

Mục lục

1.	ReactJS là gì ? Các khái niệm cần biết trước khi học React JS	1
1.	ReactJS là gì ?	1
2.	Các khái niệm cơ bản về ReactJS.....	1
i.	Virtual DOM.....	1
ii.	JSX.....	3
iii.	Components	3
iv.	Props và State.....	4
v.	React Lifecycle	5
2.	Cài đặt môi trường chạy ReactJS	6
3.	Cài đặt NodeJS và NPM.....	6
4.	Khởi tạo ReactJS App	6
vi.	Khởi chạy dự án ReactJS.....	8
vii.	Xây dựng ứng dụng ReactJS đầu tiên.....	8
3.	Giới thiệu JSX trong ReactJS	11
5.	JSX là gì ?	11
6.	JSX trong ReactJS	11
viii.	Gán một biểu thức trong JSX.....	12
ix.	JSX là một biểu thức.....	13
x.	Chỉ định attributes với JSX	13
xi.	Phần tử con trong JSX.....	14
xii.	JSX Object	14
xiii.	Ngăn chặn Injection Attacks.....	15
4.	Components trong ReactJS	17
7.	Component trong ReactJS là gì?	17
8.	Khởi tạo một React Component	18
xiv.	Functional Component.....	18
xv.	Class Component.....	20
5.	Tìm hiểu về Props trong ReactJS	22
9.	Props là gì ?.....	22
10.	Props trong React	22
xvi.	Truyền props trong các components.....	23
xvii.	Nhận props trong components.....	23
11.	Ví dụ thực tế.....	24
6.	Tìm hiểu State trong ReactJS	27

12.	State trong ReactJS là gì ?.....	27
13.	Thao tác với state trong ReactJS	27
xviii.	Khởi tạo một state.....	27
xix.	Cập nhật một state.....	28
14.	Sự khác nhau giữa props và state	30
7.	Tìm hiểu Props Validation trong ReactJS.....	32
15.	Props validation là gì ?	32
16.	Validating props.....	34
xx.	Thuộc tính propTypes	34
xxi.	Thư viện PropTypes.....	36
xxii.	Ví dụ thực tế	40
8.	Tìm hiểu về Component API trong ReactJS.....	44
17.	Set State API.....	44
18.	Force Update.....	46
19.	Find Dom Node	47
20.	Bind function.....	48
9.	Tìm hiểu Component Life Cycle trong ReactJS	51
21.	Component Life Cycle là gì ?	51
22.	Component Life Cycle	51
xxiii.	Initialization	51
xxiv.	Mounting.....	52
xxv.	Updating.....	53
xxvi.	Unmounting	55
10.	Handling Events (xử lý sự kiện) trong ReactJS	57
23.	Handling Events	57
24.	Lưu ý với this trong xử lý Events	59
xxvii.	Xây dựng ví dụ	60
11.	Xử lý Form trong ReactJS	64
25.	Thao tác với Form trong ReactJS.....	64
xxviii.	Lấy giá trị của input.....	64
xxix.	Submit Form	65
xxx.	Validation Form	66
26.	Xây dựng ví dụ form trong React JS	66
12.	Render với điều kiện trong ReactJS	72
27.	Gán element vào biến	73
28.	Biểu thức điều kiện trong JSX	74

29.	Ngăn chặn component render	75
13.	Tìm hiểu về List và Keys trong ReactJS	78
30.	Lists trong React	78
31.	Keys trong React.....	79
32.	Một vài lưu ý khi sử dụng Keys	80
xxxi.	Keys là duy nhất	80
xxxii.	Tránh chỉ định index làm key	82
14.	Kỹ thuật Lifting State Up trong ReactJS.....	83
33.	Lifting State Up trong React	83
34.	Ví dụ cụ thể	84
15.	Tìm hiểu về Refs trong ReactJS	91
35.	React Refs là gì?	91
36.	Sử dụng React Refs	91
xxxiii.	Khởi tạo một Ref.....	92
xxxiv.	Forwarding Refs	95
16.	Tìm hiểu về Context trong ReactJS	98
37.	Context trong ReactJS là gì ?.....	98
38.	Context API trong ReactJS.....	100
xxxv.	React.createContext	100
xxxvi.	Context.Provider	100
xxxvii.	Context.Consumer.....	100
xxxviii.	Class.contextType	101
39.	Sử dụng Context trong ReactJS.....	101
17.	Tìm hiểu về Fragments trong ReactJS	108
40.	Tại sao phải sử dụng Fragments.....	108
41.	Sử dụng fragments trong ReactJS	110
42.	Cú pháp của Fragments.....	111
xxxix.	React.Fragments	111
xl.	Viết tắt.....	112
18.	Tìm hiểu về Render Props trong ReactJS	113
43.	Render props trong ReactJS.....	113
44.	Triển khai ví dụ về render props trong React	114
xli.	Ví dụ 1.....	114
xlii.	Ví dụ 2.....	116
19.	Higher-Order Components trong ReactJS	120
45.	Higher-Order Components là gì ?	120

46.	Triển khai ví dụ	122
20.	Giới thiệu Hooks trong React JS	126
47.	React Hooks là gì?	126
48.	Hooks trong React JS cơ bản	128
xlili.	useState()	129
xliv.	useEffect().....	129
xlv.	useContext().....	129
xlvi.	useReducer()	130
21.	Tìm hiểu React Hook useState	131
49.	useState trong React.....	131
50.	Xây dựng ví dụ	132
xlvii.	Ẩn/Hiện một component.....	132
xlviii.	Lấy dữ liệu từ API và hiển thị	135
22.	useEffect trong React Hooks	142
51.	useEffect trong React Hooks.....	142
52.	Sử dụng useEffect.....	142
xlix.	Sử dụng useEffect như componentDidMount	143
l.	Sử dụng useEffect như componentDidUpdate.....	146
li.	Sử dụng useEffect như componentWillUnmount	147
23.	useContext trong React Hook.....	150
53.	useContext là gì ?	150
54.	Ví dụ về useContext trong React.....	151
24.	Xây dựng Hook trong React JS (React Custom Hook)	157
55.	Custom Hooks là gì?	157
56.	Khi nào dùng Custom Hooks.....	158
57.	Tự xây dựng một custom hook.....	160
25.	Redux là gì? Tại sao lại ứng dụng trong ReactJS	164
58.	Cần hiểu trước khi tìm hiểu Redux là gì	164
lii.	State là gì?	164
liii.	Props là gì?	165
59.	Vậy Redux là gì?	166
liv.	Tại sao chúng ta phải sử dụng Redux?	166
lv.	Redux hoạt động như thế nào?	167
26.	React Router cơ bản.....	170
60.	React Router là gì ?.....	170
61.	Sử dụng React Router trong ReactJS.....	170

62.	Xây dựng ví dụ	170
27.	Tích hợp Redux vào ReactJS	176
63.	Cài đặt Redux.....	176
64.	Tích hợp Redux vào ReactJS	176
lvi.	Khởi tạo các hằng.....	177
lvii.	Khởi tạo actions	177
lviii.	Khởi tạo reducers.....	178
lix.	Tích hợp Redux	179
lx.	Lấy và cập nhật giá trị của state từ Store.....	180
28.	Cách đẩy ứng dụng ReactJS lên Heroku và Deploy trên đó.....	183
65.	Cài đặt phần mềm hỗ trợ đưa ứng dụng lên Heroku.....	183
lxi.	Window và MacOS.....	183
lxii.	Ubuntu	183
66.	Đưa ứng dụng ReactJS lên Heroku	184
lxiii.	Thêm Buildpacks.....	184
lxiv.	Đẩy source code lên Heroku	185
lxv.	Tinh chỉnh ứng dụng	186

1. ReactJS là gì ? Các khái niệm cần biết trước khi học React JS

Trong bài viết này chúng mình sẽ cùng nhau đi tìm hiểu về ReactJS là gì và các khái niệm cơ bản để bắt đầu làm quen với nó. Đây là bài viết đầu tiên trong loạt bài về ReactJS, vì vậy bài viết chỉ tập trung vào giới thiệu và giải thích các khái niệm liên quan.

Table of Content

- [1. ReactJS là gì ?](#)
- [2. Các khái niệm cơ bản về ReactJS](#)
 - [Virtual DOM](#)
 - [JSX](#)
 - [Components](#)
 - [Props và State](#)
 - [React Lifecycle](#)

1. ReactJS là gì ?

ReactJS là một thư viện JavaScript có tính hiệu quả và linh hoạt để xây dựng các thành phần giao diện người dùng (UI) có thể sử dụng lại. **ReactJS** giúp phân chia các UI phức tạp thành các thành phần nhỏ (được gọi là *component*). Nó được tạo ra bởi Jordan Walke, một kỹ sư phần mềm tại Facebook. **ReactJS** ban đầu được phát triển và duy trì bởi Facebook và sau đó được sử dụng trong các sản phẩm của mình như **WhatsApp & Instagram**.

ReactJS được dùng để xây dựng các ứng dụng [*single page application*] (*SPA*). Một trong những điểm hấp dẫn của ReactJS là nó không chỉ được xây dựng bên phía clients mà còn sử dụng được bên phía server.

2. Các khái niệm cơ bản về ReactJS

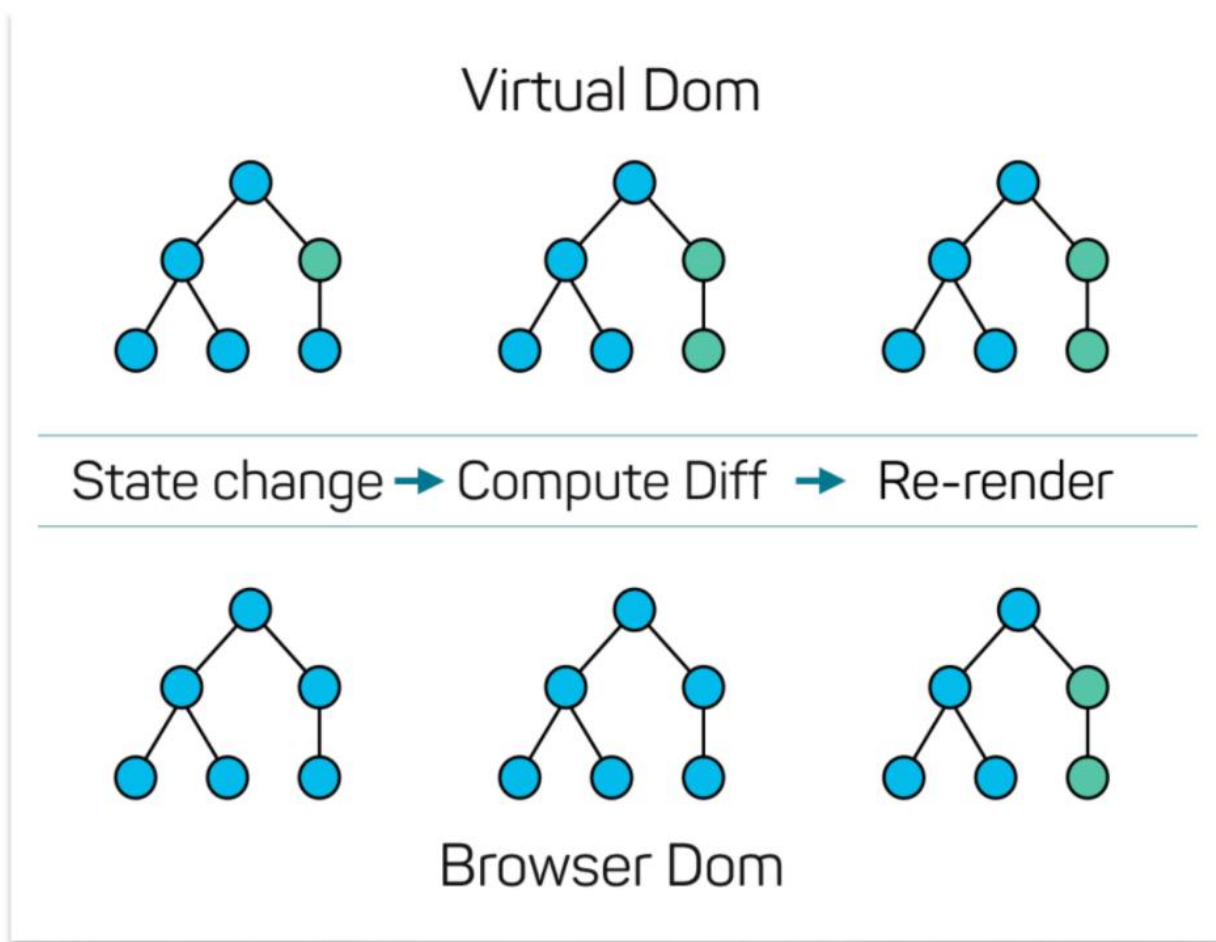
Khi bắt đầu làm quen với **ReactJS**, chúng ta nên làm quen với các khái niệm cơ bản của nó trước, bởi các khái niệm này sẽ đi cùng với chúng ta trong suốt quá trình học tập và làm việc với **ReactJS** sau này.

i. Virtual DOM

Để hiểu rõ khái niệm về Virtual DOM, chúng ta cùng nhau đi tìm hiểu về DOM trước. DOM là một *Document Object Model* và là một cấu trúc trừu tượng của text. Các đoạn mã HTML được gọi là HTML DOM. Mỗi elements trong HTML là các nodes của DOM đó.

Tại sao có DOM rồi lại cần Virtual DOM (*DOM ảo*)? Khi chúng ta làm việc với một DOM, khi một nodes thay đổi thì tất cả các nodes cũng phải thay đổi thay. Giả sử, chúng ta có một list danh sách gồm 10 items, nếu chúng ta thay đổi 1 items thì DOM cũng thay đổi 9 items còn lại về trạng thái ban đầu của nó.

Điều này là không cần thiết, mặc dù tốc độ xử lý của DOM khá nhanh nhưng đối với các ứng dụng SPA việc thay đổi các DOM này là liên tục nên nó sẽ xảy ra khá chậm và không khả thi đi xây dựng ứng dụng lớn. Lúc này Virtual DOM sẽ được dùng để thay thế. Nó được xây dựng dựa trên DOM thật, có một vài thuộc tính của DOM thật nhưng khi thay đổi Virtual DOM sẽ không thực hiện thay đổi trên màn hình giống như DOM thật.



Khi chúng ta thực hiện render một JSX element, mỗi Virtual DOM object sẽ được cập nhật, khi virtual DOM được cập nhật, **ReactJS** sẽ so sánh virtual DOM với virtual DOM trước đó để kiểm tra trước khi thực hiện cập nhật và sau đó sẽ cập nhật trên một phần của DOM thật. Thay đổi của DOM thật sẽ được hiển thị ra màn hình.

Quay lại ví dụ bên trên, thì lúc này khi chúng ta sử dụng Virtual DOM thì nó chỉ cập nhật duy nhất 1 items, lúc này tài nguyên sẽ được tiết kiệm cũng như tốc độ xử lý cũng nhanh hơn rất nhiều.

ii. JSX

JSX là viết tắt là Javascript XML, nó cho phép bạn viết các đoạn mã HTML trong React một cách dễ dàng và có cấu trúc hơn. Về cú pháp cũng gần tương tự như HTML, giả sử mình có 1 đoạn mã HTML như sau:

```
1 <p class="text">freetuts.net</p>
```

Thì trong JSX thì sẽ được viết như này :

```
1 <p className="text">freetuts.net</p>
```

chỉ cần thay **class** thành **className** là xong, ngoài ra còn một vài cú pháp đặc biệt của JSX mình sẽ giới thiệu ở bài viết tiếp theo.

iii. Components

Khi bạn làm việc với một dự án lớn, UI có độ phức tạp cao chia thành các phần khác nhau. Việc chia nhỏ các thành phần trong UI là một điều cần thiết, các phần nhỏ này được gọi là các components, cho phép render các đoạn mã HTML,... Trong **ReactJS** cách viết components được chia thành 2 loại:

- class components
- function components.

```
1 //Function component
2 function Clock(props) {
3   return (
4     <div>
5       <h1>Hello, world!</h1>
6     </div>
7   );
8 }

1 //Class component
2 class Clock extends React.Component {
```



```

3      render() {
4          return (
5              <div>
6                  <h1>Hello, world!</h1>
7              </div>
8          );
9      }
10 }

```

Mỗi loại sẽ có ưu và nhược điểm khác nhau, trong bài viết về phần này mình sẽ giới thiệu kĩ hơn.

iv. Props và State

Props là một tham số được chuyển qua lại giữa các React Components, các **props** này được truyền qua các component với cú pháp giống như là HTML attributes.

```

1    <Post title="Học ReactJS cùng Freetuts.net">

```

State là một object mà lưu trữ giá trị của các thuộc tính bên trong components và chỉ tồn tại trong phạm vi của component đó. Mỗi khi bạn thay đổi giá trị của một **state** thì component đó sẽ được render lại.

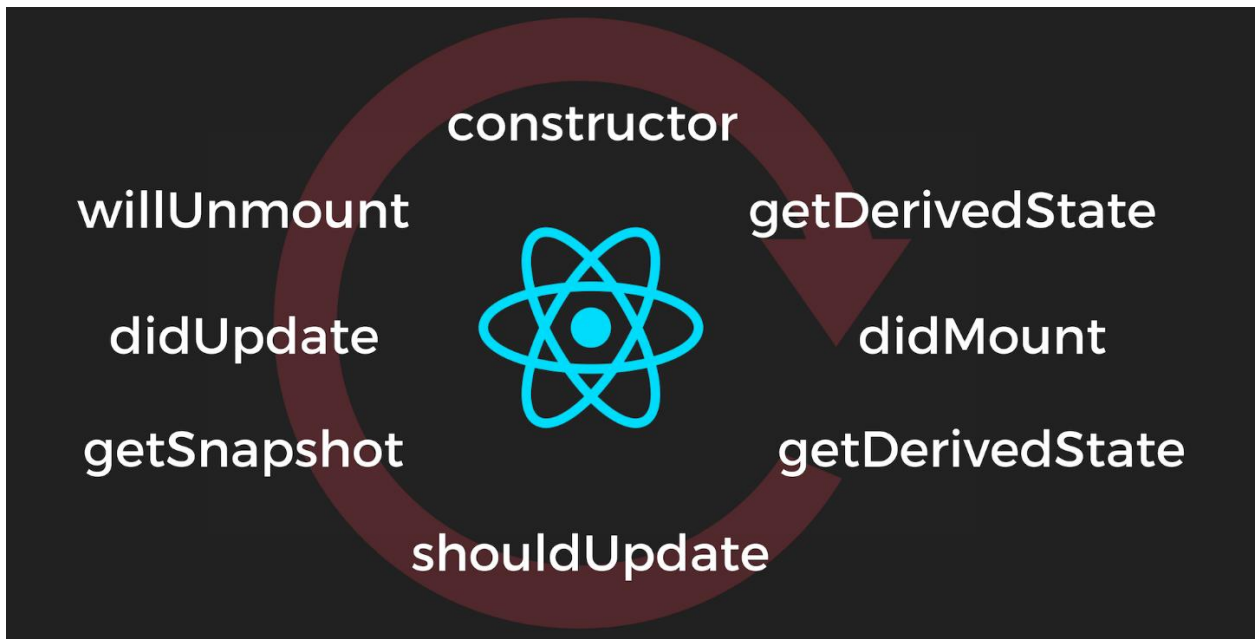
```

1    class Car extends React.Component {
2        constructor(props) {
3            super(props);
4            this.state = {brand: "Ford"};
5        }
6        render() {
7            return (
8                <div>
9                    <h1>My Car</h1>
10                </div>
11            );
12        }

```

v. React Lifecycle

React Lifecycle là một vòng đời của component, khi chúng ta tiến hành render một component thì **ReactJS** thực hiện nhiều tiến trình khác nhau, các tiến trình này được lặp đi lặp lại đối với các component.



Giả sử khi một component được gọi trước tiên nó sẽ cài đặt props và state, sau đó tiến hành mounting, update, unmounting,...việc tham gia vào quá trình này bạn cần sử dụng đến các hàm hỗ trợ của lifecycle.

2. Cài đặt môi trường chạy ReactJS

Trong bài viết này chúng ta sẽ đi tìm hiểu về **cách cài đặt môi trường chạy ReactJS**, đối với những bạn mới bắt đầu làm quen với ReactJS thì đây là bước đầu tiên để khởi chạy một project hello world. Các bước cài đặt không quá phức tạp bạn chỉ cần thực hiện theo từng bước bên dưới là có thể cài đặt môi trường khởi chạy cho dự án ReactJS của mình rồi.

