp. Thay đổi các file cấu hình, tạo virtual host

**keyword: create virtual host xampp**

* **cách lấy dữ liệu từ input**

**Cách 1: ko xài Redux/ xài React thuần**

1. **lấy dữ liệu trong ComCon**

* tạo 1 hàm

isChange() = (event) =>{

const name = event.target.**name** \\ **name:** là attr của <input name=”ten” >

const value = event.target.**value \\ value: là** cái mình nhập vào trong field

}

* Nhớ có constructor(), state trong comCon này . Để lưu data tạm thời > state tổng.

This.state = {

Title: ‘’, \\\\ đây sẽ hứng **name**

Content: ‘’, \\\\ đây se hứng **value**

}

/\*\*\*\*\* state này ko cần khai báo cũng dc vì khi dùng [name]: value thì nó sẽ tự thêm vào state lun \*\*\*\*\*\*\*\*\*/

* This.setState({

[name]: value \\ đây là cách tạo các properties trong OBJ - react

})

* <input onChange={(event) => this.isChange(event)}> \\gọi hàm trong cái input cần lấy value
* Đã lấy dc data trong các field. Kiểm tra xem state của Component này có data chưa: F12> React > *view state*.
* Khai báo 1 hàm (hàm nhận 2 tham số name, value của state)  
   addData = (name, value) => {

Const item = {}; \\\ áp dụng cách thêm property trong Obj của JS

Item.name = name;

Item.value = value; \\\\\ nên đặt trùng tên với name của input

/\*\* console.log(item); \*\*\*/

}

* Đưa vào nút submit/reset (nhớ nút này phải nằm trong tag Form chung với mấy input lấy data) <button type=”reset” onClick={() => this.addData(this.state.name, this.state.value)}>
* Xong phần lấy data của ComCon

1. **Đưa data lên App.js – ComCha**

* **Flowchart:** app.js > comCon = Props, conCom gửi dữ liệu lên = chạy hàm của app.js
* **<Com**Con getData={(item) => this.addData(item)}/> \\\\ truyền Props cho ComCon từ App.js
* **Constructor** trong app.js.
* **Define func \\\\\\\\\App.js**

addData() = (item) =>{

[arr-can-them].push(item)

}

* Sau khi đã **truyền Props** cho ComCon thì bên ComCOn sẽ có this.**props**.**getData \\getData** là Props mà ComCha truyền xuống ComCon. Quay lại hàm lấy data trong ComCon addmore:
* addData = (name, value) => {

Const item = {}; \\\ áp dụng cách thêm property trong Obj của JS

Item.name = name;

Item.value = value;

/\*\* console.log(item); \*\*\*/

this.props.**getData(item); \\\ up lên App.js**

}

* trong App.js tạo 1 hàm đẩy dữ liệu lên database
* DONE

**Cách 2: Dùng Redux**

* **Flowchart: mọi xử** lý chức năng đều bình thường. Nhưng cách 1 thì gửi từng tầng lên App.js, thì Redux cho phép gửi lên thẳng Store
* **Cài** npm i redux, npm i react-redux --save
* **Tạo** Store.js. Xong **export default** store lun để ko quên
  + **Var/const**  redux = require(‘redux’);
  + **Reducer ở đây. (**đặt sai là error**)**
  + var/const store = redux.createStore(allReducer);
* **Reducer: là mấy cái hàm trong ComCon để xử lý data**
  + Tạo State: **shortkey: reducer**
* Done Store
* **Kết nối Store với các Components (store: lun chứa state, dispatch)**
  1. **mapStateToProps() \\** để lấy State nào đó rồi biến thành Props trong ComCon
  2. **mapDispatchtoProps() \\ lấy** reducer nào đó rồi biến thành hàm trong ComCon
  3. **connect()([**Component**]) nhớ** import connect from ‘react-redux’
* **1 và 2** truyền vào như Props nên: this.props.[name]
* **Connected**

**\*Cách chuyển Obj về 1 mảng chứa nhiều Obj**

Let arr =[];

Arr.push({

Property: value,

Property: value,

Property: value,

Property: value

})

\*Thường thì data sẽ được lấy trong hàm componentWillMount() > đưa lên State > 1 hàm để xuất ra.

\*Tạo 1 hàm với điều kiện để xuất data ra. Rồi trong hàm render {this.*tenfunc*()} (cái component chứa data thì cho nó vào trong hàm lun)

\*Biến trạng thái: isShow: true/false. Define trong state của Store

\*Khi làm với Redux: tạo 1 action chỉ thị để get data> lưu tạm data vào state trong Store> trong ComCon có hàm StatetoProps rồi lấy data ra.

* **Cách sửa nội dung**

1. Đẩy **item** lên store
   1. Trong store:Tạo action get data, tạo 1 state là *emptyObj*. Dùng ...state, *emptyObj.***action**.*tenprop*
2. **Store** update trong **state**
   1. Dùng subscribe(this.state) để the dõi state
3. Đẩy **item** từ **state** vào form
   1. Lun dùng 3 hàm kết nối với store
   2. Kết nối comCOn với store để lấy state.*emptyObj*
   3. Dùng **defaultValue attr** của **input**
4. **Form** nhận dc
   1. **Click** botton lấy data đã sửa > store> update data

* **Cách tạo Notifier**

1. Tạo 1 component và đặt trong App.js
2. Copy trong Docs
3. Để display Notifier thì phải qui định 1 biến/state: true/false
4. Kết nối comNotifie với store = 3 hàm cơ bản.
5. Tạo 2 actions/2 chỉ thị: tắt và bật thông báo
6. Khi bật dc rồi thì setTimeOut thời gian để chạy hàm tắt thông báo

**CÁCH KẾT NỐI REACTJS VỚI NODEJS**

1. **Dùng Axios**
2. **Khai báo 1 const** khai báo từ node js axios.get ...
3. **Tạo 1 state ban đầu: null**
4. **Dùng conponentWillMount(){}:**kt nếu ko có data, thì lấy data từ axios(promise). Rồi setState vào là cái data promise tra về
5. **Define 1 func** để Rồi dùng map() để lặp ra các item trong mảng trong state đó.
6. **Trong render**  gọi hàm ở bước 5 ra thôi {this.*tenham*()}