***React JS***

1. **AXIOS**:

-nhiệm vụ: tương tác với **back-end** (cụ thể là các **API** của **Back-End**).

-cú pháp **install**: “**npm i axios -S**”.

-**Axios** **return** về 1 đối tượng gọi là **promise**.

-**Axios** được xây dưng theo kiểu **bất đồng bộ**.

-**METHOD**:

-**GET:** dùng để **lấy dữ Liệu**.

- **POST:** dùng để **thêm dữ liệu mới**.

-**PUT:** dùng để **cập nhật dữ liệu** **ghi đè** dữ liệu **cũ** thành dữ liệu **mới**.

-**PATCH:** dùng để **cập nhật từng phần** của dữ liệu.

-**URL**:

-Là đường dẫn **API**.

        Axios({

            method: 'GET',

            url: 'http://elearning0706.cybersoft.edu.vn/api/QuanLyKhoaHoc/LayDanhSachKhoaHoc?MaNhom=GP01',

        }).then((res) **=>** {*//res đại diện cho csdl trả về*

            console.log(res);

        }).catch((err) **=>** {

            console.log(err);

        });

console.log(12);

Vì **Bất đồng bộ** nên nó **không** chờ **dữ** **liệu** **trả** **về** mà nó tiếp tục **chạy** **tiếp** **dòng** **lệch** “console.log(12)’’ vì chúng ta ko biết khi nào **server** trả về, lỡ **server** **chết** hay **đứng** bất chợt thì chúng ta sẽ bị **block** ngay Axios!

-Nó ko quan tâm **server** trả vể khi nào việc của **axios** là **sent** **request** lên **server** do đó dẫn tới nó sẽ trả về cho chúng ta một đối tượng **promise**.

-đối tượng **promise** này có 2 phương thức là **then**() và **catch**() , lấy dữ liệu **thành công** thì nhảy vào **then**() nếu **thất bại** thì nhảy vào **catch**() .

2. **REDUX**:

-Cú pháp **install**: “**npm i react-redux redux -S**”

Bước 1 :

-import **createStore** , **Provider** vào .

import { createStore } from 'redux';

import { Provider } from 'react-redux';

-tạo 1 **RootReducer** dữ liệu tổng.

import RootReducer from './Redux/Reducers/root';

-khởi tạo **store**  và truyền theo như dưới. **index.js**

//khởi tạo store

const store = createStore(

  RootReducer,

  window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_()

);

-bọc App bằng **Provider**

ReactDOM.render(

  <Provider store={store}> //đầy store dc tạo trên vào **Provider**

    <App />

  </Provider>,

  document.getElementById('root')

);

- khởi tạo 1 **RootReducer** dữ liệu tổng. **root.js**

import {combineReducers} from 'redux';

const RootReducer = combineReducers({

})

export default RootReducer;

-chúng ra cần 1 thành phần của **Redux** chính là **combineReducers** để có thể tạo ra **RootReducer**  -toàn bộ dữ liệu trên **store** bên trên sẽ được lưu trong **RootReducer.**

-khi **lấy** **dữ liệu** từ trên **API** thì mình sẽ gửi về trên **store** và cất trên đó. Để làm dc điều đó thì đầu tiên trên store mình phải có một biến dùng để **chứa** dữ liệu **get** từ **API.**

Nên trong **root.js**

Ta sẽ tạo 1 dữ liệu tên là **course**  và nó được **quản lý** bởi 1 **reducer** có tên là **CourseReducer.**

import {combineReducers} from 'redux';

import CourseReducer from './course'; //đường dẫn đúng ./Redux/Reducers/course

const RootReducer = combineReducers({

    course:CourseReducer,

})

export default RootReducer;

**//course.js**

**Một Reducer** bao gồm 2 thành phần chính:

\* là giá trị ban đầu của **course** ta lưu trên **store** là gì.

C1/

let initialState = {

    courses: [], //danh sách khóa học

courseDetails: [], //chi tiết khóa học

}

Tại sao ko khởi tạo theo kiểu bên dưới ‘C2’.