Table of Content

- [1. Cài đặt NodeJS và NPM](#)
- [2. Khởi tạo ReactJS App](#)
 - [Khởi chạy dự án ReactJS](#)
 - [Xây dựng ứng dụng ReactJS đầu tiên](#)

3. Cài đặt NodeJS và NPM

Để cài đặt môi trường chạy ReactJS trước tiên bạn phải cài đặt NodeJS và NPM, nó là một nền tảng bắt buộc, bạn có thể xem bài viết về [cách cài đặt NodeJS và NPM](#) để hiểu rõ hơn về cách cài đặt nó. Thông thường NPM sẽ được cài đặt kèm theo khi bạn cài đặt NodeJS.

Để kiểm tra xem NodeJS và NPM đã được cài đặt trong máy chưa? Chúng ta có thể mở terminal và gõ dòng lệnh:

```
1 node -v
2 npm -v
```

Khi cài đặt thành công, sẽ hiển thị ra phiên bản NodeJS và NPM hiện đã được cài đặt:

```
1 tri@tri:~$ npm -v
2 6.13.4
3 tri@tri:~$ node -v
4 v10.19.0
```

4. Khởi tạo ReactJS App

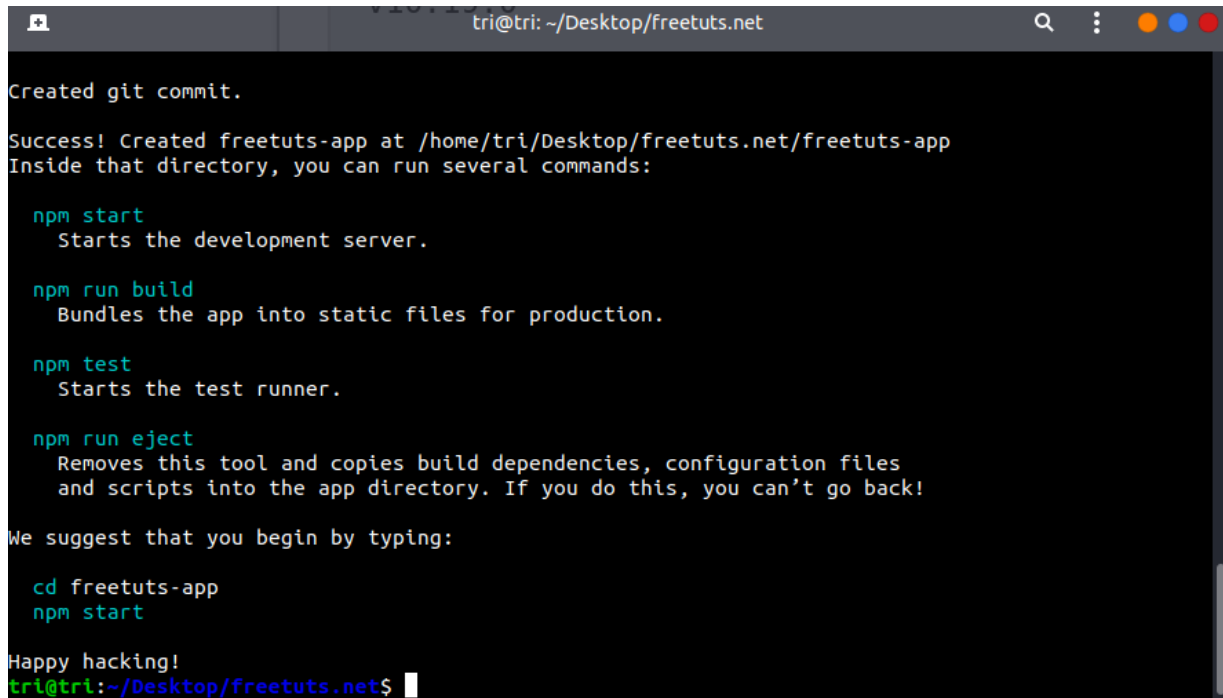
Tiếp theo, bạn có thể cài đặt ReactJS bằng cách vào thư mục chứa dự án và mở terminal và gõ dòng lệnh:

```
1 npx create-react-app my-app
```

Trong đó `my-app` là tên thư mục chứa dự án của bạn, giả sử mình muốn khởi tạo dự án có tên `freetuts-app`, chúng ta sẽ có:

```
1 npx create-react-app freetuts-app
```

Bạn đợi một khoảng thời gian cho quá trình cài đặt hoàn tất, sau khi cài đặt thành công bạn sẽ thấy terminal hiển thị như hình:



```
tri@tri: ~/Desktop/freetuts.net
Created git commit.
Success! Created freetuts-app at /home/tri/Desktop/freetuts.net/freetuts-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

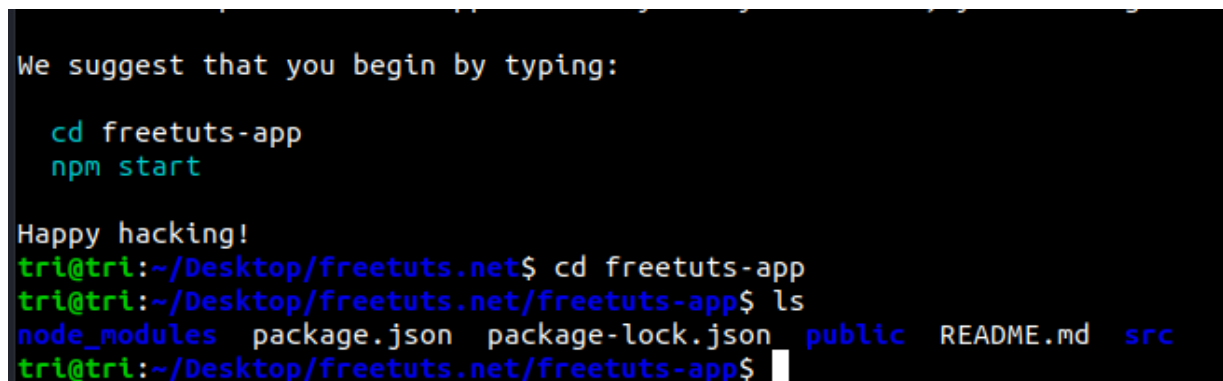
  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd freetuts-app
  npm start

Happy hacking!
tri@tri:~/Desktop/freetuts.net$
```

Vào thư mục vừa được khởi tạo có tên `freetuts-app` chúng ta sẽ thấy được các thư mục được tạo tự động, chú ý đến 2 thư mục chính đó là `src` và `public`:



```
We suggest that you begin by typing:

  cd freetuts-app
  npm start

Happy hacking!
tri@tri:~/Desktop/freetuts.net$ cd freetuts-app
tri@tri:~/Desktop/freetuts.net/freetuts-app$ ls
node_modules  package.json  package-lock.json  public  README.md  src
tri@tri:~/Desktop/freetuts.net/freetuts-app$
```

Chúng ta sẽ tìm hiểu nhiệm vụ của từng thư mục được khởi tạo :

- `src` sẽ chứa những đoạn mã mà chúng ta viết sau này,
- `public` sẽ chứa các file ảnh, css, js,...hay bất cứ thứ gì mà bạn muốn,..
- `node_modules`: các module cài tự động khi tạo react app, bạn không cần phải quan tâm đến nó.
- `package.json` và `package-lock.json`: chứa thông tin của các module cần thiết.

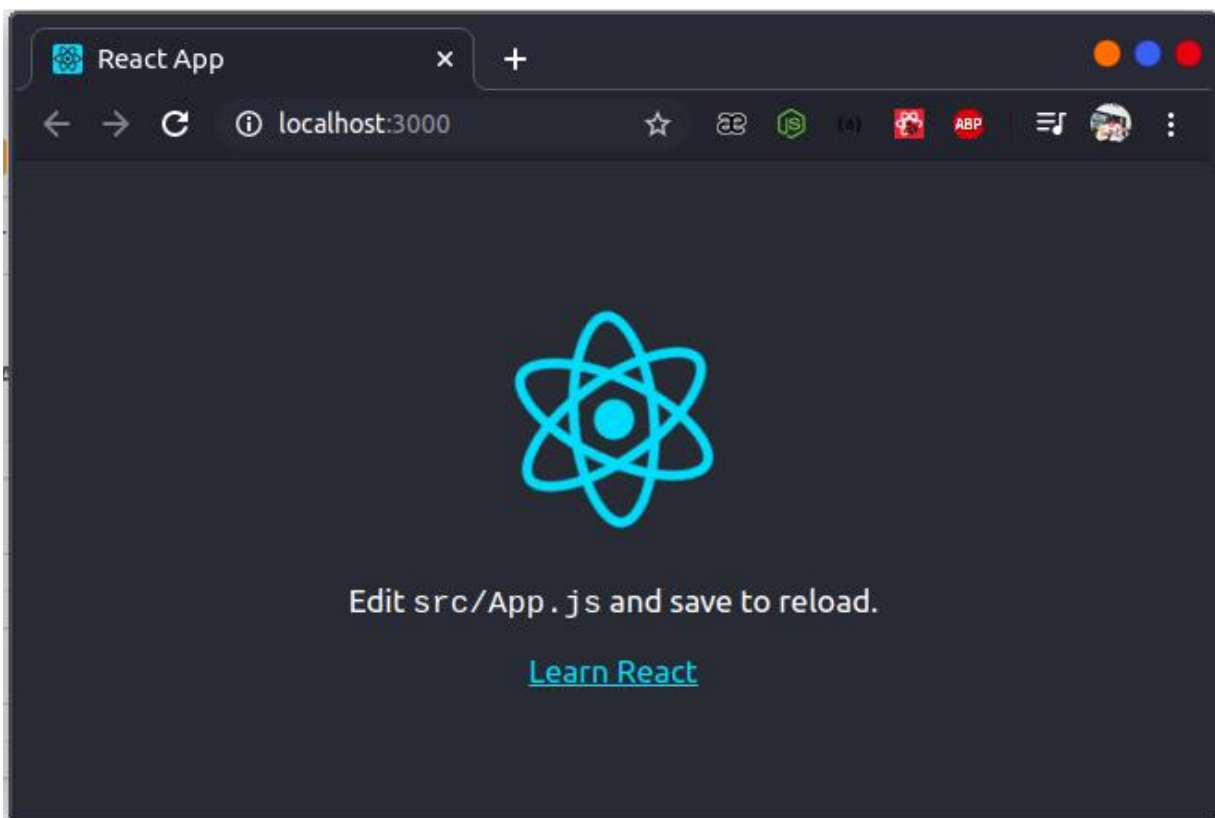
Ngoài ra bạn còn có thể thêm các thư mục theo mục đích mà bạn muốn sử dụng.

vi. Khởi chạy dự án ReactJS

Để khởi chạy dự án bạn cần phải truy cập vào thư mục vừa được khởi tạo mà mở terminal:

```
1 npm start
```

Lúc này, ReactJS sẽ khởi chạy dự án mặc định ở port `3000`, chúng ta có thể mở trình duyệt và truy cập đường dẫn `http://localhost:3000`



vii. Xây dựng ứng dụng ReactJS đầu tiên

Chúng ta sẽ đi xây dựng một ứng dụng ReactJS đầu tiên bằng cách truy cập vào thư mục `src` trong dự án, như bên trên mình đã đề cập thì nó là thư mục chứa mã nguồn, tìm đến file `App.js` và sửa thành:

```

1   import React from 'react';
2   function App() {
3     return (
4       <div>
5         <h1>Hello React.js - Freetuts.net</h1>
6       </div>
7     );
8   }
9
10  export default App;

```

Một điều chú ý là trong file `index.js`

```

1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   import * as serviceWorker from './serviceWorker';
6
7   ReactDOM.render(
8     <React.StrictMode>
9     <App />
10    </React.StrictMode>,
11    document.getElementById('root')
12  );
13
14  // If you want your app to work offline and load faster, you can change
15  // unregister() to register() below. Note this comes with some pitfalls.
16  // Learn more about service workers: https://bit.ly/CRA-PWA

```

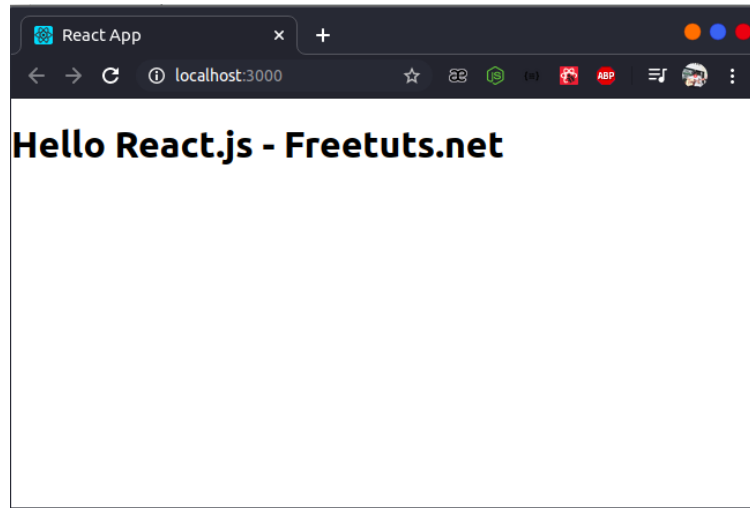
```
17     serviceWorker.unregister();
```

Chú ý đến thư đoạn `document.getElementById('root')`, đây là ví trị mà tất cả các component sẽ render ra. Bạn có thể tìm id root ở trong file `public/index.html`.

Tiến hành khởi chạy dự án bằng cách mở terminal và gõ dòng lệnh :

```
1     npm start
```

Truy cập vào đường dẫn **`http://localhost:3000`** chúng ta sẽ thấy kết quả:



Một lưu ý khi khởi chạy dự án thành công khi bạn sửa đổi các file thì ReactJS sẽ tự động reload lại sự thay đổi đó. Bạn không cần phải khởi động lại server bằng cách thủ công nữa.

3. Giới thiệu JSX trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về JSX trong ReactJS. Ở bài viết trước mình cũng đã giới thiệu về JSX một cách căn bản nhất, bài này chúng ta sẽ tiếp tục và đi tìm hiểu sâu hơn về cú pháp của JSX trong ReactJS, cũng như cách sử dụng nó.

Table of Content

- [1. JSX là gì ?](#)
- [2. JSX trong ReactJS](#)
 - [Gán một biểu thức trong JSX](#)
 - [JSX là một biểu thức](#)
 - [Chỉ định attributes với JSX](#)
 - [Phần tử con trong JSX](#)
 - [JSX Object](#)
 - [Ngăn chặn Injection Attacks](#)

5. JSX là gì ?

JSX là viết tắt là Javascript XML, một template languages nhưng nó lại mang hầu hết tính năng của Javascript. Nó cho phép bạn viết các đoạn mã HTML trong React một cách dễ dàng và có cấu trúc hơn. React sử dụng JSX cho việc xây dựng bố cục thay vì javascript thông thường. JSX giúp tạo ra các React 'elements'. Việc sử dụng nó trong ReactJS rất hữu ích bởi:

- JSX giúp cho việc xây dựng các ứng dụng React một cách nhanh hơn, dễ tối ưu trong việc compile code sang javascript.
- JSX rất dễ xem các lỗi trong quá trình triển khai bởi hầu hết các lỗi sẽ được hiển thị trong quá trình compile, không như các đoạn mã HTML có thể thừa thiếu các thẻ div khiến giao diện bị hiển thị sai. JSX lại hoàn toàn ngược lại, khi bạn quên đóng div chẳng hạn thì nó lập tức sẽ hiển thị lỗi.
- Cú pháp khá giống với HTML nên dễ dàng cho việc viết chuyển đổi.

Ngoài ra bạn có thể tham khảo thêm về JSX.

6. JSX trong ReactJS

Trong **ReactJS** không bắt buộc bạn phải sử dụng **JSX** nhưng hầu hết mọi người đều sử dụng nó bởi đây là cách hữu ích nhất để làm việc với các UI bên trong Javascript code. JSX cũng cho phép React hiển thị đầy đủ các lỗi nhất và hiệu quả hơn.

viii. Gán một biểu thức trong JSX

Giả sử bạn muốn gán một biểu thức trong JSX, trong ví dụ bên dưới mình sẽ gán biến `name` vào trong JSX bằng cách bọc nó trong dấu `{` :

```
1  const name = 'Freetuts.net';
2  const element = <h1>Welcome to {name}</h1>;
3
4  ReactDOM.render(
5    element,
6    document.getElementById('root')
7  );
```

Ngoài ra bạn có thêm bất cứ biểu thức javascript nào vào trong dấu ngoặc kép này như `info.name`, `1+1`, `formatMoney(10000)`,...Như trong ví dụ dưới đây mình sử dụng hàm `formatName` :

```
1  function formatName(user) {
2    return user.firstName + ' ' + user.lastName;
3  }
4
5  const user = {
6    firstName: 'Nguyễn',
7    lastName: 'Trí'
8  };
9
10 const element = (
11   <h1>
12     Xin chào, {formatName(user)}!
```

```

13     </h1>
14   );
15
16   ReactDOM.render (
17     element,
18     document.getElementById('root')
19   );

```

ix. JSX là một biểu thức

Sau khi compile, các đoạn đoạn mã JSX sẽ như các object Javascript thông thường, cho phép bạn có thể gọi hoặc làm bất cứ gì với nó.

Có nghĩa là bạn có thể sử dụng JSX bên trong *if, for, function,...* hay là chỉ định nó làm giá trị của một biến,... Trong ví dụ mình có một hàm trả về một JSX:

```

1   function sayHi(name) {
2     if(name) {
3       return <p>Xin chào, {name} !</p>
4     }else{
5       return <p>Xin chào bạn !</p>
6     }
7   }

```

x. Chỉ định attributes với JSX

Bạn cũng có thể chỉ định một attribute trong JSX, cú pháp giống như HTML thông thường :

```

1   const element = <div tabIndex="0"></div>;

```

hay chỉ định attributes với JSX bằng biểu thức javascript như này:

```

1   const element = <img src={user.avatarUrl}></img>;

```

Bạn nên dùng dấu *ngoặc kép* (`""`) cho giá trị chuỗi và *ngoặc nhọn* (`{ }`) cho biểu thức như trong ví dụ trên, React khuyên chúng ta không nên dùng cả 2 cái lồng nhau như thế này :

```
1    const element = <div tabIndex={"1"}></div>;
```

Quy ước đặt tên của JSX gần giống với HTML, React DOM sử dụng thuộc tính `camelCase` cho tên của thuộc tính cho phép chuyển đổi dễ hơn giữa HTML và JSX. Ví dụ trong HTML có thuộc tính `class`, JSX sẽ chuyển thành `className`, `tabindex` -> `tabIndex`.

xi. Phần tử con trong JSX

Nếu chỉ có một tag bạn chỉ cần đóng nó bằng dấu `/>` như ví dụ :

```
1    const element = <img src={user.avatarUrl} />;
```

trong trường hợp trong tag có nhiều phần tử con bạn cần phải bọc ngoài nó bằng một JSX tags:

```
1    //Đúng cú pháp
2    //Phải bọc nó bằng một tags
3    const element = (
4        <div>
5            <h1>Hello</h1>
6            <p>Welcome to Freetuts</p>
7        </div>
8    );
9
10   //Viết sai
11   //Các phần tử không được bọc
12   const element = (
13       <h1>Hello</h1>
14       <p>Welcome to Freetuts</p>
15   );
```

xii. JSX Object

Để compile một JSX object thành JSX thông thường chúng ta sử dụng `React.createElement()` như ví dụ :

```

1   const element = React.createElement(
2     "p",
3     { className: "welcome" },
4     "Welcome to Freetuts.net!"
5   );
6
7   const element = <p className="welcome">Welcome to Freetuts.net!</p>

```

JSX object cho phép bạn tạo ra các JSX dễ dàng debug hơn, ngoài ra JSX object còn có thể được viết theo dạng như:

```

1   const element = {
2     type: "p",
3     props: {
4       className: "welcome",
5       children: "Welcome to Freetuts.net!"
6     }
7   };
8
9   const element = <p className="welcome">Welcome to Freetuts.net!</p>;

```

xiii. Ngăn chặn Injection Attacks

Đây là một tính năng bảo mật của React, React DOM sẽ tiến hành **escaped** tất cả các giá trị bên trong JSX một cách tự động trước khi render chúng, điều này rất hữu ích cho việc ngăn chặn các hình thức tấn công bằng cách tiêm mã độc.

```

1   //Khi sử dụng trong React sẽ không nguy hiểm
2   const content = '<script>XSS</script>'
3   const element = <p className="welcome">{content}</p>;

```


4. Components trong ReactJS

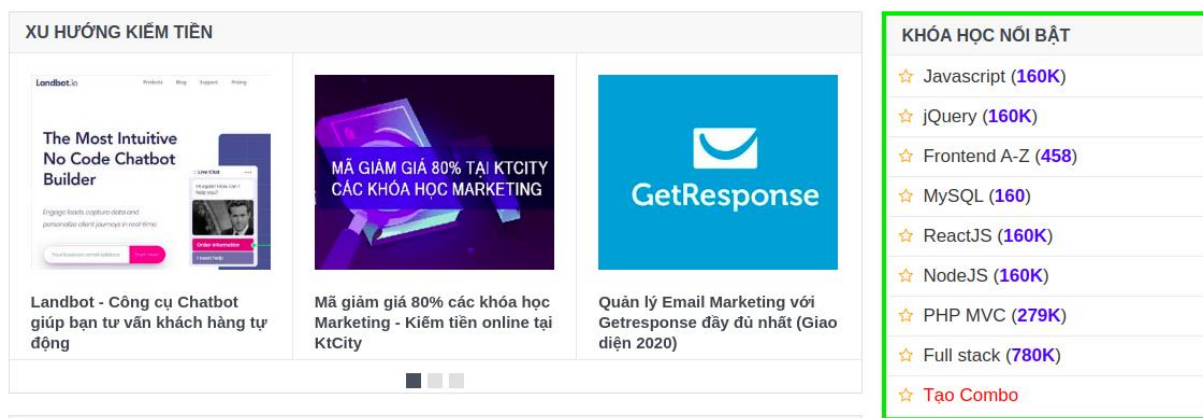
Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về **components trong ReactJS**. Trong bài viết đầu tiên giới thiệu về ReactJS mình cũng đã đề cập đến **component**, đây là một phần không thể thiếu trong dự án **ReactJS**. Bài này chúng ta sẽ cùng nhau đi tìm hiểu kĩ hơn về nó.

