LỘ TRÌNH REACTJS

3, Làm quen với React element

-> Thêm React vào dự án

-> Document.createElement

-> React.createElement

-> Hiểu về thư viện React-DOM

-> Render UI với React-DOM

4, Tính năng cơ bản và cú pháp

-> Tạo dự án với create-react-app

-> Cấu trúc thư mục của create-react-app

-> JSX là gì?

-> Function component

-> Class component

-> Sử dụng lại Component

-> Khái niệm Props

-> Children Props

-> Thư viện “prop-types”

-> Khái niệm State

-> Phân biệt Props & State

-> Xử lý DOM events

-> Stateless & Stateful Component

-> Viết CSS inline

-> Viết CSS external

-> Sử dụng SCSS

-> Render nội dung với điều kiện

5, Style trong ReactJS

-> Giới thiệu CSS modules

-> Sử dụng đồng thời nhiều classes

-> Sử dụng className động (dynamically)

-> Sử dụng thư viện “classname”

-> Sử dụngg Styled Components

6, Thư viện React Router DOM

-> Điều hướng trang với React Router DOM

7, Debug

-> Sử dụng React Developer Tools

-> Khái niệm Sourcemaps

-> Sử dụng Error Boundaries Component

8, Xây dựng cấu trúc dự án

-> Cấu trúc thư mục dự án

-> Cấu trúc Component

-> Cấu trúc assets

-> Cấu trúc util

-> Thêm file cấu hình

9, Cấu hình theo môi trường

-> Enviroment là gì? Tại sao phải dung

-> Thêm lớp cầu hình môi trường vào dự án

10, Hooks

-> Khái niệm Hooks

-> Sử dụng useState() hook

-> Sử dụng useEffect() hook

-> Sử dụng useCallback() hook

-> Sử dụng Refs & useRef() hook

-> Tự tạo hook (custom hook)

11, Các kỹ thuật tối ưu

-> Sử dụng props “key” sao cho đúng

-> Tối ưu Component với React.memo()

-> Hiểu cách React cập nhật lại DOM

-> Các kỹ thuật tránh re-render không cần thiết

-> Kỹ thuật Lazy Load Components

-> Sử dụng thư virnj Loadable

-> Component bậc cao (HOC)

12, Context API

-> Vấn đề khi truền props nhiều cấp

-> Sử dụng context API

TỰ HỌC JAVASCRIPT

**1, Array map() method**

* Phương thức map() của Array sẽ **tạo ra 1 mảng mới** là kết quả của việc gọi một hàm trả về mọi phần tử của mảng ban đầu
* Phương thức này gọi đến một hàm đã cung cấp một lần cho mỗi phần tử trong mảng theo thứ tự
* map() không thực thi hàm cho các phần tử trống
* map() không thay đổi mảng ban đầu

**Syntax :**

*array*.map(*function(currentValue, index, arr), thisValue*)

**Parameter Values :**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| function(*curentValue*, *index*, *arr*) | Required. A function to be run for each element in the array  **Function arguments:**  *currentValue*: Required. The value of the current element.  *index*: Optional. The array index of the curent element  *arr*: Optional. The array object the current element belongs to |
| *thisValue* | Optional. A value to be passed to the function to be used as its “this” value  If this parameter is empty, the value “undefined” will be passed as its “this” value |

**Return value** : An Array containing the result of calling the provided function for each element in the original array

**Example:**

const numbers = [65, 44, 12, 4];  
const newArr = numbers.map(myFunction)  
  
function myFunction(num) {  
  return num \* 10;  
}

// Result: newArr = [650,440,120,40];

**2, Array reduce() method**

* Phương thức reduce() thực thi một hàm giảm dần cho mỗi giá trị của mảng ban đầu
* Phương thức reduce() trả về một giá trị đơn là kết quả tích luỹ của hàm
* reduce() không thực thi hàm cho một mảng các phần tử trống
* reduce() không thay đổi mảng ban đầu

**Syntax :**

*array*.reduce(function(total, currentValue, currentIndex, arr), initialValue)

**Parameter Values :**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *function*(*total, currentValue, currentIndex, arr*) | Required. A function to be run for each element in the array  **Function arguments:**  *total*: Required. The initialValue or the previously returned value of the function  *currentValue*: Required. The value of the current element.  *currentIndex*: Optional. The index of the current element.  *arr* : Optional. The array object the current element belongs to. |
| *initialValue* | Optional. A value to be passed to the function as the initial value.  **Chú ý**: Nếu không truyền initalValue, mặc định ban đầu total sẽ lấy giá trị của phần tử đầu tiên của mảng, currentValue sẽ được lấy là giá trị của phần từ đầu tiên trong mảng |

**Return Value :** Returns the *accumulated result*(kết quả tích luỹ) from the last call of the callback function

**Example**

const numbers = [15.5, 2.3, 1.1, 4.7];  
document.getElementById("demo").innerHTML = numbers.reduce(getSum, 0);  
  
function getSum(total, num) {  
  return total + Math.round(num);  
}

**3, includes() method**

* Phương thức includes() trả về true nếu một mảng chứa một phần tử xác định, nếu không chứa phần tử đó sẽ trả về false.
* Phương thức includes() có phân biệt chữ hoa, chữ thường

**Syntax :**

*array*.includes(element, start)

**Parameter Values :**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *element* | Required. The element to search for |
| *start* | Optional. Default 0. At which position in the array to start search |

**Return Value :** A boolean

**Example :**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.includes("Banana", 3);   // Returns false

**4, Array forEach() method**

* Phương thức forEach() gọi một hàm mỗi lần cho từng phần tử trong mảng theo thức tự.
* forEach() không thực thi các phần tử mảng không có giá trị.