C2/ gây khó khăn trong việc quản lý dữ liệu vì tạo rất nhìu producer quản lý

let courses = []

let coursesDetail = []

vì khi khởi tạo theo kiểu này :

< < *-ngoài dữ liệu khóa học ra chúng ta còn có dữ liệu chi tiết khóa học rồi còn nhìu dữ liệu khác.*

*-và nếu làm theo cách ‘C2’ thì t phải tạo rất nhìu* ***reducer*** *để quản lý cho từng dữ liệu đó.*

*Nên vì vậy nhìu kiểu dữ liệu nhưng tất cả chúng để bổ trợ cho khóa học. nên ta sẽ gôm chung lại và ta chỉ cần 1* ***reducer*** *để quản lý đống dữ liệu liên quan đế khóa học đó. Tiện hơn trong việc quản lý. Và tường minh hơn.*  > >

\*Thành phần thứ nhất là dữ liệu ban đầu. Nên thành phần thứ 2 là 1 **reducer** quản lý có tên **courseReducer.**

let initialState = {

    courses: [],

}

const CourseReducer = (state = initialState, action) => {

    switch (action.type) {

        case 'FETCH\_COURSES': {

            state.courses = action.payload;

            return { ...state };

        }

        default: return state;

    }

}

//export ra bên ngoài để sử dụng

export default CourseReducer;

**\*\*CourseReducer**:

-là 1 **reducer** quản lý biến **courses.**

**-**nhận vào giá trị hiện tại của dữ liệu là **courses.**

-ban đầu mặt đinh dữ liệu cho nó bằng **initialState** lúc mới vào.

-nó nhận **action** được gửi lên từ các **component**.

-và nó tiến hành xử lý **action** và trả về dữ liệu mới cho mình.

-ta dùng switch để kiểm tra **loại** của **action**.

-**FETCH\_COURSES** là 1 **action** được gửi từ **component** ờ đây là **Home component** component này lấy dữ liệu từ **API** sau đó dó nó đẩy vào payload của **action** đó.

 Axios ({

            method: 'GET',

            url: 'http://elearning0706.cybersoft.edu.vn/api/QuanLyKhoaHoc/LayDanhSachKhoaHoc?MaNhom=GP01',

        }).then((res) => { //res đại diện cho csdl trả về

            // console.log(res);

            // dispatch lên 1 action

            this.props.dispatch({

                type: 'FETCH\_COURSES', //loai action tạo ra bên CourseReducer

                payload: res.data, //ta đẩy dữ liệu từ API vào store luu tru.

            })

        }).catch((err) => {

            console.log(err);

        });

**\*\* Tại Component** khi ta muốn lấy dữ liệu trả về ta chỉ cần khởi tạo 1 biến để hứng dữ liệu đó.

Để được vậy ta phải sử dụng một hàm của thư viện “**react-redux**” là **connect**()

import { connect } from 'react-redux';

sau khi **import**, việc của ta chỉ cần khởi tạo hàm để Map dữ liệu về **props** của **component**

const mapStateToProps = (state) => ({

    courselist: state.course.courses,

})

export default connect(mapStateToProps)(HomeScreen);

Việc còn lại chỉ cần xử lý dữ liệu nhận được từ **store** thôi là xong.

class HomeScreen extends Component {

    state = {}

    render() {

        return (

            <div>

                <h1 className="display-4 text-center">Danh Sách Khóa Học</h1>

                <div className="container">

                    <div className="row">

                        {this.props.courselist.map((item, index) => (

                            <div className="col-3">

                                <CourseItem item={item} />

                            </div>

                        ))}

                    </div>

                </div>

            </div>

        );

    }

Khi hàm **connect (MapStatetoProps)** chạy nó sẽ lấy dữ liệu và sau đó nó sẽ map dữ liệu vào biến **courseList** ta đã khởi tạo trong hàm **MapStatetoProps () .**