Table of Content

- [1. Component trong ReactJS là gì?](#)
- [2. Khởi tạo một React Component](#)
 - [Functional Component](#)
 - [Class Component](#)

7. Component trong ReactJS là gì?

Components giúp phân chia các UI (giao diện người dùng) thành các phần nhỏ để dễ dàng quản lý và tái sử dụng. Giả sử mình có một website gồm nhiều phần bố cục khác nhau và mình muốn chia nhỏ các phần ra để dễ quản lý.



Ở hình ảnh bên trên chúng ta có thể chia nó thành 2 **components**, đó là phần "**khóa học nổi bật**" và "**xu hướng kiểm tiền**". Mỗi **components** sẽ đảm nhiệm phần hiển thị khác nhau. Khi bạn muốn làm một trang hoàn chỉnh chỉ ghép các components này lại với nhau.

Trong mỗi React App đều có thể chứa rất nhiều components, mỗi **components** trong đó thường nhận về các **props** và trả về **React elements** từ đó hiển thị ra cho UI. **Components trong React** thường được viết theo 2 loại chính đó là functional component và class components. Bên dưới là một functional components:

```
1    const App = () => <h1>Hello Freetuts.net</h1>;
```

Components bên trên được viết theo cú pháp ES6, không nhận bất cứ props nào và trả về một react element.

8. Khởi tạo một React Component

Ở đây mình đề cập 2 cách để khởi tạo một **components**, mỗi cách có ưu và nhược điểm riêng và bạn có thể lựa chọn theo từng trường hợp sử dụng. Trước khi thực hiện viết components, chúng ta nên khởi tạo một thư mục có tên **components** trong thư mục **src** để chứa tất cả các *component* trong dự án. Cấu trúc thư mục của dự án lúc này sẽ là:

```
public/

node_modules/

src/

----components/

-----Components sẽ viết ở trong thư mục này

----App.js

----index.js

---- vv....

packages.json

packages-lock.json
```

xiv. Functional Component

Đây là cách viết phổ biến và được sử dụng nhiều nhất trong các dự án mà mình viết, bởi tính nhanh gọn và không quá phức tạp như các viết *class component*. Tiến hành tạo một file có tên **Welcome.js** trong thư mục **src/components**:

```

1    //Import react vào trong dự án
2    import React from "react";
3
4    const Welcome = function(props) {
5        return (
6            <div>
7                <h1>Welcome ! I am a functional component </h1>
8            </div>
9        )
10    }
11
12    export default Welcome;

```

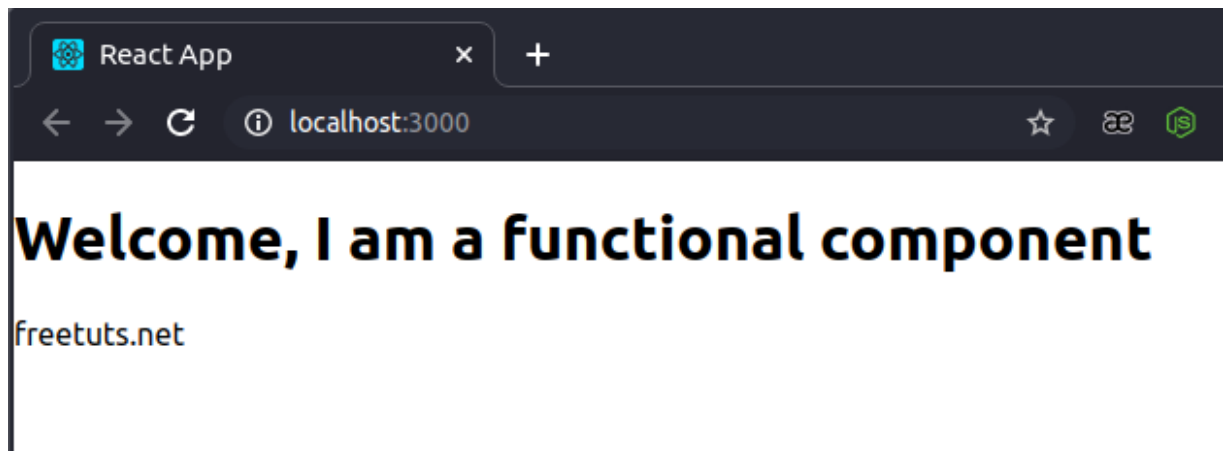
Bạn cũng có thể nhận các props được truyền vào component bằng cách nhận một tham số trong function. Chúng ta sẽ return các react element bằng cách sử dụng cú pháp `return()` như ví dụ bên trên, cuối file bạn cần phải `export` component này để file khác có thể lấy để sử dụng. Bây giờ chúng ta sẽ tiến hành import file `Welcome.js` vào trong file `App.js` và hiển thị nó ra :

```

1    import React from 'react';
2    import Welcome from '../components/Welcome'
3    function App() {
4        return (
5            <div>
6                <Welcome />
7                <p>freetuts.net</p>
8            </div>
9        );
10    }
11
12    export default App;

```


Tiến hành khởi chạy dự án chúng ta sẽ có kết quả như hình :



Vậy khi nào bạn cần sử dụng **functional component**? Mình hay sử dụng functional component nhất bởi nó rất gọn nhẹ và không quá nhiều dòng code dài dòng.

Có một vấn đề là trong functional component thì sẽ không có các khái niệm như state, life cycles, events,...nhưng trong phiên bản gần đây thì React cũng đã hỗ trợ thêm React Hooks cho phép bạn làm việc với state, lifecycles dễ dàng hơn trong functional components.

Bởi vậy, bây giờ hầu như **functional component** có hầu hết chức năng giống như **class component** nên bạn có thể cân nhắc sử dụng nó nhiều hơn.

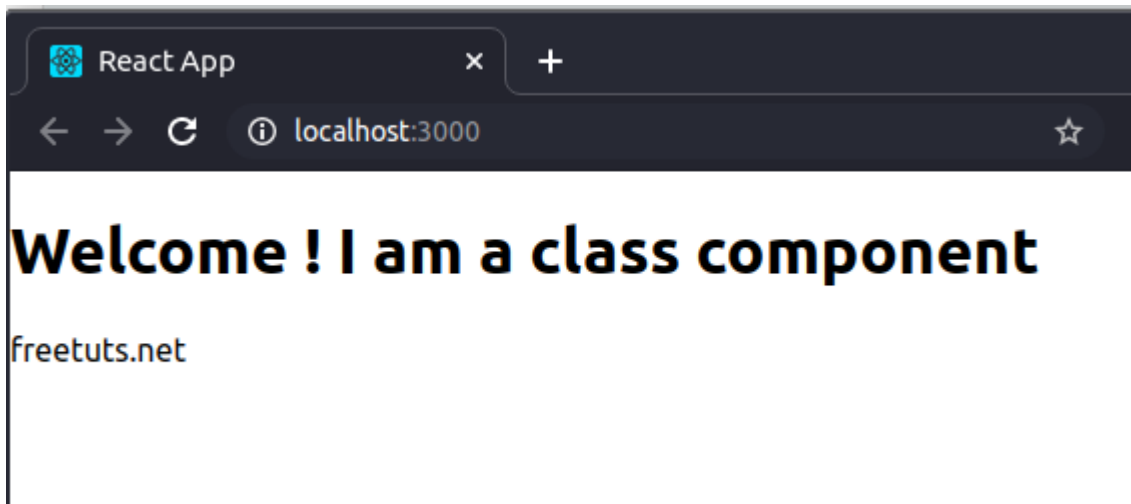
xv. Class Component

Cách viết này là cách viết đầy đủ của một component, khi bạn viết một **class component** bạn sẽ sử dụng được hầu hết các chức năng của component như state, props, lifecycle,...Chúng ta tiếp tục với ví dụ trên, chúng ta chỉ sửa đổi file `src/Welcome.js` :

```
1   import React, { Component } from "react";
2   class Welcome extends Component {
3     render() {
4       return (
5         <div>
6           <h1>Welcome ! I am a class component </h1>
7         </div>
```

```
8         );  
9     }  
10 }  
11 export default Welcome;
```

Khởi chạy mà bạn sẽ nhận được kết quả giống như ví dụ bên trên.



Khi bạn muốn làm việc với các chức năng của component như *events, state, lifce cycles* hay tổ chức các đoạn code theo cấu trúc theo mô hình OOP thì bạn có thể cân nhắc sử dụng **class components**. Ngược lại, bạn có thể sử dụng functional component để thay thế.

5. Tìm hiểu về Props trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về khái niệm props trong ReactJS, và cách để sử dụng nó trong một dự án. Trong quá trình lập trình một ứng dụng React, việc giao tiếp giữa các components với nhau là điều không thể thiếu. React cho phép chúng ta làm điều này bằng cách sử dụng props.

Table of Content

- [Props là gì ?](#)
- [Props trong React](#)
 - [Truyền props trong các components](#)
 - [Nhận props trong components](#)
- [Ví dụ thực tế](#)

9. Props là gì ?

Trước tiên, bạn cần tìm hiểu về [khái niệm components trong ReactJS](#) bởi props và state được coi là 2 phần khá quan trọng trong một components.

Props là một object được truyền vào trong một components, mỗi components sẽ nhận vào props và trả về `react` element. Props cho phép chúng ta giao tiếp giữa các components với nhau bằng cách truyền tham số qua lại giữa các components.

Khi một components cha truyền cho component con một props thì components con chỉ có thể đọc và không có quyền chỉnh sửa nó bên phía components cha.

Cách truyền một props cũng giống như cách mà bạn thêm một attributes cho một element HTML. Ở đây mình có một ví dụ:

```
1 const App = () => <Welcome name="Freetuts"></Welcome>
```

Trong ví dụ bên trên, component có tên `Welcome` sẽ nhận được giá trị của props có tên `name` vừa mới được truyền vào.

10. Props trong React

Tiếp theo, chúng ta sẽ đi tìm hiểu về cách làm việc với props trong React bao gồm các phần như truyền một props, đọc props,...

xvi. Truyền props trong các components

Bạn có thể truyền dữ liệu từ một component với nhau bằng cách truyền như một attributes trong HTML element như sau:

```
1    const App = () => <Welcome tenProps1="giatri" tenProps2 = "giatri2">Giá trị của props</Welcome>
```

Giả sử mình muốn truyền cho components có tên `Welcome` các giá trị như:

```
1    const App = () => <Welcome name="Nguyễn Trí" age=18 gender=1>Xin chào Freetuts.net</Welcome>
```

Vậy trong components `Welcome` giá trị của props sẽ là một object bao gồm các giá trị truyền vào :

```
1    {
2      name: "Nguyễn Trí",
3      age: 18,
4      gender : 1,
5      children: "Xin chào Freetuts.net"
6    }
```

Khi bạn truyền một giá trị bên trong một tags thì nó sẽ là giá trị của thuộc tính `children` trong object props như bên trên ví dụ cụ thể mình sẽ đề cập ở cuối bài để hiểu rõ hơn.

xvii. Nhận props trong components

Chúng ta có thể nhận giá trị của một **props** bằng cách nhận vào tham số trong functional components và `this.props` trong một class components. Ở bên dưới mình có ví dụ:

```
1    //Nhận giá trị của props trong class component bằng this.props
2    import React, { Component } from "react";
3    class Welcome extends Component {
4      render() {
5        console.log(this.props) //Giá trị của props
6        return (
```

```

7          <div>
8              <h1>Xin chào {this.props.name} !</h1>
9          </div>
10      );
11  }
12  }
13  export default Welcome;

```

```

1  //Nhận props trong functional components bằng cách
2  //chỉ định tham số trong function.
3  import React from "react";
4  const Welcome = (props) => {
5      console.log(props) //Giá trị của props
6      return (
7          <div>
8              <h1>Xin chào {props.name} !</h1>
9          </div>
10      );
11  };
12  export default Welcome;

```

11. Ví dụ thực tế

Giả sử mình muốn truyền các props có tên *name*, *type*, *color*, *size*,... vào trong components có tên **Clothes**. Chúng ta sẽ thực hiện các bước lần lượt như sau :

Trong thư mục **src** của dự án chúng ta sẽ tiến hành tạo một file có tên **Clothes.js**:

```

1  import React from "react";
2  const Clothes = (props) => {

```

```

3      console.log(props) //Giá trị của props
4      return (
5          <div>
6              <h1>{props.children}</h1>
7              <ul>
8                  <li><b>Tên:</b> {props.name}</li>
9                  <li><b>Loại:</b> {props.type}</li>
10                 <li><b>Màu:</b> {props.color}</li>
11                 <li><b>Kích cỡ:</b> {props.size}</li>
12             </ul>
13             <hr></hr>
14         </div>
15     );
16 };
17 export default Clothes;

```

Component này sẽ hiển thị các props được truyền vào bao gồm: *name, type, age, size,....*

Tiếp theo ở file **App.js**, chúng ta sẽ import component **Clothes** và truyền vào đó các props.

```

1      import React from "react";
2      import Clothes from "../Clothes"; //Import component vào
3      function App(props) {
4          return (
5              <div>
6                  <Clothes name="Quần jean" type="Skinny" color ="Đen" size = "L">Clothes 1</Clothes>
7                  <Clothes name="Váy" type="váy công chúa" color ="Trắng" size = "M">Clothes 2</Clothes>
8              </div>
9          );
10     }

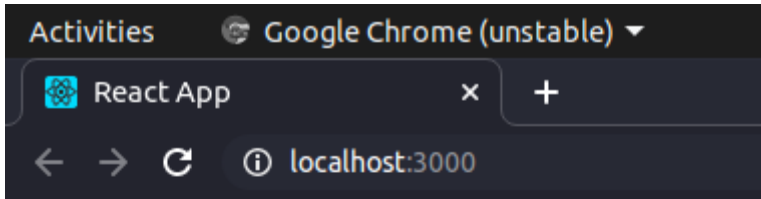
```

```
11 export default App;
```

Bên trên mình mình truyền vào các props cần thiết, và mình gọi component `Clothes` 2 lần với các props khác nhau. Chúng ta cùng chạy dự án để xem kết quả.

```
1 npm start
```

Truy cập đường dẫn <http://localhost:3000> chúng ta có thể thấy các props được truyền đi đã được hiển thị:



Clothes 1

- **Tên:** Quần jean
 - **Loại:** Skinny
 - **Màu:** Đen
 - **Kích cỡ:** L
-

Clothes 2

- **Tên:** Váy
 - **Loại:** váy công chúa
 - **Màu:** Trắng
 - **Kích cỡ:** M
-

6. Tìm hiểu State trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về khái niệm state trong ReactJS. ReactJS được xây dựng để làm các trang web *single page applicationn* (SPA) nên vẫn re-render lại các component là điều tất yếu và không thể thiếu. **State** cho phép chúng ta lưu trữ dữ liệu trong component, và mỗi khi **state** thay đổi thì component đó sẽ tự động re-render.

Table of Content

- State trong ReactJS là gì ?
- Thao tác với state trong ReactJS
 - Khởi tạo một state
 - Cập nhật một state
- Sự khác nhau giữa props và state

12. State trong ReactJS là gì ?

State là một object có thể được sử dụng để chứa dữ liệu hoặc thông tin về components. State có thể được thay đổi bất cứ khi nào mong muốn. Khác với props bạn có thể truyền props sang các components khác nhau thì state chỉ tồn tại trong phạm vi của components chứa nó, mỗi khi state thay đổi thì components đó sẽ được render lại.

Trong các dự án React, state được dùng để phản hồi các yêu cầu từ người dùng, hay lưu trữ một dữ liệu nào đó trong components.

13. Thao tác với state trong ReactJS

Chúng ta có thể thao tác với **state** trong một component rất dễ dàng bằng cách sử dụng class components. Bên dưới mình sẽ chỉ ra các thao tác với state trong một component.

xviii. Khởi tạo một state

Chúng ta có thể khởi tạo một state bằng cách gán giá trị cho biến `this.state`:

```
1    this.state = { name : 'freetuts.net' }
```

và lấy giá trị của state bằng cách gọi `this.state`:

```
1    console.log(this.state.website) //freetuts.net
```


Ví dụ bên dưới mình có một class components và mình sẽ tiến hành khởi tạo state bên trong phương thức **constructor** :

```
1    import React from "react";
2
3    class App extends React.Component {
4      constructor(props) {
5        super(props);
6        //Chỉ định một state
7        this.state = { website: "Freetuts.net" };
8      }
9      render() {
10       return (
11         <div>
12           <h1>Học ReactJS căn bản tại {this.state.website} </h1>
13         </div>
14       );
15     }
16   }
17   export default App;
```

Trong hầu hết các trường hợp bạn nên khởi tạo state bên trong hàm **constructor()** để tránh gặp các lỗi không mong muốn. Vì đây sẽ là hàm khởi chạy đầu tiên khi một components được gọi.

xix. Cập nhật một state

Trong các components bạn cần phải thao tác với state rất nhiều , ngoài thêm và lấy giá trị của state bạn còn phải cập nhật các states để ReactJS có thể tự động re-render lại components. Điều này khá quan trọng, giả sử bạn đang cho người dùng nhập vào một form nào đó và khi click **Lưu** thì nội dung được điền trong form lúc này sẽ phải hiển thị ra màn hình. Đây là lúc bạn cần dùng đến **cách thay đổi giá trị của một state**.

Để **cập nhật một state** bạn sử dụng phương thức:

```

1    this.setState({
2        name : 'newValue'
3    })

```

Ngoài ra bạn cũng có thể lấy giá trị của state trước khi cập nhật:

```

1    this.setState((state) => {
2        return newValue;
3    });

```

Bên dưới mình có một ví dụ về cập nhật state index khi nhấn vào click vào button tương ứng :

```

1    import React from "react";
2
3    class App extends React.Component {
4        constructor(props) {
5            super(props);
6            //Chỉ định một state
7            this.state = { index: 1 };
8        }
9        render() {
10            return (
11                <div>
12                    <p>Giá trị {this.state.index}</p>
13                    <button
14                        onClick={() => {
15                            this.setState({
16                                index: this.state.index + 1
17                            })
18                        }}
19                    >

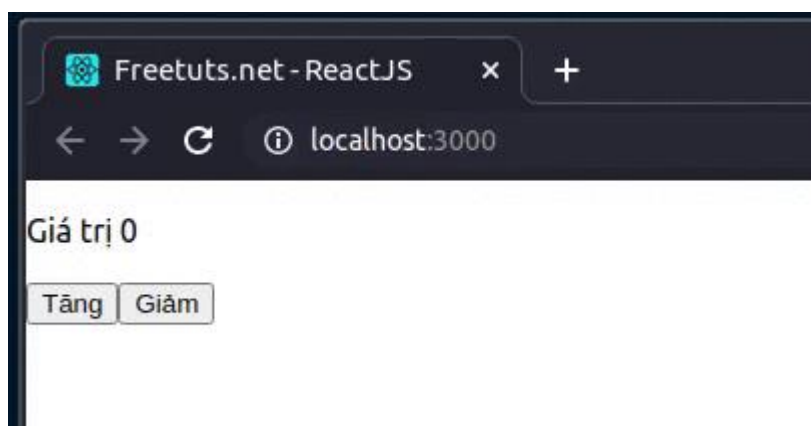
```

```

20         Tăng
21     </button>
22     <button
23         onClick={() => {
24             this.setState({
25                 index: this.state.index - 1
26             })
27         }}
28     >
29         Giảm
30     </button>
31 </div>
32 );
33 }
34 }
35 export default App;

```

Chúng ta sẽ thấy giá trị của state sẽ được thay đổi mỗi khi bạn click vào button Tăng hoặc Giảm :



State là một khái niệm khá đơn giản nhưng cũng hết sức quan trọng trong component, mặc dù props và state hay đi đôi với nhau nhưng nó hoàn toàn khác biệt với nhau.