**Syntax :**

*array*.forEach(function(currentValue, index, arr), thisValue)

**Parameter Values :**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| function(*curentValue*, *index*, *arr*) | Required. A function to be run for each element in the array  **Function arguments:**  *currentValue*: Required. The value of the current element.  *index*: Optional. The array index of the curent element  *arr*: Optional. The array object the current element belongs to |
| *thisValue* | Optional. A value to be passed to the function to be used as its “this” value  If this parameter is empty, the value “undefined” will be passed as its “this” value |

**Example :**

let sum = 0;  
const numbers = [65, 44, 12, 4];  
numbers.forEach((item) => {

sum += item;

});  
//sum = 125;

**5, Array filter() method**

* Phương thức filter() tạo ra một mảng được lấp đầy bằng tất cả các phần tử của mảng ban đầu nếu thoả mãn hàm callback truyền vào
* filter() không thực thi hàm cho mảng các phần tử trống
* filter() không làm thay đổi mảng ban đầu

**Syntax :**

*array*.filter(function(currentValue, index, arr), thisValue)

**Parameter Values :**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| function(*curentValue*, *index*, *arr*) | Required. A function to be run for each element in the array  **Function arguments:**  *currentValue*: Required. The value of the current element.  *index*: Optional. The array index of the curent element  *arr*: Optional. The array object the current element belongs to |
| *thisValue* | Optional. A value to be passed to the function to be used as its “this” value  If this parameter is empty, the value “undefined” will be passed as its “this” value |

**Return Value :** An Array containing all the array elements that pass the test. If no elements pass the test it returns an empty array.

**Example :**

const ages = [32, 33, 12, 40];  
const ageToCheck = 18;

cónt newArr = ages.filter((age) => {

return age > ageToCheck;

})

// newArr = [32,33,40];

**6, Array some() method**

* Phương thức some() kiểm tra nếu bất kì phần tử nào trong mảng vượt qua bài test (được cung cấp bởi 1 hàm)
* some() thực thi hàm một lần với từng phần tử của mảng :
* Nếu tìm thấy một mảng phần tử nơi hàm trả về một giá trị true, phương thức some() trả về giá trị true (và không kiểm tra các giá trị còn lại nữa)
* Nếu không sẽ trả về giá trị false
* some() khong thực thi hàm cho mảng các phần tử trống
* some() không thay đổi mảng ban đầu

**Syntax :**

*array*.some(function(currentValue, index, arr), thisValue)

**Parameter Values :**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| function(*curentValue*, *index*, *arr*) | Required. A function to be run for each element in the array  **Function arguments:**  *currentValue*: Required. The value of the current element.  *index*: Optional. The array index of the curent element  *arr*: Optional. The array object the current element belongs to |
| *thisValue* | Optional. A value to be passed to the function to be used as its “this” value  If this parameter is empty, the value “undefined” will be passed as its “this” value |

**Return Value :** A boolean. Returns true if any of the elements in the array pass the test, otherwise it returns false.

**Example :**

const ages = [4, 12, 16, 20];  
const ageToCheck = 18;

const result = ages.some((age) => {

return age >= ageToCheck;

})

// result = true;

**7, Array every() method**

* Phương thức every() trả về true nếu mọi phần tử trong mảng vượt qua bài test(được cung cấp như một hàm)
* every() thực thi hàm một lần với từng phần tử của mảng :
* Nếu tìm thấy một phần tử nơi hàm trả về một giá trị false, phương thức every() trả về giá trị false (và không kiểm tra các giá trị còn lại nữa)
* Nếu không sẽ trả về giá trị true
* some() khong thực thi hàm cho mảng các phần tử trống
* some() không thay đổi mảng ban đầu

**Syntax :**

*array*.every(function(currentValue, index, arr), thisValue)

**Parameter Values :**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| function(*curentValue*, *index*, *arr*) | Required. A function to be run for each element in the array  **Function arguments:**  *currentValue*: Required. The value of the current element.  *index*: Optional. The array index of the curent element  *arr*: Optional. The array object the current element belongs to |
| *thisValue* | Optional. A value to be passed to the function to be used as its “this” value  If this parameter is empty, the value “undefined” will be passed as its “this” value |

**Return Value :** A boolean. Returns true if all the elements in the array pass the test, otherwise it returns false.

**Example :**

const ages = [4, 12, 16, 20];  
const ageToCheck = 18;

const result = ages.every((age) => {

return age >= ageToCheck;

})

// result = false;

**8, Promise**

Sync/Async

Javascript có các xử lý bất đồng bộ vì vậy js cần sử dụng callback để xử lý các thao tác bất đồng bộ này.