**Render** dữ liệu từ props và truyền nó vào các **component** chúng ta cần show lên.

**Memo ----- ‘react’**

: giúp cho việc khi **component cha** **render** mà chúng ta **ko muốn component con render** lại thì dữ liệu của component con ko thay đổi. thì chúng ta sử dụng **memo**

import React, { memo } from 'react';

*const* demoHookChildren = () *=>* {

    console.log('Demo Children rerender');

    return (

        <div>

            <h5>demo hook children</h5>

        </div>

    );

};

export default memo(demoHookChildren);

component cha :

 return (

        <div>

            <DemoHookChildren />

        </div>

    );

Nếu ta để v khi component cha render thì component con k render lại.

Nếu chúng ta muốn component con render khi 1 dữ liệu bất kì truyền vào nó thay đổi thì ta chỉ cần dùng

return (

        <div>

            <DemoHookChildren count={count} />

        </div>

    );

Nếu biến count thay đổi thì component con sẽ thay đổi.

**useState -hook**

dùng để thây thế cho **state** bên **class** **component**

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

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

*const* [isAgree, setIsAgree] = useState(false);

useState tạo ra 1 mảng với 2 phần tử. phần tử 1 là biến và phần từ 2 là phương thức set của biến đó.

**useEffect -hook**

dùng để thay thế 3 cho **lifeCycle** là**:** **componentDidMount**, **componentDidUpdate**, **componentUnMount** của **class** **component**

useEffect() nhận input là 2 đối số , 1 là **call back function thực thi** , 2 là **mảng đối tượng cần so sánh để thay chạy hàm call back.**

**-**chính nhờ việc truyền mảng đối tương so sánh nên useEffect dc sử dụng cho 3 trường hợp

- 1 đống vai là **componentDidMount** :

useEffect(() *=>* {

        console.log('useEffect DidMount === componentDidMount ')

    }, []);

**Hàm này sẽ chạy 1 lần duy nhất sau đó ko chạy nữa.**

**-2** nếu ta muôn nó chạy khi có thay đổi trong function giống như **lifeCycle componentDidUpdate** thì ta dùng hàm này với đối số 2 là null:

 useEffect(() *=>* {

        console.log('useEffect run');

    });

**Hàm này sẽ chạy liên tục khi có gì thay đổi trong component.**

-**3** nếu ta muốn nó chỉ chạy khi có 1 đối tương 1 biến cụ thể nào đó thay đổi thì ta dùng như sau:

 useEffect(() *=>* {

        console.log('UseEffect just run when the variable isAgree change');

    }, [isAgree]);

**Nếu isAgree** thay đổi thì hàm này sẽ thực thi.

=> trong Hook chúng ta có thể sử dụng nhìu useEffect tùy ý .

**useCallBack** :

- nhận vào 1 hàm và có công dụng là trả ra 1 bản sao của hàm đó.

*const* increaseCountUseCallBack = useCallback(()*=>*increaseCourse,[]);

nếu ta viết như vậy thì biến **increaseCountUseCallBack** chính là bản sao của **increaseCourse** .

và ta chỉ muốn nó tạo bản sau vào 1 lân duy nhất thì ta thêm đối số 2 là 1 mảng rỗng []

-Và hàm **increaseCountUseCallBack** này sẽ không thay đổi trong suốt quả trình component render nhìu lại nhìu lần và dùng hàm **increaseCourse**  có th**ay đổi**  thì nó vẫn giữ nguyên.

-Và nếu ta muốn component con nào dc truyền 1 hàm tương tự **increaseCourse** mà nó ko render lại khi component Cha thay đổi thì ta dùng **useCallBack** để tạo 1 bản sao.

*const* callSumUseCallBack = useCallback(() *=>* calSum(), [num1]);

khi truyền vào 1 giá trị . tức là khi biến **num1** thây đổi thì hàm **useCallBack** này tiến hành sao chép 1ai 1 lần nữa