14. Sự khác nhau giữa props và state

Trong quá trình học React, cũng có vài trường hợp bạn chưa hiểu về 2 khái niệm props và state kĩ nên có sự nhầm lẫn ở đây.

- **State** - Dữ liệu chỉ nằm trong phạm vi của một component. Nó được sở hữu bởi một components cụ thể mà chỉ là của component đó thôi. Ví dụ, như người yêu bạn chỉ là của bạn vậy =))). Và mỗi khi state thay đổi thì component cũng phải thay đổi theo.
- **Props** - Dữ liệu được truyền từ component cha cho component con, components con khi nhận được sẽ chỉ được đọc mà không thể thay đổi dữ liệu đó.

Sự khác nhau chính của 2 khái niệm này là component sở hữu dữ liệu. State là chỉ riêng nó có thể sử dụng. Props là dữ liệu mà component con được nhận về từ một component cha.

Vì phạm vi của state chỉ nằm trong components nên việc truyền dữ liệu từ các components với nhau người ta thường dùng props. Nhưng vấn đề ở đây là props chỉ có thể truyền cho component con của nó và khi truyền cho các component cháu, chất khá rắc rối. Bởi vậy chúng ta có thêm khái niệm về Redux, mình sẽ giới thiệu ở phần tiếp theo vì đây là phần khá phức tạp và đau não :))

7. Tìm hiểu Props Validation trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về props validation. Trước khi đi qua tìm hiểu về phần này chúng ta cần xem bài viết trước về [props trong ReactJS](#) để hiểu về nó đã nhé.

Table of Content

- [Props validation là gì ?](#)
- [Validating props](#)
 - [Thuộc tính propTypes](#)
 - [Thư viện PropTypes](#)
 - [Ví dụ thực tế](#)

15. Props validation là gì ?

Props là một phần rất quan trọng được truyền vào React components, các props này thường phải có một kiểu dữ liệu nhất định. Nếu props được truyền đến component có một kiểu mà không mong muốn, component đó sẽ rất khó để kiểm soát và từ đó gây ra bugs.

Props validation là cách để kiểm soát các vấn đề về props, nó cho phép bạn kiểm tra dữ liệu đầu vào của props trước khi components được render. **Props validation** giúp chúng ta giảm thiểu các lỗi không mong muốn trong quá trình xây dựng ứng dụng React.

Giả sử mình đang xây dựng một component cho phép chúng ta cộng 2 số với nhau.

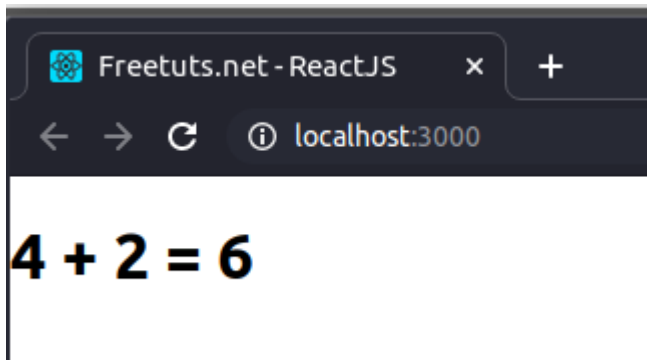
Trường hợp 1: Chúng ta sẽ truyền vào đó 2 giá trị là kiểu int, và bạn sẽ nhận được kết quả đúng:

```
1   import React from "react";
2
3   class App extends React.Component {
4     render() {
5       const {number1, number2} = this.props
6       return (
7         <div>
8           <h1>{number1} + {number2} = {number1 + number2}</h1>
```

```

9         </div>
10     );
11 }
12 }
13 //Chỉ định props mặc định.
14 //Ở đây mình cho props number1, number2 là kiểu int
15 App.defaultProps = {
16     number1: 4,
17     number2: 2
18 }
19 export default App;

```



Trường hợp 2: Nếu bạn để props là kiểu **string** thì component sẽ hiển thị sai kết quả:

```

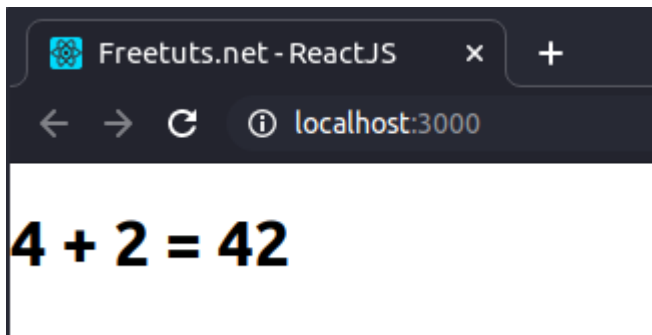
1     import React from "react";
2
3     class App extends React.Component {
4         render() {
5             const {number1, number2} = this.props
6             return (
7                 <div>
8                     <h1>{number1} + {number2} = {number1 + number2}</h1>
9                 </div>

```

```

10     );
11   }
12 }
13 // Chỉ định props mặc định.
14 // Ở đây mình chỉ định props number1, number 2 là kiểu string.
15 App.defaultProps = {
16   number1: '4',
17   number2: '2'
18 }
19 export default App;

```



Đây là lỗi mà bạn không mong muốn, khi mà kiểu dữ liệu của props được truyền sai. Trước khi chạy component bạn cần phải kiểm tra kiểu dữ liệu trước, điều này rất quan trọng trong các dự án lớn. Bởi vậy chúng ta sẽ đi tìm hiểu về **Props validation**.

16. Validating props

Trong phần trước chúng ta đã biết tại sao cần phải kiểm tra props trước khi khởi chạy, đây là bước quan trọng để chúng ta có thể kiểm tra kiểu và các giá trị của props trong React app. Trong phần này chúng ta sẽ đi tìm hiểu về cách để **kiểm tra props trong một components**.

xx. Thuộc tính propTypes

React cung cấp cho chúng ta một phương thức để kiểm tra props trong mỗi component. Trong tất cả các React component đều có một thuộc tính có tên `propTypes`:

```

1  /**

```

```

2      * FUNCTIONAL COMPONENTS
3      */
4      function ReactComponent(props) {
5          // ...render ()...
6      }
7
8      ReactComponent.propTypes = {
9          // ...chúng ta sẽ viết props type ở đây
10     }
11
12
13     /**
14      * CLASS COMPONENTS: METHOD 1
15      */
16     class ReactComponent extends React.Component {
17         // ...render()...
18     }
19
20     ReactComponent.propTypes = {
21         // ...chúng ta sẽ viết props type ở đây
22     }
23
24
25     /**
26      * CLASS COMPONENTS: METHOD 2
27      * Sử dụng class statics...
28      */
29     class ReactComponent extends React.Component {

```

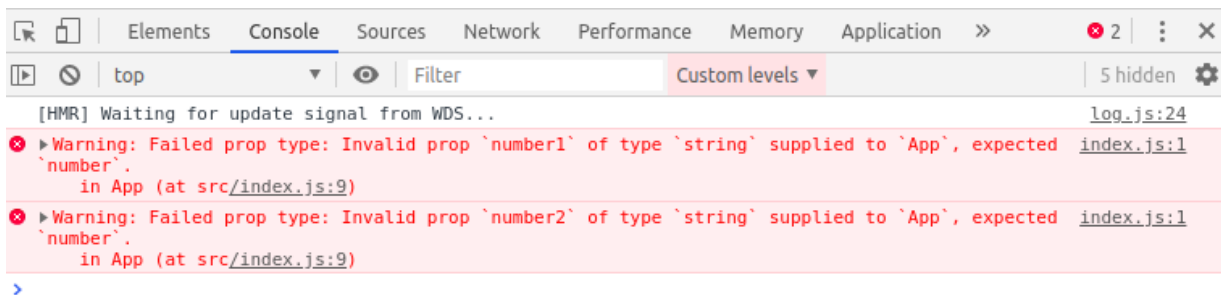


```

30      // ...component class body here
31
32      static propTypes = {
33          // ...chúng ta sẽ viết props type ở đây
34      }
35  }

```

Khi xuất hiện lỗi, chúng ta có thể thấy được lỗi được hiển thị trong console, nó chỉ hiển thị trong môi trường phát triển, khi deploy dự án lên thì lỗi này sẽ không được hiển thị.



xxi. Thư viện PropTypes

Trong các phiên bản cũ, người ta thường dùng `React.PropTypes` làm cách giá trị truyền vào trong `propTypes`, ví dụ như :

```

1  App.propTypes = {
2      number1: React.PropTypes.number,
3      number2: React.PropTypes.number
4  }

```

Kể từ phiên bản React v15.5 trở đi `React.PropTypes` đã bị xóa bỏ và thay vào đó là thư viện **PropTypes**, cho phép chúng ta chỉ định kiểu dữ liệu mà bạn mong muốn cho props. Vì bị tách ra thành thư viện nên bạn cần phải cài thêm vào React bằng `npm`:

```

1  npm i prop-types

```

Để sử dụng bạn chỉ cần import nó vào:

```

1  import PropTypes from 'prop-types';

```

Đây là hầu hết các thuộc tính `PropTypes` có trong thư viện:

```
1      import PropTypes from "prop-types";
2
3      MyComponent.propTypes = {
4          // Kiểu dữ liệu
5          optionalArray: PropTypes.array,
6          optionalBool: PropTypes.bool,
7          optionalFunc: PropTypes.func,
8          optionalNumber: PropTypes.number,
9          optionalObject: PropTypes.object,
10         optionalString: PropTypes.string,
11         optionalSymbol: PropTypes.symbol,
12
13         // Bất cứ thứ gì có thể render như : numbers, strings, elements hoặc array
14         optionalNode: PropTypes.node,
15
16         // React Element
17         optionalElement: PropTypes.element,
18
19         // React Element Type
20         optionalElementType: PropTypes.elementType,
21
22         // Instance
23         optionalMessage: PropTypes.instanceOf(Message),
24
25         // Giá trị của props bao gồm
26         optionalEnum: PropTypes.oneOf(["News", "Photos"]),
```

```

27 // Một object có thể bao gồm nhiều kiểu
28 optionalUnion: PropTypes.oneOfType([
29     PropTypes.string,
30     PropTypes.number,
31     PropTypes.instanceOf(Message)
32 ]),
33
34 // Một mảng chứa giá trị là các kiểu
35 optionalArrayOf: PropTypes.arrayOf(PropTypes.number),
36
37 // Một object có thuộc tính là kiểu
38 optionalObjectOf: PropTypes.objectOf(PropTypes.number),
39
40 // Một object mà mỗi phần có kiểu dữ liệu riêng
41 optionalObjectWithShape: PropTypes.shape({
42     color: PropTypes.string,
43     fontSize: PropTypes.number
44 }),
45
46 // Một object có cảnh báo về các thuộc tính bổ sung
47 optionalObjectWithStrictShape: PropTypes.exact({
48     name: PropTypes.string,
49     quantity: PropTypes.number
50 }),
51
52 // Bạn có thể xâu chuỗi bất kỳ câu hỏi nào ở trên với `isRequired` để đảm bảo cả
53 // được hiển thị nếu prop không được cung cấp.
54 requiredFunc: PropTypes.func.isRequired,

```

```

55
56 // Bất kì kiểu dữ liệu nào nhưng nó phải tồn tại..
57 requiredAny: PropTypes.any.isRequired,
58
59 //Chỉ định props validation tùy chỉnh
60 customProp: function(props, propName, componentName) {
61     if (!/matchme/.test(props[propName])) {
62         return new Error(
63             "Invalid prop `" +
64                 propName +
65                 "` supplied to" +
66                 " `" +
67                 componentName +
68                 "` . Validation failed."
69         );
70     }
71 },
72
73 // Bạn cũng có thể cung cấp trình xác nhận tùy chỉnh cho `ArrayOf` và ` objectOf`
74 // Nó sẽ trả về một đối tượng Error nếu xác nhận thất bại. Trình xác nhận
75 // sẽ được gọi cho mỗi khóa trong mảng hoặc đối tượng. Hai cái đầu tiên
76 // đối số của trình xác nhận là chính mảng hoặc đối tượng và
77 // khóa hiện tại của mục.
78 customArrayProp: PropTypes.arrayOf(function(
79     propValue,
80     key,
81     componentName,
82     location,

```

```

83     propFullName
84   ) {
85     if (!/matchme/.test(propValue[key])) {
86       return new Error(
87         "Invalid prop `" +
88           propFullName +
89           "` supplied to" +
90           " `" +
91           componentName +
92           "` . Validation failed."
93       );
94     }
95   })
96 };
97

```

Phần dưới mình sẽ đi vào ví dụ cụ thể để hình dung rõ hơn chứ ngồi học lý thuyết khá buồn ngủ :>

xxii. Ví dụ thực tế

Ví dụ 1: Tiếp tục với ví dụ ở trên đầu bài viết là cộng 2 số lại với nhau chúng ta sẽ thực hiện kiểm tra bằng cách sử dụng:

```

1   import React from "react";
2   import PropTypes from 'prop-types';
3
4   class App extends React.Component {
5     render() {
6       const {number1, number2} = this.props
7       return (
8         <div>

```

```

9         <h1>{number1} + {number2} = {number1 + number2}</h1>
10     </div>
11 );
12 }
13 }
14 App.defaultProps = {
15     number1: 4,
16     number2: 2
17 }
18 //Kiểm tra dữ liệu sử dụng PropTypes.
19 App.propTypes = {
20     number1: PropTypes.number,
21     number2: PropTypes.number
22 }
23 export default App;

```

Các bạn lưu ý phải cài đặt thư viện **PropTypes** trước nhé, lúc này dữ liệu đã được kiểm tra và sẽ hiển thị lỗi ra console như bên trên mình đã đề cập.

Ví dụ 2: Trong ví dụ này mình sẽ hiển thị ra thông tin của những chiếc điện thoại của Apple bao gồm name, type, public_year, storage,...

```

1     import React from "react";
2     import PropTypes from 'prop-types';
3
4     class App extends React.Component {
5         render() {
6
7             return (
8                 <div>
9                     <h1>{this.props.name}</h1>

```

```

10         <ul>
11             <li>{this.props.type}</li>
12             <li>{this.props.public_year}</li>
13             <li>{this.props.storage}</li>
14         </ul>
15     </div>
16     );
17 }
18 }
19 // Đúng kiểu dữ liệu
20 App.defaultProps = {
21     name: 'iPhone Xs Max',
22     type: 'iPhone',
23     public_year: 2018,
24     storage: '64 GB'
25 }
26
27 // Đúng kiểu dữ liệu
28 App.defaultProps = {
29     name: 'iPad Mini 2019',
30     type: 'iPad',
31     public_year: 2019,
32     storage: 64
33 }
34
35
36
37 // Sai kiểu dữ liệu vì type phải là các giá

```

```

38 // trị như iPhone, Ipad, Mac, SmartWatch
39 App.defaultProps = {
40   name: 'Airpods 2',
41   type: 'Airpod',
42   public_year: 2019
43 }
44
45 //Chỉ định validation props
46 App.propTypes = {
47   name: PropTypes.string,
48   type: PropTypes.oneOf(["iPhone", "iPad", "Mac", "SmartWatch"]),
49   public_year: PropTypes.oneOfType([
50     PropTypes.string,
51     PropTypes.number,
52   ])
53 }
54 export default App;

```


8. Tìm hiểu về Component API trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về các API bên trong các React Component, đây là những API được sử dụng khá nhiều trong quá trình làm việc với các component bao gồm `setState()`, `forceUpdate()`, `ReactDOM.findDOMNode()`,...

Table of Content

- Set State API
- Force Update
- Find Dom Node
- Bind function

17. Set State API

Hàm `setSate()` này cho phép chúng ta cập nhật giá trị của state, tham số truyền vào của API này sẽ là giá trị của state bạn muốn update:

```
1    this.setSate({
2      website: 'freetuts.net'
3    })
```

hoặc là một function callback bao gồm các tham số lần lượt là state trước đó, và props có trong component.

```
1    this.setState((prevState, props) => {
2      return newState;
3    })
```

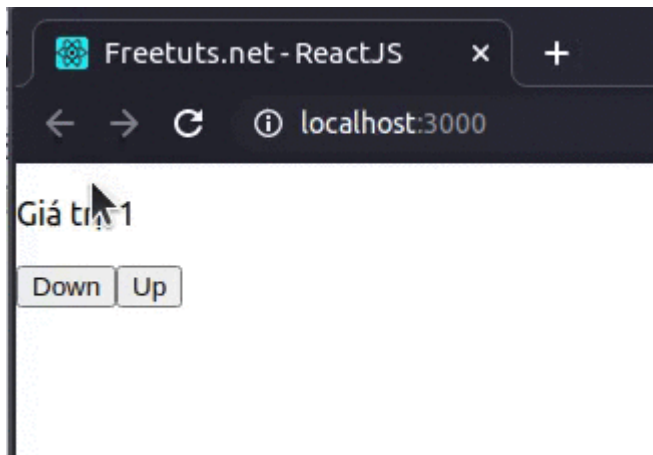
Trong ví dụ bên dưới mình sẽ tiến hành update state bằng 2 cách trên.

```
1    import React from "react";
2
3    class App extends React.Component {
4      constructor(props) {
```

```

5      super(props);
6      this.state = {
7          index: 1
8      };
9  }
10     countdown() {
11         this.setState({
12             index: this.state.index - 1
13         });
14     }
15     countUp(){
16         this.setState((prevState, props) => {
17             return {
18                 index: prevState.index + 1
19             }
20         });
21     }
22     render() {
23         return (
24             <div>
25                 <p>Giá trị: {this.state.index}</p>
26                 <button onClick={() => this.countDown()}>Down</button>
27                 <button onClick={() => this.countUp()}>Up</button>
28             </div>
29         );
30     }
31 }
32

```



18. Force Update

Component chỉ thực hiện re-render khi các state thay đổi, trong trường hợp cần phải re-render mà không cần bất cứ sự thay đổi nào của state thì chúng ta chỉ cần gọi hàm `forceUpdate()`

```
this.forceUpdate()
```

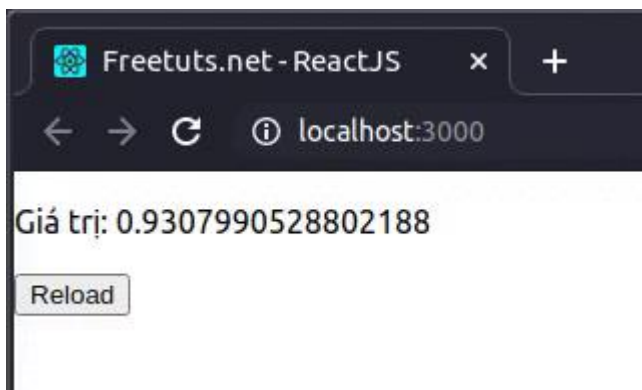
Bên dưới mình có ví dụ về hàm `forceUpdate()`, với mỗi lần component được re-render sẽ random ra một số khác nhau:

```
1    import React from "react";
2
3    class App extends React.Component {
4      constructor(props) {
5        super(props);
6        this.state = {
7          index: 1
8        };
9      }
10     render() {
11       return (
```

```

12         <div>
13             <p>Giá trị: {Math.random()}</p>
14             <button onClick={() => this.forceUpdate()}>Reload</button>
15         </div>
16     );
17 }
18 }
19
20 export default App;

```



19. Find Dom Node

Hàm này cho phép chúng ta làm việc với DOM trong React, nó không được khuyến khích bởi sẽ làm ảnh hưởng đến các Virtual DOM gây ra lỗi không mong muốn. Bạn cần phải gọi đối tượng **ReactDOM** trong thư viện **react-dom** bằng cách:

```
1 import ReactDOM from 'react-dom';
```

Để sử dụng hàm **findDOMNode()** chúng ta có cú pháp như sau:

```
1 ReactDOM.findDOMNode(component)
```

Giả sử ví dụ bên dưới mình muốn sửa đổi màu chữ của thẻ **h1** có id là **title** thành màu đỏ.

```

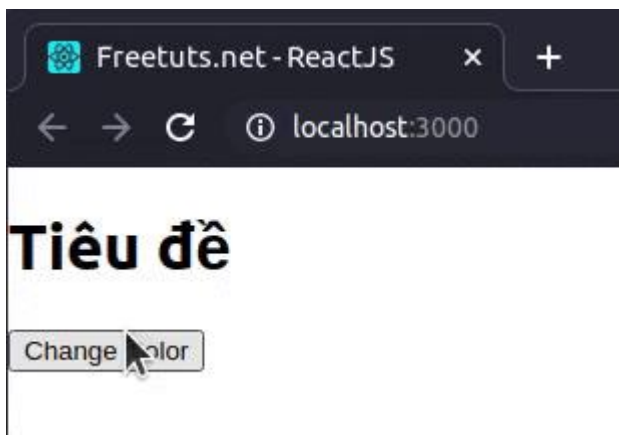
1 import React from "react";
2 import ReactDOM from "react-dom";

```

```

3
4   class App extends React.Component {
5       changeColor() {
6           var title = document.getElementById("title");
7           ReactDOM.findDOMNode(title).style.color = "red";
8       }
9       render() {
10          return (
11              <div>
12                  <h1 id="title">Tiêu đề</h1>
13                  <button onClick={() => this.changeColor()}>Change Color</button>
14              </div>
15          );
16      }
17  }
18
19  export default App;

```



20. Bind function

Đây là phần mà rất nhiều bạn gặp lỗi liên quan đến cú pháp ES6 khi thực hiện gọi hàm trong class component, giả sử mình muốn khi người dùng click vào nút **Change** sẽ gọi hàm `changeTitle()`.