Vấn đề khi sử dụng callback : callback hell? Vấn đề khi viết code : pyramid of doom => giải quyết bằng promise

Các bước : Khởi tạo promisse -> Khởi tạo Excutor

Trạng thái của promise : pending (không gọi resolve hay reject sẽ bị memory leak), Fulfilled (thành công), Rejected (thất bại).

Khái niệm promise : là khái niệm sinh ra để xử lý các thao tác bất đồng bộ, trước khi có promise chúng ta thường sử dụng callback, và callback sẽ xảy ra một vấn đề callback hell. Promise sinh ra từ phiên bản es6 chúng ta có thể sử dụng nó để khắc phục tình trạng callback hell

Để sử dụng promise chúng ta sử dụng từ khoá new Promise() chuyền vào nó là một Excutor function, function này sẽ nhận 2 tham số là 2 function resolve() và reject(). Resolve() gọi nó khi thao tác xử lý logic thành công, reject() sẽ được gọi khi thao tác xử lý thất bại.

Khi sử dụng promise thì đối tượng promise được tạo ra, sử dụng qua 2 phương thức .then() và .catch(). 2 phương thức này đều nhận vào một callback function và nó sẽ được thực thi ở then() khi resolve() được thực thi và thực thi ở catch() khi reject() được thực thi.

// Object Constructor

promise = new Promise(

// Excutor : func này đc thực thi khi gọi đến new Promise()

function(resolve, reject) {

// Logic

// Thành công : resolve()

// Thất bại : reject()

}

)

// Tính chất chuỗi : chain

promise

.then(function() {

// Khi thằng promise, trong Excutor, resolve() được gọi => callback của then() được gọi

/\* Trong callback của .then() nếu không return ra một 1 promise thì .then() đằng sau sẽ được lọt vào luôn, còn nếu return ra 1 promise thì .then sau là việc bạn đang .then trên thằng promise được trả về trong callback của .then() trước. \*/

.

})

.catch(function() {

// Khi thằng promise, trong Excutor, reject() được gọi => callback của catch() được gọi

})

.finally(function() {

// Khi một trong resolve() hoặc reject() được gọi, callback của finally() được gọi

})

**9, Fetch**

API : Application programing interface

Backend -> API -> Fetch -> JSON/XML -> JSON.parse -> Javascript types -> Render ra giao diện với HTML

var postApi = ‘…’; //link api

Fetch() sử dụng promise, khi gọi fetch() sẽ nhận đc 1 promise

// strea

fetch(postApi)

.then(function (response) {

Return response.json(); // Trả về 1 promise

// JSON.parse : JSON -> Javascript types

})

.then(function(posts) {

Console.log(posts);

})

.catch(function(err) {

// Lọt vào catch khi thất bại

});

**10, JSON server**

HTTP protocol (Giao thức HTTP) : 4 phương thức chính :

+, Creat : Tạo mới -> POST

+, Read: Lấy dữ liệu -> GET

+, Update : chỉnh sửa -> PUT/PATCH

+, Delete : Xoá -> DELETE

Water fall: các ct nhỏ, có cấu trúc, khi thay đổi hay sửa đổi các yêu cầu phải thực hiện lại nên rủi ro của pm rất cao.

Rup : toàn bộ process workflows ở các phases, mỗi giai đoạn sẽ tập trung cho một quy trình chính, mỗi giai đoạn đều thực hiện kiểm thử để xác định lỗi của phần mềm. Các phần mềm có quy mô thực hiện theo rup sẽ quản trị được rủi ro. Mỗi giai đoạn sẽ hình thành các bước lặp

**ReactJS**

SPA – Single-Page Application

MPA – Multi-Page Application

* Ôn lại ES6+

**1, Arrow Function**

- Expression Function :

const logger = function(log) {

console.log(log);

}

-> Arrow function :

const logger = (log) => {

console.log(log);

}

Example 2 :

Const sum = (a,b) => a+b;

/\*

Bất kỳ thứ gì sau dấu “=>” ngoại trừ cặp ngoặc “ {} “ đều được hiểu là return tuôn  
Muốn return “ {} ” ta gom lại trong một cặp ngoặc đơn ().

Nếu arrow function chỉ có 1 tham số được truyền vào ta có thể bỏ đi cặp ngoặc tròn :

const logger = log => console.log(log);

Arrow function không có context (môi trường mới được tạo ra khi gọi hàm)

Không sử dụng để làm constructor function được

\*/

**2, JS Module**

<script **type= “module”** src=“”></script>

/\*

**export** ra cái gì thì có thể import được cái đó  
1 **module** chỉ **export default**

Khi **import** mà không có **destructuring** thì nó đang **import** từ cái **export** **default**

Nếu **import** có **destructuring** nghĩa là đang **import** từ những cái **export** bình thường

\*/

**export default** logger;n

**import** logger **from** ‘./logger.js’

// import tất cả những thứ từ file logger.js export và được as như “logger”

**import** \* **as** logger **from** ‘./logger.js’

**3, Enhanced Object literal**

Định nghĩa key: value cho object

Định nghĩa method cho object

Định nghĩa key cho object dưới dạng variable

Var name = ‘Javascirpt’;

Var price = 1000;

//Code thông thường

Var couse = {

name: name,

price: price,

getname: function() {

return name;

}

}

//Sử dụng enhanced object literals

Var couse = {

name,

price,

getname() {

return name;

}

}

Var fieldName = ‘name’;

Var fieldPrice = ‘price’;

Var course = {

[fieldName]: ‘Javascript’,

[fileldPrice]: 1000

}

**4, Destructuring: phân rã**

Var array = [‘Javascript’, ‘PHP’, ‘Ruby’];

Var [a, b, c] = array;

//a = Javascript, b = PHP, c = Ruby

Var [a, , c] = array;

//a = Javascript, c = Ruby

**5, Rest parameters**: **“…rest”**

Var [a,…rest] = array;

//a=Javascript, rest: [‘PHP’, ‘Ruby’]; rest là 1 object

// Với object phải truyền vào { } đúng thuộc tính của object đó

**6, Spread Operator**

Khi giải toán tử spread ( … ) trước một array hoặc object nó sẽ bỏ đi cặp ngoặc [] hoặc {}

Var arr1 = [‘Javascript’, ‘PHP’, ‘Ruby’]

Var arr2 = [‘ReactJS’, ‘NodeJS’]

// Spread giải arr2 và arr1 tạo thành arr3

Var arr3 = […arr2,…arr1]

Var obj1 = {

Name : ‘javascript’

}

Var obj1 = {

price : 1000

}

Var obj3 = {

…obj1,

…obj2

}

\* Phân biệt rest… và spread…

Rest : Khi sử dụng kết hợp với destructuring, sử dụng nó để định nghĩa ra tham số

Khi truyền đối số (trong một function) thì nó không phải là rest mà là spread. Khi sử dụng để giải trong một array hay một object khác nó là spread

**7, document.creatElement() method**

const root = document.getElementById(‘root’)

const h1 = document.creatElement(‘h1’)

h1.innertext = “Hello world”

h1.id = ‘heading’ // Thêm id cho thẻ h1

h1.className = ‘header’ // Thêm class cho thẻ h1

// Thêm style cho thẻ h1

h1.style.color = ‘red’

h1.style = ‘color : red, font-size: 16px’

Object.assign(h1.style, {

Color : red,

fontSize: 16px

})

// Thêm element vào root

root.appenChild(h1)

**8, React.createElement()**

Syntax : **React.creatElement(type, props, children\_1,…children\_n)**

**type** : các loại thẻ trong HTML. VD : h1, div, span, p, a,…

**props** : các thuộc tính DOM của thẻ đó

children\_1 -> children\_n : bóc tách thành đối số thứ 3, nhưng React vẫn gom children\_1 đến children\_n thành thuộc tính children trong props

const h1React = React. creatElement(‘h1’, {

title : ‘hello’,

className : ‘heading’

}, “Hello world”)

divReact = React.createElement(

'div',

{

className : 'post-item'

},

React.createElement('h2','title: Học React tại F8','Học ReactJS'),

React.createElement('p',null,'Học ReactJS từ cơ bản tới nâng cao')

)

**9, React-DOM**

React-DOM là một thư viện là cầu nối giữa React và DOM

Khi cài React-DOM sẽ có 1 biến global là ReactDOM

root = document.getElementById(‘root’)

const divReact = React.createElement(

'div',

{

className : 'post-item'

},

React.createElement('h2','title: Học React tại F8','Học ReactJS'),

React.createElement('p',null,'Học ReactJS từ cơ bản tới nâng cao')

)

reactDOM.render(divReact, root)

\* render() method :

**Syntax : render(element, container, callback)**

element : react Element

container : root element,

callback :

**10, JSX**

Example :

const arr = [

{

name : 'HTML, CSS',

},

{

name : 'Responsive web design',

},

{

name : 'ReactJS',

}

]

const root = document.getElementById("root");

const listItem = arr.map((*item*, *index*) => {

return <li *index*={*index*}>{*item*.name}</li>

})

const ul = <ul>{listItem}</ul>;

ReactDOM.render(ul, root)

JSX sử dụng thư viện Babel để chuyển đồ từ ES6+ về ES5, dễ dàng hơn trong việc creatElement

Để có thể viết code Javascript trong JSX cần sử dụng cặp dấu {}, nếu muốn thể hiện 1 object sử dụng { {object} }

Trong JSX không render được 2 element cùng 1 lúc, cầu wrap lại trong 1 thẻ khác, có thể là thẻ <div></div> hoặc sử dụng thẻ <React.fragment></ React.fragment>. Khi sử dụng React.fragment sẽ không sinh ra thêm thẻ nào trong DOM nhưng vẫn đảm bảo được cấu trúc wrap của JSX.