Khi bạn gọi hàm bằng cách này sẽ gây ra lỗi không tồn tại biến `this` trong hàm `changeTitle()`.

```
1    class App extends React.Component {
2      constructor() {
3        super();
4        this.state = {
5          title: "Freetuts.net"
6        };
7      }
8      changeTitle() {
9        //Sẽ không nhận được giá trị của this.
10       //Từ đó không thể set state được.
11       this.setState({
12         title: "Freetuts.net New"
13       });
14     }
15
16     render() {
17       //Khi bạn gọi hàm như này sẽ gây ra lỗi.
18       //Vì không chỉ định giá trị của biến this cho hàm changeTitle.
19       return (
20         <div>
21           <h1 id="title">{this.state.title}</h1>
22           <button onClick={this.changeColor}>Change Color</button>
23         </div>
24       );
25     }
26   }
```

Chúng ta cần phải chỉ định biến `this` cho hàm vừa gọi bằng cách dùng cú pháp `bind` trong ES6:

```
1    <button onClick={this.changeColor.bind(this)}>Change Color</button>
```

hoặc viết như sau :

```
1    <button onClick={() => this.changeColor()}>Change Color</button>
```

9. Tìm hiểu Component Life Cycle trong ReactJS

Đăng bởi: Jickme- Vào ngày: 03-04-2020- View: 1746

Trong bài viết này chúng ta sẽ cùng nhau đi **tìm hiểu về component life cycle trong React**, đây là một phần quan trọng trong React, nó cho phép bạn hiểu được cách mà một component hoạt động ra sao ?

Table of Content

- [Component Life Cycle là gì ?](#)
- [Component Life Cycle](#)
 - [Initialization](#)
 - [Mounting](#)
 - [Updating](#)
 - [Unmounting](#)

21. Component Life Cycle là gì ?

Chúng ta có thể thấy được mọi thứ trong thế giới đều hoạt động theo một chu kì (ví dụ như con người và cây cối). Cây được mọc lên, sẽ phát triển rồi đến một khoảng thời gian nào đó là sẽ chết đi. Trong React Component cũng vậy, một chu kì cũng xuất hiện, components được khởi tạo (hiển thị ra DOM), update, và kết thúc (unmount),...đó được gọi là một component life cycle.

React cho phép chúng ta tham gia vào các giai đoạn của mỗi component bằng cách sử dụng các phương thức được xây dựng sẵn trong mỗi giai đoạn đó. Khi một components được khởi chạy nó sẽ phải trải qua 4 giai đoạn chính:

- initialization
- mounting
- updating
- unmounting

Phần tiếp theo chúng ta sẽ đi vào cách mà bạn có thể tham gia vào giai đoạn trong một components.

22. Component Life Cycle

Chúng ta sẽ tìm hiểu về các lifecycle methods có trong mỗi giai đoạn.

xxiii. Initialization

Đây là giai đoạn mà thành phần sẽ bắt đầu hành trình của mình bằng cách *khởi tạo state và props*. Điều này thường được thực hiện bên trong phương thức *constructor*. Ở đây mình có ví dụ:

```
1  class App extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        website: 'Học ReactJS cùng Freetuts.net'
6      };
7    }
8  }
```

Ở giai đoạn này, React Component sẽ tiến hành khởi tạo các state, props hay các câu lệnh được khởi tạo trong *constructor()*,...

xxiv. Mounting

Giai đoạn này được thực hiện sau khi quá trình initialization(khởi tạo) được hoàn thành. Nó thực hiện nhiệm vụ chuyển **virtual DOM (DOM ảo)** trong React thành **DOM** và hiển thị trên trình duyệt. Component sẽ được render lần đầu tiên, ở đây chúng ta có 3 phương thức để tham gia vào giai đoạn này.

componentWillMount()

Được khởi chạy khi một component chuẩn bị được mount (tức là trước khi thực hiện render), sau khi thực hiện xong *componentWillMount()* thì component mới có thể được mount.

Lưu ý: Chúng ta **không** nên thực hiện bất cứ thay đổi nào liên quan đến state, props hay call API ở trong hàm này, bởi thời gian chuẩn bị mount -> mount rất ngắn nên các tác vụ đó không thể hoàn thành kịp. Khiến cho component render ra kết quả không như bạn mong muốn.

componentDidMount()

Được gọi khi component đã được mount (render thành công), quá trình này xảy ra sau khi *componentWillMount()* thực hiện xong. Trong phương thức này bạn có thể gọi API, thay đổi *state*, *props*.

Ở đây mình có ví dụ về 2 phương thức mà mình vừa đề cập :

```

1    class Lifecycle extends React.Component {
2      componentWillMount() {
3        console.log('Component will mount!')
4      }
5      componentDidMount() {
6        console.log('Component did mount!')
7        this.getList();
8      }
9      getList=()=>{
10       /** Gõ API, update state,vv...***/
11     }
12     render() {
13       return (
14         <div>
15           <h3>Mounting Method</h3>
16         </div>
17       );
18     }
19   }

```

xxv. Updating

Đây là giai đoạn thứ ba mà các component phải thực hiện, sau giai đoạn *initialization (khởi tạo)* , *mount (render lần đầu)*,... . Trong giai đoạn này, dữ liệu của các phần (props & state) sẽ được cập nhật để đáp ứng với các sự kiện của người dùng như click, gõ, v.v. Điều này dẫn đến việc re-render lại component, ở trong giai đoạn này chúng ta sẽ có 4 phương thức chính:

shouldComponentUpdate()

Phương thức này xác định xem component có nên được render lại hay không ? Theo mặc định, nó trả về *true*. Nhưng bạn có thể thay đổi giá trị trả về của nó theo từng trường hợp.

Nó sẽ nhận về 2 tham số truyền vào là `nextState` và `nextProps`.

componentWillUpdate()

Phương thức này được gọi trước khi tiến hành re-render, bạn có thể thực hiện các hành động như update state, props,...trong phương thức này trước khi tiến hành re-render. Giống như `shouldComponentUpdate()`, `componentWillUpdate()` sẽ nhận vào 2 tham số đó là `nextState` và `nextProps`

ComponentDidUpdate()

Phương thức này được gọi khi component đã re-render xong. Chúng ta có ví dụ về cả 3 phương thức về đề cập ở trên.

```
1    class Lifecycle extends React.Component {
2      constructor(props)
3      {
4        super(props);
5        this.state = {
6          date : new Date(),
7          clickedStatus: false,
8          list:[]
9        };
10     }
11     componentWillMount() {
12       console.log('Component will mount!')
13     }
14     componentDidMount() {
15       console.log('Component did mount!')
16       this.getList();
17     }
18     getList=()=>{
```

```

19      /*** method to make api call***/
20      fetch('https://api.mydomain.com')
21          .then(response => response.json())
22          .then(data => this.setState({ list:data }));
23      }
24      shouldComponentUpdate(nextProps, nextState){
25          return this.state.list!==nextState.list
26      }
27      componentWillUpdate(nextProps, nextState) {
28          console.log('Component will update!');
29      }
30      componentDidUpdate(prevProps, prevState) {
31          console.log('Component did update!')
32      }
33      render() {
34          return (
35              <div>
36                  <h3>Mounting Lifecycle Methods</h3>
37              </div>
38          );
39      }
40  }

```

xxvi. Unmounting

Đây là bước cuối cùng trong mỗi component, khi tất cả các tác vụ hoàn thành và bạn tiến hành unmount DOM. Quá trình này chỉ có duy nhất 1 phương thức đó là `componentWillUnmount()` :

```

1      class App extends React.Component {
2          constructor(props) {

```

```
3      super(props);
4      this.state = {
5        website: 'Học ReactJS'
6      };
7    }
8    componentWillMount() {
9      console.log('component will unmount')
10    }
11    render() {
12      return (
13        <div>
14          <p>Component LifeCycle</p>
15        </div>
16      );
17    }
18  }
```

Đây là bước cuối cùng và sẽ kết thúc một vòng đời của components.

10. Handling Events (xử lý sự kiện) trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về **Handling Events trong Reactjs** - xử lý sự kiện trong ReactJS. Trong một website việc tương tác giữa người dùng là điều không thể thiếu như click, nhập form,...chúng ta có thể thực hiện bắt các sự kiện này trong React một cách dễ dàng.

Table of Content

- [1. Handling Events](#)
- [2. Lưu ý với this trong xử lý Events](#)
 - [Xây dựng ví dụ](#)

23. Handling Events

Xử lý các sự kiện trong React rất giống với xử lý các sự kiện trên các phần tử DOM. Có một số khác biệt về cú pháp:

- Các sự kiện React được đặt tên bằng **camelCase**, thay vì chữ thường. Ví dụ: `onclick` -> `onClick`, `onchange` -> `onChange`
- Với JSX, bạn truyền một hàm để bắt sự kiện, thay vì một chuỗi như HTML thông thường.

Ở đây mình có ví dụ với HTML

```
1 <button onclick="changeName()">
2   Change Name
3 </button>
```

khi làm việc với JSX trong React chúng ta sẽ phải viết như sau :

```
1 <button onClick={changeName}>
2   Change Name
3 </button>
```

Một điểm khác biệt nữa là bạn không thể sử dụng **return false** để chặn các hành động mặc định được (prevent default), trong React bạn cần phải sử dụng **preventDefault()**. Giả sử trong HTML mình muốn dừng hành động mặc định của một form:

```

1    <form onSubmit="console.log('form submit'); return false;">
2      <div className="form-group">
3        <label htmlFor="text">Email:</label>
4        <input type="text" className="form-control" name="email" placeholder="Enter
5          this.changeInputValue(e) }
6        />
7      </div>
8      <button type="submit" className="btn btn-primary">
9        Submit
10     </button>
11   </div>
12 </form>

```

trong React, chúng ta cần phải sử dụng `e.preventDefault()`:

```

1    function ActionLink() {
2      function submitForm(e) {
3        e.preventDefault();
4        console.log('form submit !!');
5      }
6
7      return (
8        <form onSubmit = {(e) => submitForm(e)}>
9          <button type="submit">Submit</button>
10        </form>
11      );
12    }

```

Ở đây, `e` là một object chứa tất cả event. React định nghĩa object này theo [W3C spec](#), vì vậy chúng ta không cần phải quan tâm về khả năng tương thích giữa các trình duyệt với nhau.

24. Lưu ý với this trong xử lý Events

Ở đây mình có một ví dụ, khi click vào button tương ứng sẽ thực hiện ẩn/hiển nội dung:

```
1    import React from "react";
2    class App extends React.Component {
3      constructor(props) {
4        super(props);
5        this.state = {
6          isShow: true
7        };
8      }
9      toggleMSG() {
10       this.setState({
11         isShow: !this.state.isShow
12       });
13     }
14     render() {
15       return (
16         <div>
17           <b>Nội dung : {this.state.isShow ? "Freetuts.net - Lập trình ReactJS" : ""}</b><br />
18           <button onClick={() => this.toggleMSG()}>
19             {this.state.isShow ? "HIDE" : "SHOW"}
20           </button>
21         </div>
22       );
23     }
24   }
25   export default App;
```


Trong sự kiện `onClick` chúng ta thực hiện gọi hàm `toggleMSG()`, hãy cẩn thận trong khi gọi hàm này, bạn cần phải truyền vào đó biến `this`, bằng các cách viết như sau:

```
1 //Sử dụng arrow function
2 <button onClick={() => this.toggleMSG()}>
3
4 //Sử dụng bind
5 <button onClick={this.toggleMSG.bind(this)}>
6
7 //hoặc bind this trong constructor()
```

xxvii. Xây dựng ví dụ

Trong phần này mình sẽ đi xây dựng ví dụ về bắt sự kiện trong React và một vài cách gọi hàm trong React. Trong file `src/App.js` chúng ta có ví dụ:

```
1 import React from "react";
2
3 class App extends React.Component {
4   constructor(props) {
5     super(props);
6     //Khởi tạo initial state
7     this.state = {
8       textareaChange: "",
9       buttonClick : "",
10      mouseOver: ""
11    };
12    //bind this cho function mouseOver để tránh gặp lỗi
13    //không tồn tại biến this.
14    this.mouseOver = this.mouseOver.bind(this)
15  }
```

```

16     changeText(e) {
17         this.setState({
18             textareaChange: e.target.value
19         });
20     }
21     mouseOver() {
22         this.setState({
23             mouseOver: this.state.mouseOver + "mouseOver..."
24         });
25     }
26     render() {
27         return (
28             <div style={{marginTop: "5%"}}>
29                 <button
30                     onClick={() => {
31                         this.setState({
32                             buttonClick: this.state.buttonClick + "onClick..."
33                         });
34                     }}
35                 >
36                     Click me..
37                 </button>{" "}
38
39                 <p>
40                     button: <code>{this.state.buttonClick}</code>
41                 </p>
42                 <hr />
43                 <textarea onChange={e => this.changeText(e)} placeholder="nhập cái gì đó.."

```

```

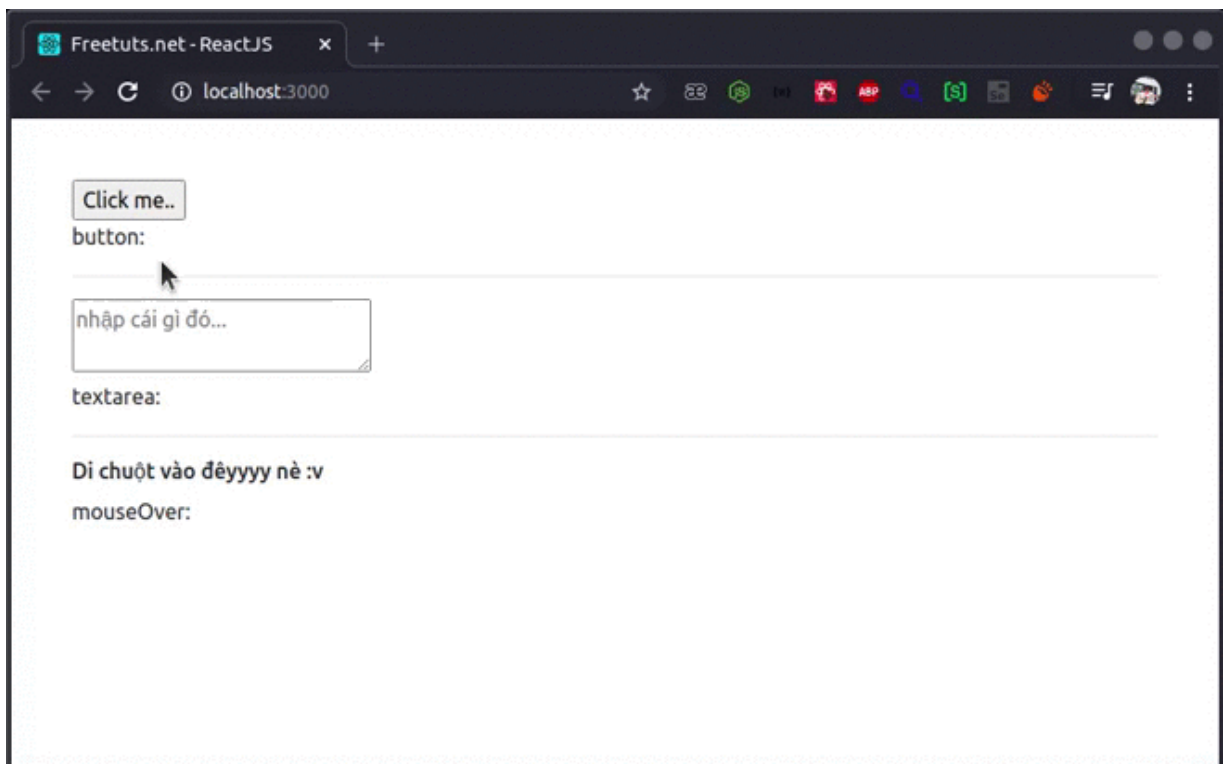
44         <p>
45             textarea: <code>{this.state.textareaChange}</code>
46         </p>
47
48         <hr />
49         <h6 onMouseOver={this.mouseOver}>Di chuột vào đâyyyyy nè :v</h6>
50         <p>
51             mouseOver: <code>{this.state.mouseOver}</code>
52         </p>
53     </div>
54     );
55 }
56 }
57
58 export default App;

```

Bên trên mình xây dựng 2 hàm `changeText()` và `mouseOver()`, khi sự kiện được gọi nó sẽ gọi hàm tương ứng. Trong sự kiện `onClick` mình thực hiện viết câu lệnh trực tiếp mà không cần gọi hàm. Chúng ta khởi chạy ví dụ bằng cách sử dụng dòng lệnh:

```
1 npm start
```

và đây là kết quả của chúng ta:



11. Xử lý Form trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về các **thao tác xử lý form trong ReactJS**.

Trong quá trình lập trình với React bạn cần phải làm việc với form nhất nhiều như đăng nhập, lấy thông tin người, vv.. Thao tác với form trong React rất đơn giản, hầu như chỉ là các kiến thức cơ bản ở phần trước như về *component, state, props*,...

Ngoài việc thao tác với form thông thường bạn còn có thể sử dụng các thư viện hỗ trợ việc validation form... Bởi vậy, trong bài này mình sẽ cùng nhau chỉ ra và đi xây dựng ví dụ về form trong ReactJS.

Table of Content

- 1. Thao tác với Form trong ReactJS
 - Lấy giá trị của input
 - Submit Form
 - Validation Form
- 2. Xây dựng ví dụ form trong React JS

25. Thao tác với Form trong ReactJS

Dưới đây là các thao tác mà bạn có thể làm việc với form trong ReactJS, hầu như đây là những thao tác quen thuộc.

xxviii. Lấy giá trị của input

Chúng ta có thể lấy giá trị của input bằng cách bắt sự kiện `onChange` của input.

Trước tiên, chúng ta sẽ tạo state dùng để chứa giá trị của input trong hàm `constructor()`.

```
1  constructor(props) {  
2      super(props);  
3      this.state = {  
4          email : ''  
5      }  
6  }
```

Tiếp theo, chúng ta sẽ bắt sự kiện `onChange` trong input :

```
1 <input type="email" onChange={(event) => this.changeInputValue(event)} />
```

Khi sự kiện `onChange` được kích hoạt, chúng ta sẽ có một biến là `event`, trong đó sẽ chứa các thông tin của input như `name`, `value`,...Ở đây mình truyền biến `event` vào trong function `changeInputValue()`.