**11, props là gì?**

- *React elements :*

+, Sử dụng props giống như với attribute của thẻ HTML

+, 2 props class, for => className, htmlFor

+, Phải tuân thủ theo quy ước có sẵn

- *React components*

+, Sử dụng props giống như đối số cho function

+, Tự do đặt tên props

-> Đặt theo camleCase

-> \*Có thể bao gồm dấu gạch ngang

- **Chú ý:**

+, Prop “key” là một prop đặc biệt

+, Props cơ bản là đối số của Component -> props có thể là bất cứ kiểu dữ liệu gì

+, Sử dụng destructuring

// Destructuring

function Postitem(*{ image, title, description, published }*) {

return (

<div *className*="post-item">

<img *src*="{title }"></img>

<div *className*="post-title">{title} </div>

<div *className*="post-desc">{description} </div>

<div *className*="post-published">{published} </div>

</div>

)

}

// App.js

function App() {

return (

<div *id*="wrapper">

<*Postitem*

*title* = "C#(.Net) ....."

*image* = "url.img"

*description* = "Description.... "

*published* = "Mot ngay truoc..."

/>

</div>

)

}

ReactDOM.render(<*App* />, document.getElementById("root"))

Bài tập :

// title, thumbnail\_cdn, description, studensCount

const courses = [

{

"id": 2,

"category\_id": 2,

"title": "HTML, CSS từ Zero đến Hero",

"slug": "html-css",

"description": "Trong khóa này chúng ta sẽ cùng nhau xây dựng giao diện 2 trang web là The Band & Shopee.",

"thumbnail": "courses/2.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "R6plN3FvzFY",

"preview\_path": null,

"language": "html",

"syntax\_highligh": "language-html",

"level": "Trình độ cơ bản",

"priority": 10,

"students\_count": 60908,

"created\_at": "2020-04-10T14:23:13.000000Z",

"updated\_at": "2021-10-13T08:24:21.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/2.png"

},

{

"id": 1,

"category\_id": 1,

"title": "Javascript Cơ Bản",

"slug": "javascript-co-ban",

"description": "Học Javascript cơ bản phù hợp cho người chưa từng học lập trình. Với hơn 100 bài học và có bài tập thực hành sau mỗi bài học.",

"thumbnail": "courses/1.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "0SJE9dYdpps",

"preview\_path": null,

"language": "javascript",

"syntax\_highligh": "language-javascript",

"level": "Trình độ cơ bản",

"priority": 30,

"students\_count": 39500,

"created\_at": "2020-06-10T14:23:13.000000Z",

"updated\_at": "2021-10-13T08:26:31.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/1.png"

},

{

"id": 7,

"category\_id": 4,

"title": "Kiến Thức Nhập Môn",

"slug": "lessons-for-newbie",

"description": "Để có cái nhìn tổng quan về ngành IT - Lập trình web các bạn nên xem các videos tại khóa này trước nhé.",

"thumbnail": "courses/7.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "M62l1xA5Eu8",

"preview\_path": null,

"language": null,

"syntax\_highligh": null,

"level": "Trình độ cơ bản",

"priority": 0,

"students\_count": 33822,

"created\_at": "2020-02-10T14:23:13.000000Z",

"updated\_at": "2021-10-13T08:14:09.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/7.png"

},

{

"id": 3,

"category\_id": 2,

"title": "Responsive Với Grid System",

"slug": "responsive-web-design",

"description": "Trong khóa này chúng ta sẽ học về cách xây dựng giao diện web responsive với Grid System, tương tự Bootstrap 4.",

"thumbnail": "courses/3.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "uz5LIP85J5Y",

"preview\_path": null,

"language": "html",

"syntax\_highligh": "language-html",

"level": "Trình độ cơ bản",

"priority": 20,

"students\_count": 13651,

"created\_at": "2020-05-10T14:23:13.000000Z",

"updated\_at": "2021-10-13T08:09:21.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/3.png"

},

{

"id": 6,

"category\_id": 3,

"title": "Node & ExpressJS",

"slug": "nodejs",

"description": "Học Back-end với Node & ExpressJS framework, hiểu các khái niệm khi làm Back-end và xây dựng RESTful API cho trang web.",

"thumbnail": "courses/6.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "z2f7RHgvddc",

"preview\_path": null,

"language": "javascript",

"syntax\_highligh": "language-javascript",

"level": "Trình độ cơ bản",

"priority": 32,

"students\_count": 11519,

"created\_at": "2020-08-01T14:23:13.000000Z",

"updated\_at": "2021-10-13T08:23:24.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/6.png"

},

{

"id": 5,

"category\_id": 2,

"title": "HTML, CSS Tips & Tricks",

"slug": "html-css-tutorials",

"description": "Tutorials về HTML, CSS, UI, UX sẽ được tổng hợp tại khóa học này, các video có nội dung ngắn gọn, súc tích giúp học viên có thể ứng dụng ngay vào thực tế",

"thumbnail": "courses/5.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "nB6cJh\_bb1U",

"preview\_path": null,

"language": "html",

"syntax\_highligh": "language-html",

"level": "Trình độ cơ bản",

"priority": 40,

"students\_count": 8658,

"created\_at": "2020-03-10T14:23:13.000000Z",

"updated\_at": "2021-10-13T08:19:00.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/5.png"

},

{

"id": 12,

"category\_id": 1,

"title": "Javascript Nâng Cao",

"slug": "javascript-nang-cao",

"description": "Hiểu sâu hơn về cách Javascript hoạt động, tìm hiểu về IIFE, closure, reference types, this keyword, bind, call, apply, prototype, ...",

"thumbnail": "courses/12.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "MGhw6XliFgo",

"preview\_path": null,

"language": "javascript",

"syntax\_highligh": "language-javascript",

"level": "Trình độ nâng cao",

"priority": 31,

"students\_count": 7067,

"created\_at": "2021-04-03T14:23:13.000000Z",

"updated\_at": "2021-10-13T07:52:40.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/12.png"

},

{

"id": 13,

"category\_id": 0,

"title": "ReactJS",

"slug": "reactjs",

"description": "Khóa học ReactJS từ cơ bản tới nâng cao, kết quả của khóa học này là bạn có thể làm hầu hết các dự án thường gặp với ReactJS. Cuối khóa học này bạn sẽ sở hữu một dự án giống Tiktok.com, bạn có thể tự tin đi xin việc khi nắm chắc các kiến thức được chia sẻ trong khóa học này.",

"thumbnail": "courses/13/13.png",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "x0fSBAgBrOQ",

"preview\_path": null,

"language": "javascript",

"syntax\_highligh": "language-jsx",

"level": "Trình độ trung bình",

"priority": 0,

"students\_count": 5186,

"created\_at": null,

"updated\_at": "2021-10-13T08:26:38.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/13/13.png"

},

{

"id": 4,

"category\_id": 1,

"title": "Don't Touch Your Face",

"slug": "tool-canh-bao-so-len-mat",

"description": "Xây dựng ứng dụng đưa ra cảnh báo khi người dùng sờ tay lên mặt. Chúng ta sẽ sử dụng thư viện ReactJS & Tensoflow để hoàn thiện ứng dụng này.",

"thumbnail": "courses/4/4.jpeg",

"content": null,

"preview\_origin": "youtube",

"preview\_youtube\_id": "r6GWbQL-qwA",

"preview\_path": null,

"language": "javascript",

"syntax\_highligh": "language-javascript",

"level": "Trình độ cơ bản",

"priority": 50,

"students\_count": 3293,

"created\_at": "2020-01-10T14:23:13.000000Z",

"updated\_at": "2021-10-13T01:29:40.000000Z",

"thumbnail\_cdn": "https://cdn.fullstack.edu.vn/f8-learning/courses/4/4.jpeg"

}

]

// CourseItem

const CourseItem = ({ *course* }) => (

<div>

<h2>{course.title}</h2>

<img *src*={course.thumbnail\_cdn} *alt*="" />

<p>{course.description}</p>

<p>{course.students\_count}</p>

</div>

)

function App() {

return (

<div *id*="wrapper">

{courses.map(*course* => (

<*CourseItem*

*key*={*course*.id}

*course*={*course*}

/>

))}

</div>

)

}

ReactDOM.render(<*App* />, document.getElementById("root"))

**12, DOM events. Làm việc với JSX**

Dom events : DOM event trong React cần phải tuân thủ theo camelCase

- Component do chúng ta định nghĩa phải viết in hoa kí tự đầu tiên. (boolean, null và undefined sẽ không được render)

function Button({ *title*, *href*, *onClick* }) {

let Component = 'button'

const props = {}

if (href) {

props.href = href

Component = 'a'

}

if (onClick) {

props.onClick = onClick

}

return (

<*Component* {...props}>{title}</*Component*>

)

}

function App() {

return (

<div *id*="wrapper">

<*Button*

*title* = 'Click me'

*href* = 'https://fullstack.edu.vn/'

// onClick = {() => console.log(Math.random())}

/>

</div>

)

}

ReactDOM.render(<*App* />, document.getElementById("root"))

- Kết hợp toán tử logic để render dữ liệu theo điều kiện

function App() {

let firstAccess = false;

return (

<div *id*="wrapper">

{firstAccess && <h1>Welcome to F8</h1>}

</div>

)

}

ReactDOM.render(<*App* />, document.getElementById("root"))

Props trong JSX :

- <YourComponent proName1=”String literals” propName2={expression} />

- Props default to “true”

Vd : <YourComponent primary />

Console.log(primary) -> trả về true, muốn là false thì phải gán primary={false}

- Spread/Rest props

// {label, ...inputProps} là toán tử rest

const Input = ({*label*, ...*inputProps*}) => (

<div>

<label>{label}</label>

<input {...inputProps} /> //...inputProps là toán tử spread

</div>

)

function App() {

return (

<div *id*="wrapper">

<*Input* *label*="Fullname" *type*="text" *placeholder*="Input..." />

</div>

)

}

ReactDOM.render(<*App* />, document.getElementById("root"))

- Children Prop

Nếu đưa bất cứ children prop nào vào trong thẻ ví dụ : <Button>Click me</Button> thì sẽ nhận được qua prop “children”, nếu truyền vào trong thẻ ví dụ : <Button str= “Click me” /> thì nhận được qua tham số nào cũng đc.