Sau đó, chúng ta sẽ xây dựng hàm `changeInputValue()` dùng để thay đổi state (cái mà sẽ lưu giá trị của input) mà mình đã khởi tạo trước đó.

```
1 changeInputValue(event) {  
2     // Cập nhật state  
3     this.setState({  
4         [event.target.name]: event.target.value  
5     })  
6 }
```

Lúc này, bạn đã thực hiện xong các bước để lấy giá trị của input trong form. Tất cả các giá trị đó được lưu trong state.

xxix. Submit Form

Sau khi lấy giá trị của input, form cần được submit, bạn có thể thực hiện submit form bằng cách bắt sự kiện `onSubmit` trong form.

```
1 <form onSubmit={(event) => {  
2     this.submitForm(event)  
3 }}>
```

khi sự kiện `onSubmit` được thực thi thì biến `event` chứa thông tin của form sẽ tồn tại, chúng ta sẽ truyền nó vào trong hàm `submitForm()` để xử lý.

```
1 submitForm(event) {  
2     //Chặn sự kiện mặc định của form  
3     event.preventDefault()  
4     //In ra giá trị của input trong form
```

```

5      console.log(this.state)
6    }

```

xxx. Validation Form

Bạn có thể thực hiện validation form trong ReactJS bằng cách xây dựng một hàm validation, giả sử mình có một hàm kiểm tra email:

```

1  validationForm() {
2    const re = /\S+@\S+\.\S+\/;
3    //Kiểm tra email
4    if (re.test(this.state.email)) return false;
5    return true;
6  }

```

và thêm nó vào hàm `submitForm()` đã khởi tạo trước đó :

```

1  submitForm(event) {
2    //Chặn sự kiện mặc định của form
3    event.preventDefault()
4    //Validaton form
5    if(!this.validationForm()) {
6      alert('Email không đúng định dạng.')
7    }
8  }

```

26. Xây dựng ví dụ form trong React JS

Trong phần này mình sẽ đi xây dựng ví dụ về xử lý form trong ReactJS, ở đây mình sẽ đi xây dựng một trang đăng nhập đơn giản bao gồm chức năng validation.

Ở phần khởi tạo giao diện mình sử dụng Bootstrap 4 để xây dựng giao diện, bởi vậy bạn cần thêm thư viện này vào trong file `public/index.html`:

```
1 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/boo
```

Từ bây giờ chúng ta sẽ làm việc với các file trong thư mục `src`

Trước tiên, chúng ta sẽ đi xây dựng giao diện cho form, trong file `App.js` chúng ta sẽ đi xây dựng giao diện và khởi tạo state chứa giá trị của input.

```
1 import React from "react";
2
3 class App extends React.Component {
4   constructor(props) {
5     super(props);
6     //Khởi tạo state chứa giá trị của input
7     this.state = {
8       email: "",
9       password: ""
10    };
11  }
12  render() {
13    return (
14      <div className="container" style={{ paddingTop: "5%" }}>
15        <form
16          onSubmit={e => {
17            this.submitForm(e);
18          }}
19        >
20          <div className="form-group">
21            <label htmlFor="text">Email:</label>
22            <input
23              type="text"
```



```

24         className="form-control"
25         name="email"
26         placeholder="Enter email"
27         onChange={e => this.changeInputValue(e)}
28     />
29 </div>
30 <div className="form-group">
31     <label htmlFor="pwd">Password:</label>
32     <input
33         type="password"
34         className="form-control"
35         name="password"
36         placeholder="Enter password"
37         onChange={e => this.changeInputValue(e)}
38     />
39 </div>
40 <button type="submit" className="btn btn-primary">
41     Submit
42 </button>
43 </form>
44
45 </div>
46     );
47 }
48 }
49 export default App;

```

Tiếp theo, xây dựng hàm `changeInputValue()` có nhiệm vụ lấy giá của input sau đó cập nhật vào state.

```

1  changeInputValue(e) {
2      this.setState({
3          [e.target.name]: e.target.value
4      });
5  }

```

Xây dựng hàm `validationForm()` để kiểm tra các giá trị khi submit form :

```

1  validationForm() {
2      let returnData = {
3          error : false,
4          msg: ''
5      }
6      const {email, password} = this.state
7      //Kiểm tra email
8      const re = /\S+@\S+\.\S+\/;
9      if (!re.test(email)) {
10         returnData = {
11             error: true,
12             msg: 'Không đúng định dạng email'
13         }
14     }
15     //Kiểm tra password
16     if(password.length < 8) {
17         returnData = {
18             error: true,
19             msg: 'Mật khẩu phải lớn hơn 8 ký tự'
20         }
21     }

```

```
22         return returnData;
23     }
```

Cuối cùng, xây dựng hàm `submitForm()` cho sự kiện submit form:

```
1     submitForm(e) {
2         //Chặn các event mặc định của form
3         e.preventDefault();
4
5         //Gọi hàm validationForm() dùng để kiểm tra form
6         const validation = this.validationForm()
7
8         //Kiểm tra lỗi của input trong form và hiển thị
9         if (validation.error) {
10             alert(validation.msg)
11         }else{
12             alert('Submit form success')
13         }
14     }
```

Khởi chạy dự án bằng cách mở terminal lên và gõ dòng lệnh:

```
1     npm start
```

Freetuts.net - ReactJS x +

localhost:3000 Incognito

Email:

reactjs@freetuts.net

Password:

.....

Submit

12. Render với điều kiện trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về cách để render với điều kiện trong ReactJS. Trong React, chúng ta có thể tạo ra các component riêng biệt, và chỉ định thành phần nào được render bằng cách sử dụng biểu thức điều kiện.

Render kèm điều kiện trong React, hoạt động tương tự như Javascript thông thường, bạn chỉ cần sử dụng biểu thức điều kiện if, hay [conditional operator](#) trong Javascript, sau đó React sẽ thực hiện nhiệm vụ kiểm tra điều kiện và render theo trường hợp yêu cầu. Ở đây mình có ví dụ về kiểm tra đăng nhập của người dùng:

```
1    import React, { Component } from 'react';
2
3    export default class App extends Component {
4
5        render() {
6            var isLogin = true
7            if(isLogin) {
8                return (
9                    <div>
10                       <h3>Freetuts.net</h3>
11                   </div>
12                );
13            }else{
14                return(
15                    <div>
16                       <h3>Vui lòng đăng nhập</h3>
17                   </div>
18                )
19            }
20        }
```

21 }

Dưới đây, là một vài cách mà bạn có thể thêm điều kiện render trong các component.

Table of Content

- [1. Gán element vào biến](#)
- [2. Biểu thức điều kiện trong JSX](#)
- [3. Ngăn chặn component render](#)

27. Gán element vào biến

Chúng ta có thể lưu các element vào một biến. Đây là cách mà bạn có thể tùy chọn các element được hiển thị bằng cách thêm điều kiện vào cho nó. Giả sử, mình sẽ viết lại ví dụ ở trên bằng cách gán element vào biến.

```
1      import React, { Component } from "react";
2
3      export default class App extends Component {
4          render() {
5              const isLogin = true;
6
7              if (isLogin) {
8                  //Gán element vào một biến
9                  var notification = <h3>Freetuts.net</h3>;
10             } else {
11                 //Gán element vào một biến
12                 var notification = <h3>Vui lòng đăng nhập</h3>;
13             }
14
15             return <div>{notification}</div>;
16         }
17     }
```

Chúng ta sẽ nhận được kết quả giống như ví dụ ở đầu bài, ngoài ra bạn còn có thể sử dụng cú pháp conditional operator, ở đây mình có ví dụ :

```
1    import React, { Component } from "react";
2
3    export default class App extends Component {
4      render() {
5        const isLogin = true;
6        //Conditional operator
7        const notification = (isLogin) ? <h3>Freetuts.net</h3> : <h3>Vui lòng đăng nhậ
8
9        return <div>{notification}</div>;
10     }
11 }
```

28. Biểu thức điều kiện trong JSX

Ngoài cách sử dụng các biểu thức điều kiện bên ngoài hàm `return()` như ví dụ bên trên, chúng ta còn có thể sử dụng nó trong JSX bằng cách viết nó trong dấu ngoặc {}, chúng ta vẫn sẽ sử dụng ví dụ ở đầu bài:

```
1    import React, { Component } from "react";
2
3    export default class App extends Component {
4      render() {
5        const isLogin = true;
6        return (
7          <div>
8            {isLogin ? (<h3>Freetuts.net</h3>) : (<h3>Vui lòng đăng nhập</h3>)}
9          </div>
10        );
11 }
```

```
11      }  
12    }  
13  }
```

Trong JSX bạn chỉ có thể sử dụng cú pháp conditional operator, kết quả hiển thị của ví dụ vừa rồi cũng có kết quả tương tự với ví dụ đầu tiên.

29. Ngăn chặn component render

Trong một số trường hợp nào đó, bạn sẽ muốn một component tự ẩn đi dù nó được render trong một component khác. Để làm được điều đó, ta sẽ trả về null thay vì trả về JSX.

```
1  const Demo = (props) => {  
2    //Return về null để không hiển thị  
3    return null  
4  }
```

Giả sử mình có một component hiển thị các thông báo:

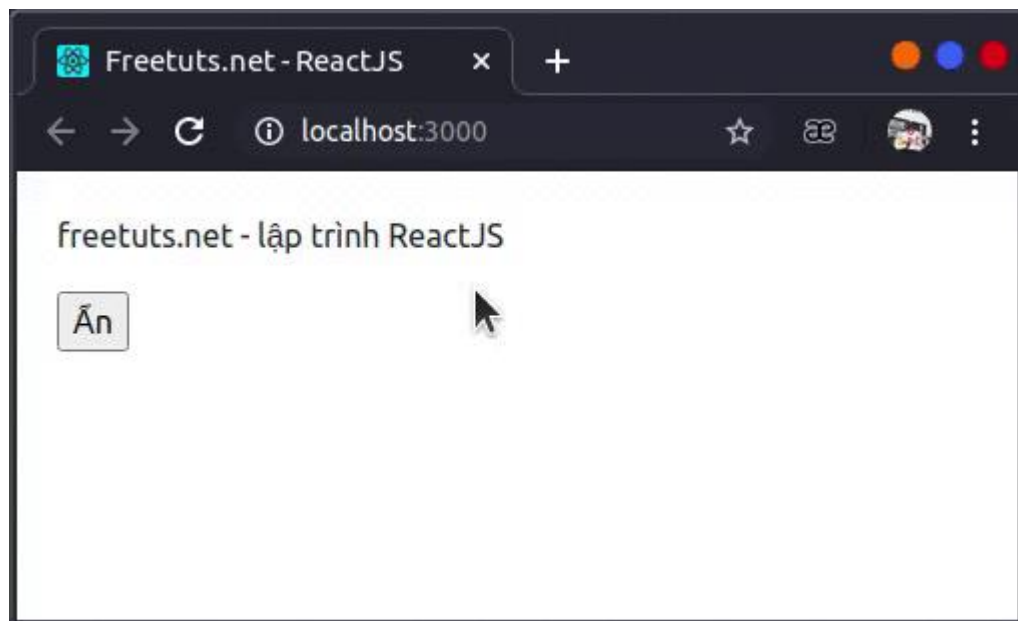
```
1  import React, { Component } from "react";  
2  const Notification = (props) => {  
3    //Kiểm tra giá trị của props  
4    if(props.isShow) {  
5      //Trả về JSX để hiển thị  
6      return (  
7        <ul>  
8          <li>Thông báo 1</li>  
9          <li>Thông báo 2</li>  
10         </ul>  
11       )  
12     }else{  
13       //Trả về null để ẩn  
14       return null  
15     }
```



```

16     }
17     export default class App extends Component {
18         constructor(props) {
19             super(props);
20             this.state = {
21                 isShowNotification: false
22             };
23         }
24         render() {
25
26             const {isShowNotification} = this.state
27
28             return (
29                 <div style={{margin: 20}}>
30                     <p>freetuts.net - lập trình ReactJS</p>
31                     <button onClick = {() => {
32                         //Cập nhật lại state
33                         this.setState({
34                             isShowNotification: !isShowNotification
35                         })
36                     }}>{isShowNotification ? 'Hiển thị' : 'Ẩn'}</button>
37
38                     {/* Gọi component Notification */}
39                     <Notification isShow = {isShowNotification}/>
40                 </div>
41             );
42         }
43     }

```



13. Tìm hiểu về List và Keys trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về lists và keys trong ReactJS, đây cũng là phần mà chúng ta sẽ làm việc rất nhiều trong React.

Table of Content

- [1. Lists trong React](#)
- [2. Keys trong React](#)
- [3. Một vài lưu ý khi sử dụng Keys](#)
 - [Keys là duy nhất](#)
 - [Tránh chỉ định index làm key](#)

30. Lists trong React

Việc khởi tạo các lists trong React, tương tự như khởi tạo lists trong Javascript. Ở đây mình sẽ tiến hành khởi tạo một lists các items.

```
1    import React from "react";
2
3    function ListComponent(props) {
4        const myList = ["php", "javascript", "python", "C++"];
5        const listItems = myList.map((item) =>
6            <li>{item}</li>
7        );
8
9        return (
10            <ul>{listItems}</ul>
11        );
12    }
13
14    export default ListComponent
```

và trình duyệt sẽ hiện thị kết quả:

- php
- javascript
- python
- C++

Việc khởi tạo các lists trong React rất đơn giản, điều mình muốn tập chung nhấn mạnh trong bài viết này đó là về **keys** mà mình sẽ đề cập bên dưới.

31. Keys trong React

Trong quá trình làm việc với React, chúng ta phải thao tác với danh sách(lists) rất nhiều như danh sách các ảnh, danh sách các item trong giỏ hàng,...Khi các lists này có hàng tá các items thì React rất khó có thể kiểm soát được items. Bởi vậy chúng ta cần phải chỉ định cho nó một key để định danh.

Nếu bạn chạy ví dụ ở phần thứ 1, React sẽ hiển thị cảnh báo như hình bên dưới:

✖ Warning: Each child in a list should have a unique "key" prop.
 Check the render method of `ListComponent`. See <https://fb.me/react-warning-keys> for more information.
 in li (at App.js:4)
 in ListComponent (at src/index.js:9)
 in StrictMode (at src/index.js:8)

Để loại bỏ cảnh báo bạn phải chỉ định cho các items trong lists một thuộc tính có tên là **key**. Chúng ta sẽ sửa ví dụ ở đầu bài thành:

```
1   import React from "react";
2
3   function ListComponent(props) {
4     const myList = [
5       {
6         id : 'p',
7         name : 'php'
8       },
9       {
10        id : 'j',
11        name : 'javascript'
```

```

12     },
13     {
14         id : 'py',
15         name : 'python'
16     },
17     {
18         id : 'c',
19         name : 'C++'
20     },
21 ]
22
23 //Thêm thuộc tính key vào trong thẻ jsx
24 const listItems = myList.map((item) =>
25     <li key = {item.id}>{item.name}</li>
26 );
27
28 return (
29     <ul>{listItems}</ul>
30 );
31 }
32
33 export default ListComponent

```

Kết quả của ví dụ vẫn tương tự, nhưng chúng ta đã hoàn toàn loại được cảnh báo. React khuyên chúng ta nên chỉ định các key duy nhất trong các lists.

32. Một vài lưu ý khi sử dụng Keys

Ở đây mình có một vài lưu ý sử dụng key cho list, các lưu ý này sẽ giúp quá trình làm việc với React không gặp các lỗi không mong muốn.

xxxi. Keys là duy nhất

Bạn cần chỉ định các keys này là duy nhất, các keys này không được trùng lặp trong các lists.

```
1    const myList = [  
2      {  
3        id : 'p',  
4        name : 'php'  
5      },  
6      {  
7        id : 'j',  
8        name : 'javascript'  
9      },  
10     {  
11       id : 'p',  
12       name : 'python'  
13     },  
14     {  
15       id : 'c',  
16       name : 'C++'  
17     },  
18   ]  
19  
20   const listItems = myList.map((item) =>  
21     <li key = {item.id}>{item.name}</li>  
22   );
```

khi các keys này trùng lặp bạn sẽ nhận được cảnh báo :

```
1    Warning: Encountered two children with the same key, `p`.  
2    Keys should be unique so that components maintain their identity across updates.  
3    Non-unique keys may cause children to be duplicated and/or omitted – the behavior is
```

Các keys chỉ cần là duy nhất khi so sánh với các anh/chị của nó trong lists chứa chúng.

xxxii. Tránh chỉ định index làm key

Trong một vài trường hợp bạn thường chỉ định giá trị của biến index thành keys như trong ví dụ này:

```
1    const listItems = myList.map((item, index) =>
2      <li key = {index}>{item.name}</li>
3    );
```

React khuyên chúng ta không nên sử dụng cách này. Bởi khi bạn thực hiện sắp xếp mảng thì index sẽ thay đổi, React lại phải xác định lại keys một lần nữa, gây ra giảm hiệu suất làm việc.

Chỉ sử dụng index làm key trong khi:

- Nếu list của bạn là tĩnh và sẽ không thay đổi.
- List sẽ không bao giờ được sắp xếp lại.
- List sẽ không được lọc (thêm / xóa các mục khỏi danh sách).
- Không có id cho các mục trong list.

Hãy chỉ sử dụng index làm key trong trường hợp đặc biệt này, và lưu ý trong quá trình sử dụng.

14. Kỹ thuật Lifting State Up trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về Lifting State Up trong ReactJS, trong quá trình làm việc với React, các component phải re-render và thay đổi rất nhiều lần. Trong một vài trường hợp, bạn muốn khi 1 component con được thay đổi, đồng thời bạn muốn component cha cũng sẽ bắt được hành động đó. Theo tài liệu chính thức của ReactJS thì Lifting State Up được định nghĩa là rằng :

several components need to reflect the same changing data. We recommend lifting the shared state up to their closest common ancestor

chúng ta có thể hiểu rằng

một vài component phải cùng nhận được sự thay đổi. Chúng tôi khuyên bạn nên chia sẻ các state cho cha, mẹ, ông bà dòng họ của chúng :v

Đọc đến đây chắc hẳn bạn cũng sẽ hiểu một phần, kỹ thuật lifting state up là cách mà có thể chia sẻ dữ liệu cho các component khác. Có nghĩa là khi bạn thay đổi dữ liệu của 1 component con thì bạn sẽ gửi dữ liệu cho component cha biết.

Table of Content

- [1. Lifting State Up trong React](#)
- [2. Ví dụ cụ thể](#)

33. Lifting State Up trong React

Trong phần này mình sẽ chỉ ra các bước thực hiện **Lifting State Up**, các bước thực hiện rất đơn giản và chúng ta sẽ có ví dụ cụ thể ở phần tiếp theo.

Trước tiên, chúng ta sẽ truyền cho component con một props có giá trị là một function. Function này sẽ được gọi khi component con trả về dữ liệu:

```
1    function ComponentCha(props) {  
2        //Hàm này sẽ được gọi khi nhận được dữ liệu  
3        const receiveData = function (data) {  
4            console.log('Data nhận được', data )  
5        }  
6        //Gọi compoennt con và truyền vào một props
```



```

7      //có giá trị là một hàm
8      return (
9          <ComponentCon onReceiveData = {receiveData}/>
10     )
11 }

```

Tiếp theo, ở `componentCon` chúng ta sẽ tiến hành gửi lại cho `componentCha` bằng cách truyền vào props `onReceiveData` giá trị cần gửi về.

```

1      function ComponentCon (props) {
2          return(
3              <div>
4                  <button onClick={() => {
5
6                      // Chúng ta sẽ gọi props có tên là
7                      // receiveData đã được truyền từ componentCha.
8                      // Và truyền vào đó giá trị cần gửi
9
10                     props.onReceiveData('data gửi đi')
11
12                     }}>Gửi lại cho componentCha</button>
13              </div>
14          )
15      }

```

Vậy là chúng ta đã gửi thành công dữ liệu từ `componentCon` sang `componentCha`. Lúc này `componentCha` có thể sử dụng dữ liệu đó. Đây là các bước cơ bản để sử dụng Lifting State Up. Chúng ta sẽ đi vào ví dụ cụ thể bên dưới.