- Render props

**13, NPM, NPX và YARN**

**- NPM :**

Khi cài thư viện về NPM có 2 cách cài

+, Project scope : cài trong phạm vi project cụ thể

npm i react react-dom (mỗi tên thư việc cách nhau 1 dấu cách)

npm install –save-dev react react-dom (cài vào devDependencies)

npm i –D react react-dom (viết tắt, cài vào devDependencies)

npm uninstall react react-dom (xoá thư viện)

+, Global scope : cài trong phạm vi toàn cầu, vào thư mục user của máy tính, sử dụng khi global không phụ thuộc vào các thứ viện đó.

npm i –global create-react-app (cài thư việc create react app)

npm i –g create-react-app (viết tắt)

npm uninstall –g create-react-app (xoá thư viện)

- NPX :

Ưu điểm : không phải cài thư viện lên máy của bạn

NPX trọc lên npm khi máy tính không có, luôn có phiên bản mới nhất

**14, Hooks**- Thực chất là method, hàm đc cung cấp bởi reactJS, mỗi method này có các tính năng, trường hợp cụ thể để sử dụng. Khi làm việc với function component cần các tính năng mà Hooks cung cấp từ đó có thể lấy ra các Hooks tương ứng thêm vào và dùng trong function component này.

- Chỉ dùng cho function component, không thể dùng cho class component

- Component đơn giản và trở nên dễ hiểu hơn khi sử dụng Hooks

+, Không bị chia logic ra như methods trong lifecycle của Class Component

+, Không cần sử dụng “this”

- Sử dụng Hooks khi nào?

+, Dự án mới => Hooks

+, Dự án cũ :

-> Component mới => Function component + Hooks

-> Component cũ => Giữ nguyên, có thời gian tối ưu sau

+, Logic nghiệp vụ cần sử dụng các tính chất của OOP => Class component

**I, useState : trạng thái của dữ liệu**

- Dùng khi nào ?

Khi muốn dữ liệu thay đổi thì giao diện tự động được cập nhật lại (render lại theo dữ liệu)

- Cách sử dụng :

import { useState } from 'react'

function Component() {

const [state, setState] = useState(initState)

}

- Import useState from react

- Thêm vào function component

- Tất cả các hook đều là hàm, useState nhận đối số initState, là giá trị khởi tạo. Giá trị khởi tạo chỉ dùng 1 lần cho lần đầu, sau đóng return ra 1 mảng gồm 2 phần tử [state, setState]. Trong lần đầu tiên, nó sẽ lấy giá trị khởi tạo initState gán cho state, setState là 1 hàm, dùng để set lại state.

- Lưu ý:

+, Component sẽ được re-render sau khi ‘setState’

+, Initial state chỉ dùng cho lần đầu

+, set State có thể nhận callback

+, Khi truyền function vào initial state, nó sẽ không dùng functin đó là initial state mà dùng cái mà hàm đó return để làm initial state

const orders = [100, 200, 300]

function App() {

const [counter, setCounter] = useState(() => {

const total = orders.reduce((*total*, *current*) => *total* + *current*)

return total;

})

}

+, setState là thay thế hoàn toàn state cũ bằng giá trị mới trong setState

**- Two-way Binding : rang buộc 2 chiều**

Ví dụ radio check

const courses = [

{

id : 1,

name : 'HTML, CSS'

},

{

id : 2,

name : 'Javascript'

},

{

id : 3,

name : 'ReactJS'

}

]

function App() {

const [checked, setChecked] = useState(2)

const handleSubmit = () => {

// Call API

console.log({ id : checked })

}

return (

<div *className*="App" *style*={{padding: 32}}>

{courses.map(*course* => (

<div *key*={*course*.id}>

<input

*type*="radio"

*checked*={checked === *course*.id}

*onChange*={() => setChecked(*course*.id)}

/>

{*course*.name}

</div>

))}

<button *onClick*={handleSubmit}>Register</button>

</div>

);

}

- Ví dụ checkbox vẫn sử dụng mảng courses ở ví dụ trên

function App() {

const [checked, setChecked] = useState([])

console.log(checked)

const handleCheck = (*id*) => {

setChecked(*prev* => {

const isChecked = checked.includes(*id*)

if (isChecked) {

return checked.filter(*item* => *item* !== *id*)

} else {

return [...*prev*, *id*]

}

})

}

const handleSubmit = () => {

// Call API

console.log({ id : checked })

}

return (

<div *className*="App" *style*={{padding: 32}}>

{courses.map(*course* => (

<div *key*={*course*.id}>

<input

*type*="checkbox"

*checked*={checked.includes(*course*.id)}

*onChange*={() => handleCheck(*course*.id)}

/>

{*course*.name}

</div>

))}

<button *onClick*={handleSubmit}>Register</button>

</div>

);

}

**- Bài tập Todo List**

import { useState, useEffect } from 'react'

function App() {

const [job, setJob] = useState('')

const [jobs, setJobs] = useState(() => {

const storageJobs = JSON.parse(localStorage.getItem('jobs'))

return storageJobs ?? []

})

const handleKeyDown = (*e*) => {

if (*e*.keyCode === 13) handleSubmit()

}

const handleSubmit = () => {

setJobs(*prev* => {

const newJobs = [...*prev*, job]

// Save to local storage

const jsonJobs = JSON.stringify(newJobs)

localStorage.setItem('jobs', jsonJobs)

return newJobs

});

setJob('')

}

const handleRemove = (*e*, *index*) => {

setJobs(*prev* => {

const newWorks = *prev*.filter((*item*, *indexRemove*) => *index* !== *indexRemove*)

localStorage.setItem('jobs', JSON.stringify(newWorks))

return newWorks

})

}

return (

<div *className*="App" *style*={{ padding: 32 }}>

<input

*value*={job}

*onChange*={*e* => setJob(*e*.target.value)}

*onKeyDown*={*e* => handleKeyDown(*e*)}

/>

<button *onClick*={handleSubmit}>Add</button>

<ul>

{jobs.map((*job*, *index*) => {

return (

<div>

<li *key*={*index*} *index*={*index*} *style*={{ display: 'inline-block' }}>{*job*}</li>

<button *onClick*={(*e*) => handleRemove(*e*, *index*)}>Xoa</button>

</div>

)

})}

</ul>

</div>

);

}

export default App;

**- Mount : lắp vào & Unmounted : tháo ra**

**II, useEffect()**

- Sử dụng khi thực hiện cái Side Effects

(Side Effect : nói tới chương trình phần mềm, khi có tác động xảy ra khiến dữ liệu của chương trình bị thay đổi

Gọi API lấy dữ liệu

Tương tác với DOM

Subscriptions

setTimeout, setInterval

)

- useEffect() được thực thi sau mỗi lần render, những lần render tiếp theo tuỳ thuộc vào deps

- Cú pháp : useEffect(callback, [deps])

Đối số thứ nhất (bắt buộc) là 1 callback để thực hiện cái side effect

Đối số thứ hai (không bắt buộc) là 1 mảng chứa những cái phụ thuộc về mặt dữ liệu

Calllback luôn được gọi sau khi component mouned

Cleanup : Trong callback của useEffect() có thể return ra 1 hàm, hàm này gọi là hàm cleanup function.

Cleanup function luôn được gọi trước khi component unmounted

Cleanup function luôn được gọi trước khi callback được gọi (trừ lần mounted)

***TH1 : useEffect(callback)***

Gọi callback mỗi khi component re-render

Gọi callback sau khi component thêm element vào DOM

***TH2 : useEffect(callback, [])***

Chỉ gọi callback 1 lần sau khi component mount

***TH3 : useEffect(callback, [deps])***

Callback sẽ được gọi lại mỗi khi deps thay đổi

**III, useLayoutEffect()**

**-** Hoạt động gần giống useEffect()

- Mục đích : thực hiện các side effect, nhưng mặc định khi thực hiện cái side effect thì dung useEffect(), nhưng một vài tình huống hi hữu khi dung useEffect gặp vấn đề với giao diện người dung mới phải sử dụng useLayoutEffect()

- So sánh useEffect và useLayoutEffect

|  |  |
| --- | --- |
| useEffect | useLayout |
| Cập nhật lại state | Cập nhật lại state |
| Cập nhật DOM (mutated) | Cập nhật DOM (mutated) |
| Render lại UI | Gọi cleanup nếu deps thay đổi (sync) |
| Gọi cleanup nếu deps thay đổi | Gọi useLayoutEffect callback |
| Gọi useEffect callback | Render lại UI |

- Tình huống sử dụng : Khi setState rồi có điều kiện check state trong 1 useEffect rồi setState.

**IV, useRef()**

- Là một hàm, nhận đối số truyền vào initialValue có thể là bất cứ giá trị gì. Chỉ sử dụng initialValue trong cái lần đầu tiên mà Component được mounted, còn khi Component re-render thì không dùng giá trị initialValue nữa

- Return về 1 object {current: initialValue} có property là current, value là initialValue

- initialValue khi không truyền thì mặc định là undefined

**V, Custom Hook**

- Một Hook do mình tạo ra, custom hook là một function hơi đặc biệt

- Có thể sử dụng các hooks khác như useState, useEffect, hoặc một custom hook khác

- Đặt tên Custom Hook theo dạng useClock(), useClickOutside(), useQuery(),…

Cách sử dụng : Khi nào dung custom hooks?

- Tách riêng biệt phần logic ra khỏi UI

- Chia sẻ logic giữa các component

**15, Memoization ? React.memo() ?**

**-** Là kỹ thuật giúp tang tốc độc xử lý máy tính bằng cách lưu trữ lại dữ liệu, kết quả của những lần tính toán trước đó để những lần sau không phải tính toán lại nếu gặp bộ input cũ

+, Tính toán và lưu kết quả cho từng bộ input

+, Khi gặp lại bộ input đã từng làm thì không tính toán lại, mà trả về kết quả sẵn có

**\* HOC React.memo()** :

+, Là một HOC, chứ không phải hooks

+, Tương tự như PureComponent, nhưng PureComponent thì cho class component, còn React.memo() dung cho functional component

+, Chỉ render lại component nếu props thay đổi

+, Sử dụng shallow comparison