34. Ví dụ cụ thể

Ở đây chúng ta sẽ đi vào ví dụ cụ thể về cách sử dụng kỹ thuật Lifting State Up. Mình sẽ đi xây dựng một ứng dụng cho phép chuyển đổi giữa USD và VNĐ. Chúng ta sẽ làm việc với file `App.js`

Trước tiên, chúng ta sẽ đi xây dựng component cha có tên `Caculator`, và hàm `handleChange` chịu trách nhiệm data từ component con.

```
1    export default class App extends Component {
2      constructor(props) {
3        super(props);
4        this.state = {
5          usd: 0,
6          vnd: 0,
7        };
8      }
9      handleChange = (data) => {
10        this.setState(data);
11      };
12
13      render() {
14        return (
15          <div style={{margin: "3%"}}>
16            <USDtoVND onHandleChange={this.handleChange} value={this.state.usd} />
17            <VNDtoUSD onHandleChange={this.handleChange} value={this.state.vnd} />
18            <hr />
19            <code>freetuts.net</code>
20          </div>
21        );
22      }
23    }
```

Mình sẽ truyền cho 2 component con có tên `USDtoVND` và `VNDtoUSD` các props như:

- **onHandleChange:** giúp cho component con có thể gửi dữ liệu về component cha.
- **value:** giá trị của input.

Tiếp theo, chúng ta sẽ khởi tạo các component con có tên `USDtoVND` và `VNDtoUSD`:

```
1    const USDtoVND = function (props) {
2      const convert = function (usd) {
3        return usd * 23632;
4      };
5      return (
6        <div>
7          <span>USD </span>
8          <input
9            onChange={ (e) => {
10              const usd = e.target.value;
11              const vnd = convert(usd);
12              props.onHandleChange({
13                usd,
14                vnd,
15              });
16            }}
17            value={props.value}
18          />
19        </div>
20      );
21    };
22    const VNDtoUSD = function (props) {
23      const convert = function (vnd) {
24        return vnd / 23632;
25      };
26      return (
27        <div>
```

```

27         <span>VND </span>
28         <input
29             onChange={ (e) => {
30                 const vnd = e.target.value;
31                 const usd = convert(vnd);
32                 props.onHandleChange({
33                     usd,
34                     vnd,
35                 });
36             }}
37             value={props.value}
38         />
39     </div>
40 );
41 };
42

```

các component này có nhiệm vụ chuyển từ đổi tiền và trả về cho component cha bằng prop `onHandleChange()`.

Cuối cùng, chúng ta sẽ có file `App.js` đầy đủ như sau:

```

1     import React, { Component } from "react";
2
3     const USDtoVND = function (props) {
4         const convert = function (usd) {
5             return usd * 23632;
6         };
7         return (
8             <div>

```

```

9      <span>USD </span>
10    <input
11      onChange={ (e) => {
12        const usd = e.target.value;
13        const vnd = convert(usd);
14        props.onHandleChange({
15          usd,
16          vnd,
17        });
18      }}
19      value={props.value}
20    />
21  </div>
22  );
23 };
24 const VNDtoUSD = function (props) {
25   const convert = function (vnd) {
26     return vnd / 23632;
27   };
28   return (
29     <div>
30       <span>VND </span>
31       <input
32         onChange={ (e) => {
33           const vnd = e.target.value;
34           const usd = convert(vnd);
35           props.onHandleChange({
36             usd,

```

```

37         vnd,
38     });
39     }}
40     value={props.value}
41     />
42 </div>
43 );
44 };
45
46 export default class App extends Component {
47     constructor(props) {
48         super(props);
49         this.state = {
50             usd: 0,
51             vnd: 0,
52         };
53     }
54     handleChange = (data) => {
55         this.setState(data);
56     };
57
58     render() {
59         return (
60             <div style={{margin: "3%"}}>
61                 <USDtoVND onHandleChange={this.handleChange} value={this.state.usd} />
62                 <VNDtoUSD onHandleChange={this.handleChange} value={this.state.vnd} />
63                 <hr />
64                 <code>freetuts.net</code>

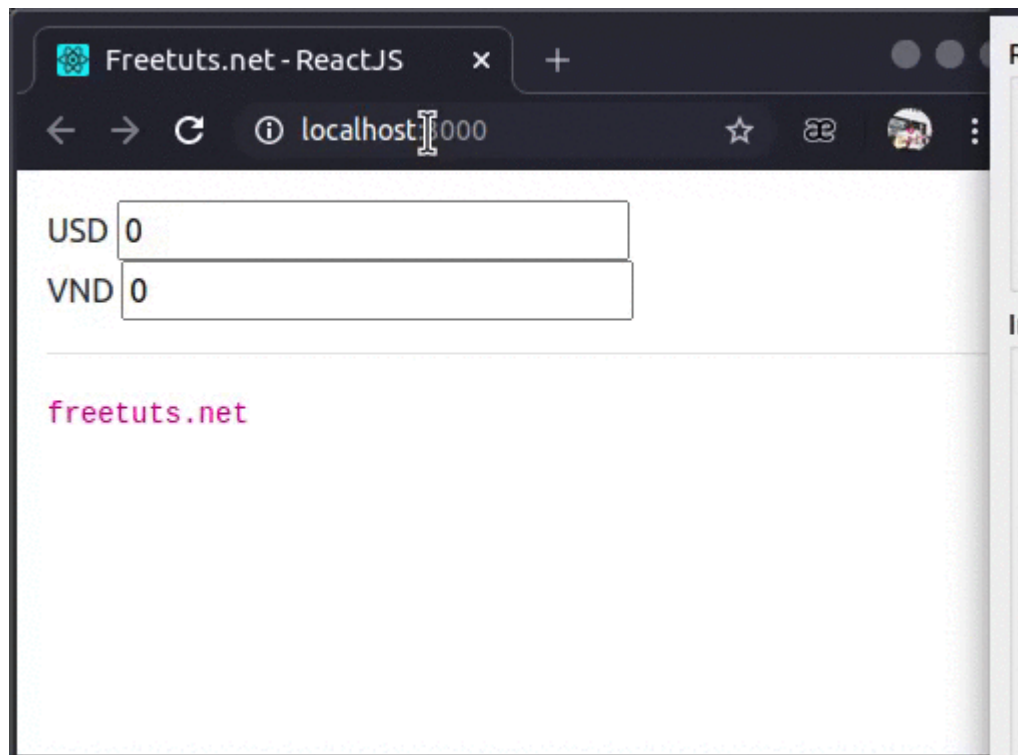
```

```
65         </div>
66     );
67 }
68 }
```

Khởi chạy dự án bằng cách mở terminal và gõ dòng lệnh:

```
1 npm start
```

Truy cập đường dẫn **http://localhost:3000** và chúng ta sẽ thấy kết quả:



Kỹ thuật Lifting State Up chỉ có thể sử dụng khi bạn truyền từ component cha sang con và ngược lại. Tức là bạn chỉ có thể truyền dữ liệu từ 1 component quan hệ cha con. Trong trường hợp, bạn muốn chia sẻ dữ liệu state sang component khác, chúng ta sẽ có khái niệm về Redux, cho phép chúng ta tạo ra local state, giúp việc chia sẻ dữ liệu giữa các component được dễ dàng hơn.

15. Tìm hiểu về Refs trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi **tìm hiểu về refs trong ReactJS**. Trong quá trình làm việc với React ở một vài trường hợp nào đó bạn cần phải thực hiện việc tham chiếu đến DOM, mặc dù điều này không được khuyến khích sử dụng. Để giúp quá trình tham chiếu đến DOM dễ dàng hơn, React cung cấp cho chúng ta **React Refs**.

Table of Content

- [1. React Refs là gì?](#)
- [2. Sử dụng React Refs](#)
 - [Khởi tạo một Ref](#)
 - [Forwarding Refs](#)

35. React Refs là gì?

React refs là một tính năng hữu ích, nó là cách mà chúng ta có thể để tham chiếu một element trong DOM hoặc từ một class component con đến component cha. Điều này cho phép chúng ta đọc và chỉnh sửa các element đó.

Cách giải thích dễ hiểu nhất về React Refs là tưởng tượng nó là một cây cầu. Khi một element được gán vào một ref nó sẽ cho phép chúng ta sửa đổi cũng như truy cập vào element đó ngay lập tức và không cần sử dụng đến props hay state để component re-render lại. Nó giống cho phép việc can thiệp vào DOM như trong Javascript DOM:

```
document.getElementsByTagName('h1').innerHTML = 'Freetuts.net'
```

Chúng ta có thể can thiệp trực tiếp vào DOM qua refs mà không cần thông qua việc render. Mặc dù đây là cách để can thiệp vào DOM thuận tiện mà không cần phải sử dụng đến state, props nhưng điều này React không khuyến khích. Bởi khi các DOM bị thay đổi thì nó sẽ ảnh hưởng một phần nào đó đến quá trình render các component. Các bạn nên sử dụng **React refs** để can thiệp vào DOM trong trường hợp cần thiết.

36. Sử dụng React Refs

Trong phần này chúng ta sẽ cùng nhau đi tìm hiểu về cách làm việc với React Refs trong React

xxiii. Khởi tạo một Ref

Refs thường được chỉ định trong hàm tạo trong **constructor** ở class component và như một biến ở functional component. Sau đó được gắn vào một element trong hàm **render()**. Ở đây chúng ta sẽ tạo một **refs** và gắn vào element **input** :

```
1    import React from "react";
2
3
4    export default class MyComponent extends React.Component {
5      constructor(props) {
6        super(props);
7        //Khởi tạo một ref
8        this.myRef = React.createRef();
9      }
10
11     render() {
12       return (
13         <input
14           name="email"
15           onChange={this.onChange}
16           ref={this.myRef}
17           type="text"
18         />
19       );
20     }
21   }
```

Ở ví dụ bên trên mình đã khởi tạo một **ref** bằng cách sử dụng hàm **React.createRef()** và gán giá trị của nó vào thuộc tính **myRef** trong class **MyComponent**. Lúc này, chúng ta chỉ cần gán refs vào element **input**. Khi một **ref** được gắn vào một element, element đó có thể được truy cập và sửa đổi thông qua **ref**. Cụ thể trong trường hợp này là **input**

Tiếp theo, mình sẽ thêm một **button** cho ví dụ bên trên, khi click vào button đó nó sẽ tham chiếu đến input qua refs và thực hiện **focus** input đó.

```
...

handleClick = () => {

  //thuộc tính current trong refs cho phép

  //chúng ta chỉ định element hiện tại được tham chiếu.

  this.myRef.current.focus();

}

render() {

  return (

    ...

    <button onClick={this.handleClick}>

      Focus Input

    </button>

  </>

  )

}
```

Thuộc tính `current` trong `myRefs` chứa giá trị element được tham chiếu khi element đó được render. Bằng cách sử dụng React ref chúng ta sẽ thay đổi trạng thái của input mà không cần phải sử dụng đến state hay props. Trong trường hợp, bạn chỉ muốn focus một input hay làm mờ hình ảnh thì bạn chỉ cần sử dụng refs mà không cần phải re-render lại component để chỉ làm những việc này.

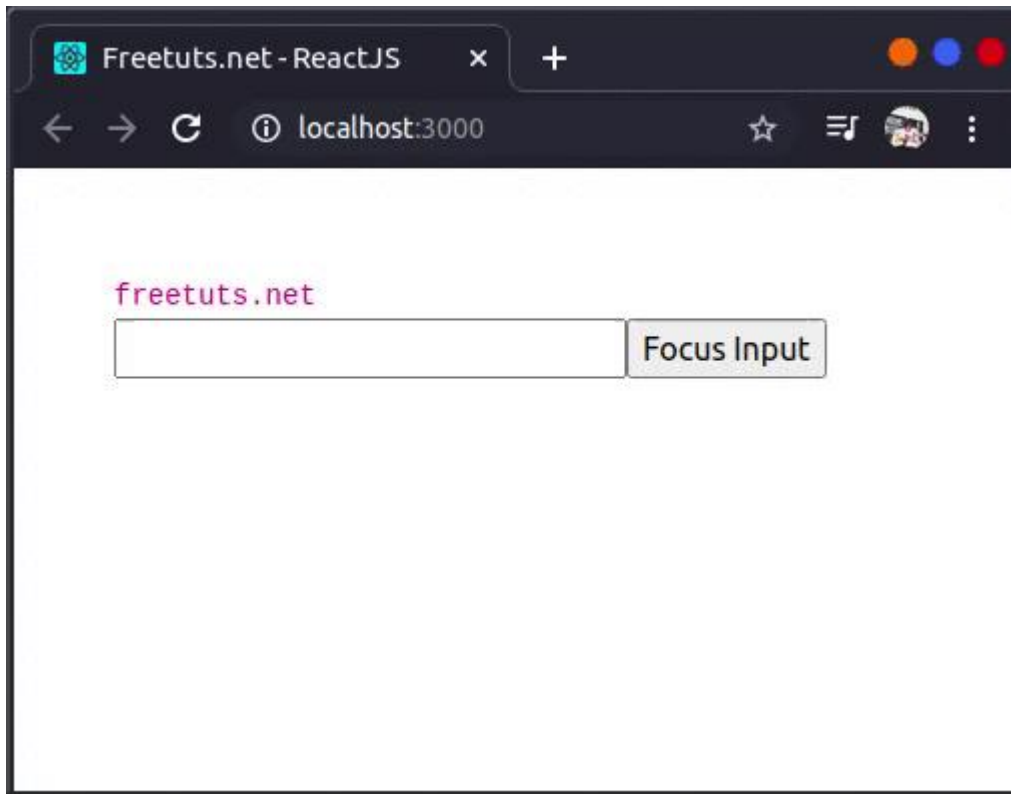
Cuối cùng, chúng ta sẽ có ví dụ đầy đủ:

```
1   import React from "react";
2
3
4   export default class MyComponent extends React.Component {
5     constructor(props) {
6       super(props);
7       this.myRef = React.createRef();
8     }
9     handleClick = () => {
10       this.myRef.current.focus();
11     }
12     render() {
13       return (
14         <>
15           <code>freetuts.net</code>
16           <input
17             name="email"
18             onChange={this.onChange}
19             ref={this.myRef}
20             type="text"
21           />
22           <button onClick={this.handleClick}>
23             Focus Input
```

```

24         </button>
25     </>
26     );
27 }
28 }

```



xxxiv. *Forwarding Refs*

Forwarding Refs là một kỹ thuật để tự động chuyển một ref từ một component tới component con, cho phép component cha có thể tham chiếu tới các element của component con để đọc và chỉnh sửa nó.

React cung cấp cho chúng ta một cách thức để thực hiện việc chuyển tiếp một ref, chúng ta sẽ bao component con trong `React.forwardRef()`, ở đây mình có ví dụ:

```

1    //Component Con
2    const MyInput = React.forwardRef((props, ref) => {
3        return(<input name={props.name} ref={ref} />);
4    });

```

```

5    // Component Cha
6    const MyComponent = () => {
7      let ref = React.createRef();
8      return (
9        <MyInput
10         name="email"
11         ref={ref}
12       />
13     );
14   }

```

Ví dụ bên trên mình đã sử dụng `React.forwardRef()`, ở đây nó cung cấp cho chúng ta 2 tham số lần lượt là **props** và **refs**, cho chúng ta nhận về giá trị của **props** và **refs** từ component cha.

Chúng ta có thể sử dụng `React.forwardRef()` trong class component bằng cách này sử dụng HOC (higher order component)

```

1    // Lấy ref thông qua props
2    class WrappedComponent extends Component {
3      render() {
4        return (
5          <input
6            type="text"
7            name={this.props.name}
8            ref={this.props.innerRef}
9          />
10         )
11       }
12     }
13
14    // Gói componentnt được bọc của chúng tôi với ForwardRef, truyền vào props giá trị
15    const MyInput = React.forwardRef((props, ref) => {

```

```

15     return (<WrappedComponent innerRef={ref} {...props} />);
16   });
17   export default MyInput;

```

Và cuối cùng chúng ta sẽ làm ví dụ có chức năng giống ví dụ ban đầu nhưng chúng ta sẽ tham chiếu đến element `input` trong component con.

```

1   import React from "react";
2
3   //Component Con
4   const MyInput = React.forwardRef((props, ref) => {
5     return <input name={props.name} ref={ref} />;
6   });
7   // Component Cha
8   const MyComponent = () => {
9     let ref = React.createRef();
10    const handleButton = () => {
11      ref.current.focus();
12    };
13    return (
14      <>
15        <code>freetuts.net</code>
16        <MyInput name="email" ref={ref} />
17        <button onClick={handleButton}>Focus Input</button>
18      </>
19    );
20  };
21  export default MyComponent;

```

Kết quả cũng tương tự như ví dụ ban đầu.

16. Tìm hiểu về Context trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về **Context trong ReactJS**. Trong quá trình làm việc với ReactJS, các dữ liệu trong các component phải được chia sẻ với nhau.

Chúng ta thực hiện điều này bằng cách đưa các dữ liệu này lên một nơi có tên là local state. Local state chịu trách nhiệm phân phối dữ liệu tới các component. **Context** hỗ trợ chúng ta thực hiện điều này một cách đơn giản.

Table of Content

- [1. Context trong ReactJS là gì ?](#)
- [2. Context API trong ReactJS](#)
 - [React.createContext](#)
 - [Context.Provider](#)
 - [Context.Consumer](#)
 - [Class.contextType](#)
- [3. Sử dụng Context trong ReactJS](#)

37. Context trong ReactJS là gì ?

Theo tài liệu chính thức của ReactJS định nghĩa **Context** là :

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

Chúng ta có thể hiểu:

Context cung cấp cho chúng ta cách để thực hiện chia sẻ dữ liệu tới các component trong cây mà không cần truyền dữ liệu qua props theo từng cấp bậc.

Khi truyền dữ liệu tới các component bằng props thì bạn chỉ có thể truyền từ component cha sang component con. Nếu bạn muốn truyền sang component cháu hoặc sang component họ hàng xa thì điều này rất phức tạp. Bởi vậy **context** sẽ là kênh giao tiếp cho các component, cho phép bạn truyền dữ liệu một cách đơn giản hơn rất nhiều.

Ở đây mình có một ví dụ, mình muốn chuyển lời nhắn từ **ComponetÔng** sang **Componet Cháu** :

Khi sử dụng truyền dữ liệu qua props thì mình bắt buộc phải gửi lời nhắn qua (**Ông -> Cha -> Cháu**)

```

1   import React from "react";
2   const ComponentChau = (props) => {
3     return <h1>Ông bảo là "{props.message}"</h1>;
4   };
5   const ComponentCha = (props) => {
6     return <ComponentChau {...props} />;
7   };
8
9   const ComponentOng = () => {
10    const message = "Vào freetuts.net học lập trình";
11    return <ComponentCha message={message} />;
12  };
13  export default ComponentOng;

```

Còn nếu sử dụng **Context** thì chúng ta sẽ gửi trực tiếp từ Ông đến Cháu luôn :

```

1   import React from "react";
2
3   const MessageContext = React.createContext();
4
5   class ComponentChau extends React.Component {
6     render() {
7       return <h1>Ông bảo là : "{this.context}"</h1>;
8     }
9   }
10  ComponentChau.contextType = MessageContext;
11
12  const ComponentOng = () => {
13    return (

```



```

14      <MessageContext.Provider value="Vào freetuts.net học lập trình">
15        <ComponentChau />
16      </MessageContext.Provider>
17    );
18  };
19  export default ComponentOng;

```

Kết quả của 2 cách này vẫn tương tự nhưng việc sử dụng **Context** sẽ đơn giản hơn nhiều.

38. Context API trong ReactJS

Trước tiên, chúng ta sẽ đi tìm hiểu về **Context API**, sau đó sẽ đi vào ví dụ cụ thể nhé. Ở đây chúng ta có một vài API mà React cung cấp.

xxxv. React.createContext

```

1  const MyContext = React.createContext(defaultValue);

```

Khởi tạo một **Context Object**, giá trị của `defaultValue` là giá trị mặc định của props value trong `Provider`.

xxxvi. Context.Provider

```

1  <MyContext.Provider value={/* some value */}>

```

Mỗi **Context Object** phải đi kèm với một `Provider`, nó cho phép bạn nhận về sự thay đổi của `context`.

xxxvii. Context.Consumer

```

1  <MyContext.Consumer>
2    {value => /* render something based on the context value */}
3  </MyContext.Consumer>

```

Một React component được khởi chạy mỗi khi giá trị của context thay đổi, và nhận về giá trị của context đó.

Context.displayName

```

1  const MyContext = React.createContext(/* some value */);
2  MyContext.displayName = 'MyDisplayName';

```

3

4 <MyContext.Provider> // "MyDisplayName.Provider" in DevTools

5 <MyContext.Consumer> // "MyDisplayName.Consumer" in DevTools

Đặt tên cho Context, tên này sẽ được hiển thị trong DevTools.

xxxviii. Class.contextType

```
class MyClass extends React.Component {  
  
  render() {  
  
    let value = this.context;  
  
  }  
  
}  
  
MyClass.contextType = MyContext;
```

`contextType` là một thuộc tính của class được tạo bởi `React.createContext()` được dùng để lấy giá trị của context.

39. Sử dụng Context trong ReactJS

Chúng ta sẽ sử dụng context theo 3 bước cụ thể:

1. Khởi tạo object context bằng phương thức `React.createContext()`, sau đó chúng ta sẽ nhận được 1 object bao gồm các thuộc tính quan trọng như `Provider` và `Consumer`.
2. Sử dụng `Provider` bọc quanh các component, và truyền giá trị vào props `value`
3. Thêm `Consumer` vào bất cứ đâu mà bạn muốn chia sẻ context miễn là ở bên trong `Provider`, bạn có thể lấy giá trị của context thông qua `props.children`.

Bây giờ chúng ta sẽ đi vào ví dụ cụ thể. Mình sẽ xây dựng 1 ứng dụng cho phép random các số và hiển thị. Trước tiên, mình sẽ tiến hành khởi tạo một object context.

```
1  import React from "react";
2  //Khởi tạo một
3  const NumberContext = React.createContext();
```

Sau đó, mình sẽ bọc quanh các component cần chia sẻ bằng `Provider`, và `Consumer` :

```
1  export default class ContextComponent extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = {
5        number: 0,
6      };
7    }
8    updateNumber = () => {
9      this.setState({
10        number: Math.random(),
11      });
12    };
13    render() {
14      return (
15        <NumberContext.Provider
16          value={{
17            number: this.state.number,
18            update: this.updateNumber.bind(this),
19          }}
20        >
21        <NumberContext.Consumer>
```

```

22         { () => (
23             <>
24                 <ShowNumber />
25                 <UpdateNumber />
26             </>
27         ) }
28     </NumberContext.Consumer>
29 </NumberContext.Provider>
30 );
31 }
32 }

```

bên trên mình truyền vào props **value** trong **provider** giá trị là một object có các thuộc tính như:

- **number**: Chứa giá trị của số.
- **update**: hàm thực hiện update số.

Consumer sẽ có nhiệm vụ xem xét sự thay đổi và trả về giá trị của context trong props **children**, mình sẽ chia sẻ props cho 2 component đó là **ShowNumber** và **UpdateNumber**.

Tiếp theo, mình sẽ đi xây dựng 2 component là **ShowNumber** và **UpdateNumber** mà mình đã gọi ở **Consumer**.

```

1     class UpdateNumber extends React.Component {
2         render() {
3             return (
4                 <button onClick={() => {
5                     //Gọi hàm update để thực hiện update số.
6                     console.log(this.context.update())
7                 }}>Update Number</button>
8             );
9         }

```

```

10     }
11     UpdateNumber.contextType = NumberContext;
12
13     class ShowNumber extends React.Component {
14         render() {
15             //Hiển thị ra số.
16             return (
17                 <p>{this.context.number}</p>
18             );
19         }
20     }
21     ShowNumber.contextType = NumberContext;

```

Ở đây mình sử dụng phương thức `contextType` để gán giá trị của context, lúc này thuộc tính `this.context` trong class sẽ chứa giá trị của context.

Cuối cùng, chúng ta sẽ có file đầy đủ:

```

1     import React from "react";
2     //Khởi tạo một
3     const NumberContext = React.createContext();
4
5     class UpdateNumber extends React.Component {
6         render() {
7             return (
8                 <button onClick={() => {
9                     console.log(this.context.update())
10                 }}>Update Number</button>
11             );
12         }

```

```

13     }
14     UpdateNumber.contextType = NumberContext;
15
16     class ShowNumber extends React.Component {
17         render() {
18             return (
19                 <p>{this.context.number}</p>
20             );
21         }
22     }
23     ShowNumber.contextType = NumberContext;
24
25
26     export default class ContextComponent extends React.Component {
27         constructor(props) {
28             super(props);
29             this.state = {
30                 number: 0,
31             };
32         }
33         updateNumber = () => {
34             this.setState({
35                 number: Math.random(),
36             });
37         };
38         render() {
39             return (
40                 <NumberContext.Provider

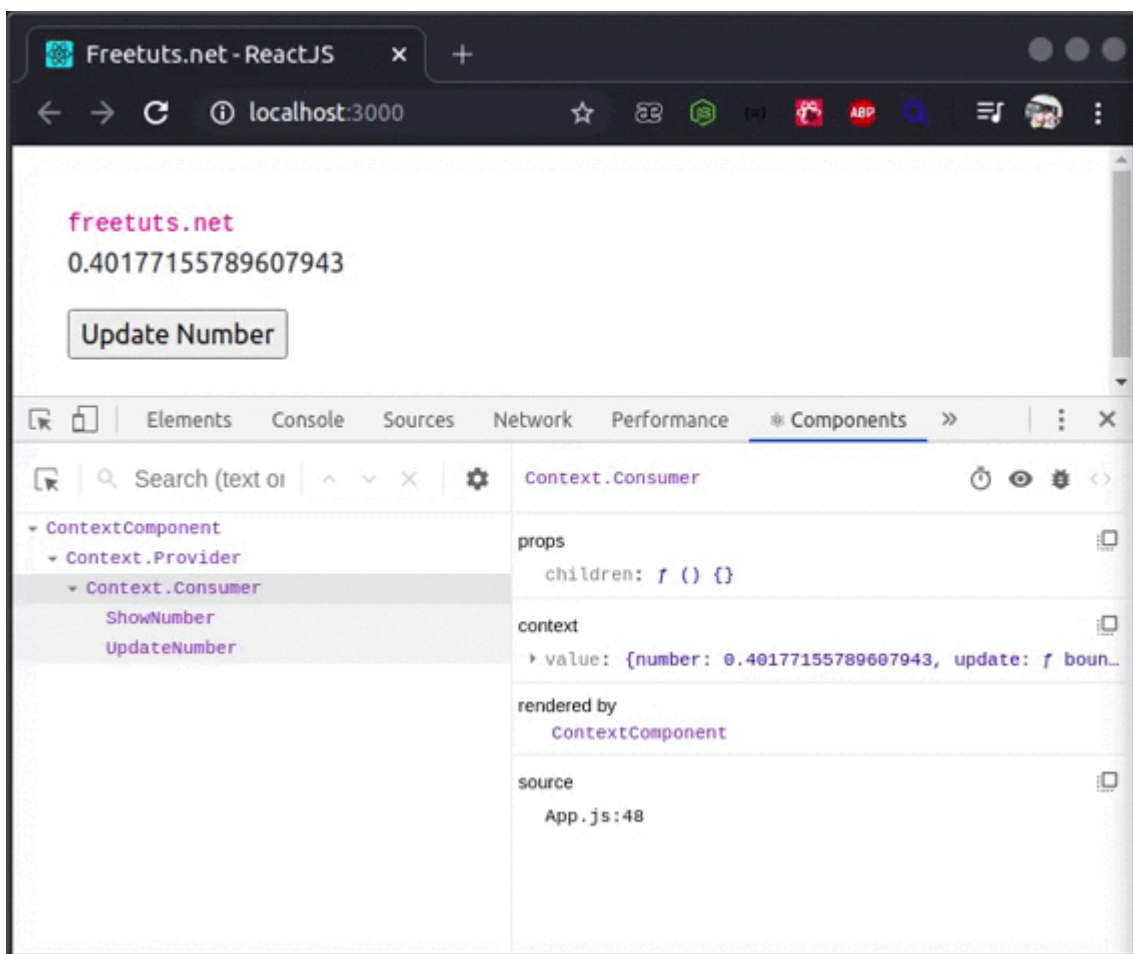
```

```

41         value={{
42             number: this.state.number,
43             update: this.updateNumber.bind(this),
44         }}
45     >
46     <NumberContext.Consumer>
47         { () => (
48             <>
49                 <ShowNumber />
50                 <UpdateNumber />
51             </>
52         ) }
53     </NumberContext.Consumer>
54 </NumberContext.Provider>
55 );
56 }
57 }

```

khởi chạy ứng dụng và xem kết quả:



17. Tìm hiểu về Fragments trong ReactJS

Trong bài này chúng ta sẽ cùng nhau đi tìm hiểu về Fragments trong ReactJS, chắc hẳn trong quá trình làm quen với React bạn gặp thông báo lỗi "*JSX parent expressions must have one parent element*" khi không bọc các element JSX quanh một element nào đó. Giả sử như thế này.

```
1 //Khai báo JSX sai.
2 return (
3   <h1>Hello, freetuts.net !</h1>
4   <p>Học lập trình ReactJS cùng Freetuts.net</p>
5 )
```

lúc này chúng ta sẽ sử dụng fragments trong React để loại bỏ lỗi.

```
1 return (
2   <React.Fragment>
3     <h1>Hello, freetuts.net !</h1>
4     <p>Học lập trình ReactJS cùng Freetuts.net</p>
5   </React.Fragment>
6 );
```

Bây giờ, chúng ta sẽ đi tìm hiểu sâu hơn về Fragments trong ReactJS, và cách nó được sử dụng như thế nào ?

Table of Content

- [1. Tại sao phải sử dụng Fragments](#)
- [2. Sử dụng fragments trong ReactJS](#)
- [3. Cú pháp của Fragments](#)
 - [React.Fragments](#)
 - [Viết tắt](#)

40. Tại sao phải sử dụng Fragments

Thông thường chúng ta dùng thẻ `div` để bao quanh các element JSX như ví dụ:

```

1    return (
2      <div>
3        <h1>Hello, freetuts.net !</h1>
4        <p>Học lập trình ReactJS cùng Freetuts.net</p>
5      </div>
6    );

```

Nhưng trong một vài trường hợp việc thêm thẻ `div` vào sẽ làm đảo lộn quy ước CSS và cấu trúc của các thẻ HTML khi được render. Giả sử trong trường hợp này.

Mình có một component có tên `Table` dùng để hiển thị bảng:

```

1    class Table extends React.Component {
2      render() {
3        return (
4          <table>
5            <tr>
6              <Columns />
7            </tr>
8          </table>
9        );
10     }
11   }

```

và component có tên `Columns` sẽ hiển thị nội dung của bảng đó. Nếu chúng ta thêm thẻ `div` để bao quanh JSX trong component `Columns` như thế này:

```

1    class Columns extends React.Component {
2      render() {
3        return (
4          <div>
5            <td>Hello</td>

```

```

6         <td>World</td>
7     </div>
8 );
9 }
10 }

```

Nó sẽ phá vỡ cấu trúc của tables và hiển thị sai, và đây là kết quả khi thực hiện render:

```

1 <table>
2   <tr>
3     <div>
4       <td>Hello</td>
5       <td>World</td>
6     </div>
7   </tr>
8 </table>

```

Lúc này bạn cần đến việc sử dụng fragments.

41. Sử dụng fragments trong ReactJS

Fragments cho phép chúng ta bọc các element JSX lại, giúp bạn triển khai các element HTML theo mong muốn. Chúng ta sẽ tiếp tục với ví dụ ở phần thứ nhất, component `Columns` sẽ được chỉnh sửa như sau :

```

1 class Columns extends React.Component {
2   render() {
3     return (
4       <React.Fragment>
5         <td>Hello</td>
6         <td>World</td>
7       </React.Fragment>
8     );

```

```

9      }
10    }

```

và khi component được render nó sẽ hiển thị theo cấu trúc bạn mong muốn.

```

1    <table>
2      <tr>
3        <td>Hello</td>
4        <td>World</td>
5      </tr>
6    </table>

```

42. Cú pháp của Fragments

Ở đây chúng ta có thể viết các fragments theo 2 cách, mỗi cách viết có những ưu điểm khác nhau.

xxxix. React.Fragments

Đây là cách viết đầy đủ, nó cho phép bạn có thể thêm các **key** vào khi triển khai các lists. Bạn có thể tham khảo thêm về [lists và keys trong ReactJS](#). Ở đây mình có một ví dụ, sẽ hiển thị ra một list :

```

1    function Glossary(props) {
2      return (
3        <dl>
4          {props.items.map(item => (
5            // Bạn phải chỉ định một keys cho mỗi items
6            // trong lists
7            <React.Fragment key={item.id}>
8              <dt>{item.term}</dt>
9              <dd>{item.description}</dd>
10           </React.Fragment>
11          ) ) }
12        </dl>

```

```
13      );
```

```
14    }
```

xl. Viết tắt

Bạn có thể viết cú pháp ngắn gọn của fragments bằng cách sử dụng 2 dấu ngoặc nhọn (`<>`)

```
1    return (  
2      <>  
3        <p>freetuts.net</p>  
4        <code>  
5          freetuts.net  
6        </code>  
7      </>  
8    );
```

khi sử dụng cú pháp ngắn gọn này bạn không thể chỉ định keys vào fragments. Trong trường hợp bạn muốn thêm key thì chúng ta sẽ dùng `React.Fragments`.

18. Tìm hiểu về Render Props trong ReactJS

Trong bài này chúng ta sẽ cùng nhau đi tìm hiểu về **Render Props trong ReactJS**. Khi triển khai một dự án React, việc tái sử dụng các component là điều rất cần thiết nhất là trong các dự án lớn. Bởi vậy, React cung cấp cho chúng ta một pattern rất hữu ích cho việc xây dựng và tái sử dụng các component đó là render props.

Table of Content

- [1. Render props trong ReactJS](#)
- [2. Triển khai ví dụ về render props trong React](#)
 - [Ví dụ 1](#)
 - [Ví dụ 2](#)

43. Render props trong ReactJS

Render props trong React là một kỹ thuật để tái sử dụng các đoạn mã, có mục đích tương tự với phương pháp sử dụng Higher Order Component, giúp chúng ta sử dụng lại logic trên nhiều component.

Nó thực hiện bằng cách *truyền vào component một props có value như là một function*. **Render props** được sử dụng trong rất nhiều các module nổi tiếng trong hệ sinh thái của react, bao gồm cả *react-router*. Chúng ta sẽ đi vào cách triển khai sau đó làm một vài ví dụ nhỏ để hiểu rõ hơn.

Để sử dụng kỹ thuật render props. Trước tiên, chúng ta sẽ truyền vào component 1 props là một function, và lúc này chúng ta sẽ nhận được giá trị trả về.

```
1    return <Freetuts render={(data) => (<p>welcome to {data.value}</p>)} />;
```

và trong component được gọi (ví dụ trên là component **Freetuts**), sẽ gọi props render và trả về giá trị cần render và giá trị cần trả về vào hàm **props.render()**.

```
1    const Freetuts = (props) => {
2      return (
3        <div>
4          {props.render({
5            value: "freetuts.net"
```

```

6      }) }
7      </div>
8    );
9  };

```

Lúc này chúng ta đã triển khai thành công kỹ thuật render props và đây là kết quả :

```

1    <div>
2      <p>welcome to freetuts.net</p>
3    </div>

```

Chắc đến đây các bạn cũng chưa hiểu rõ cách hoạt động của nó. Chúng ta sẽ đi vào ví dụ cụ thể bên dưới.

44. Triển khai ví dụ về render props trong React

Ở đây mình sẽ đi xây dựng 2 ví dụ cụ thể để các bạn hiểu rõ hơn về render props.

xli. Ví dụ 1

Chúng ta sẽ dùng xây dựng một component nhận về một object chứa danh sách người yêu của bạn sau đó hiển thị ra :)) Ở đây mình sẽ sử dụng render props để có thể tái sử dụng đoạn code nhiều lần.

Trước tiên, chúng ta sẽ đi xây dựng component có tên `ShowGirlFriends`

```

1    const ShowGirlFriends = (props) => {
2      return (
3        <ul>
4          {props.listGirlFriends.map((person, index) => {
5            props.children(person);
6            return <li key={person.id}>{person.name}</li>;
7          }) }
8        </ul>
9      );
10   };

```

component này có nhiệm vụ lấy giá trị của props `listGirlFriends` sau đó sẽ dùng `map` để render lần lượt từng người, và trả về cho props `children` giá trị của người hiện tại.

Sau đó, chúng ta chỉ cần gọi component `ShowGirlFriends` ở bất cứ đâu bạn muốn nó hiển thị.

```
1  function App(props) {
2    //Object chứa thông tin của người iwww :))
3    const myGirlFriends = [
4      {
5        id: 1,
6        name: "Khanh Huyen",
7        email: "khanhhuyen123@freetuts.net",
8      },
9      {
10       id: 2,
11       name: "Nguyen Hang",
12       email: "nguyenhang3dzas@freetuts.net",
13     },
14     {
15       id: 3,
16       name: "Pham Uyen",
17       email: "phamuyenz@freetuts.net",
18     },
19   ];
20
21   return (
22     <ShowGirlFriends listGirlFriends={myGirlFriends}>
23       {(data) => {
24         //Nhận data từ component ShowGirlFriends khi nó trả về
25         //bằng đoạn props.children(person)
```



```

25         console.log(data);
26     }}
27     </ShowGirlFriends>
28 );
29 }
30

```

Lúc này chúng ta sẽ thấy kết quả, các đoạn mã html khi được render sẽ như sau:

```

1     <ul>
2         <li>Khanh Huyen</li>
3         <li>Nguyen Hang</li>
4         <li>Pham Uyen</li>
5     </ul>

```

và đồng thời bạn sẽ nhận được giá trị của mỗi người được trả về thông qua props `children`:

```

[HMR] Waiting for update signal from WDS...
▼ {id: 1, name: "Khanh Huyen", email: "khanhhuyen123@freetuts.net"} ⓘ
  email: "khanhhuyen123@freetuts.net"
  id: 1
  name: "Khanh Huyen"
  ► __proto__: Object
▼ {id: 2, name: "Nguyen Hang", email: "nguyenhang3dzas@freetuts.net"} ⓘ
  email: "nguyenhang3dzas@freetuts.net"
  id: 2
  name: "Nguyen Hang"
  ► __proto__: Object
▼ {id: 3, name: "Pham Uyen", email: "phamuyenz@freetuts.net"} ⓘ
  email: "phamuyenz@freetuts.net"
  id: 3
  name: "Pham Uyen"
  ► __proto__: Object
> |

```

xlii. Ví dụ 2

Ở ví dụ tiếp theo này chúng ta sẽ đi xây dựng bộ đếm, bao gồm nút tăng giảm..

Trước tiên, chúng ta sẽ đi xây dựng một component có tên `Counter`, có nhiệm vụ xây dựng hàm tăng, giảm và hiển thị số.

```

1    class Counter extends React.Component {
2      constructor(props) {
3        super(props);
4        this.state = {
5          count: 0,
6        };
7        //Bind this
8        this.increment = this.increment.bind(this)
9        this.decrement = this.decrement.bind(this)
10     }
11     //Hàm này sẽ TĂNG giá trị của số
12     increment() {
13       this.setState({
14         count: this.state.count + 1,
15       });
16     }
17     //Hàm này sẽ GIẢM giá trị của số
18     decrement() {
19       this.setState({
20         count: this.state.count - 1,
21       });
22     }
23     render() {
24       return <div>{
25         //Trả về giá trị cho props render.
26         this.props.render({
27           count: this.state.count,
28           increment: this.increment,

```

```

29         decrement: this.decrement
30     })
31 }</div>;
32 }
33 }

```

Ở đây chúng ta sẽ trả về giá trị của số, và hàm thực hiện tăng giảm số vào props có tên `render`.

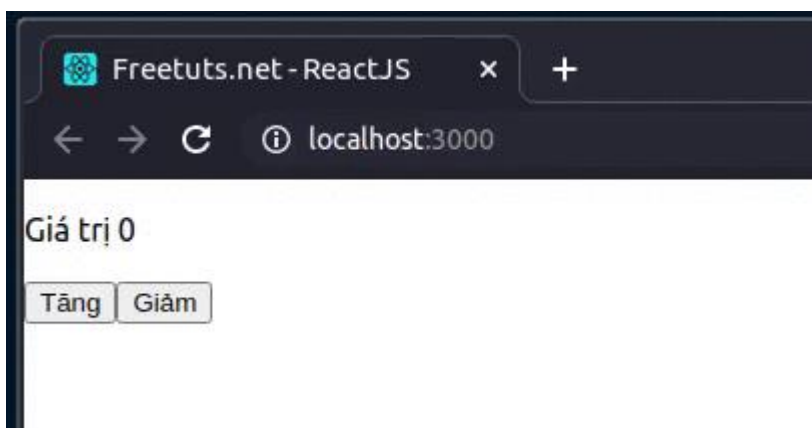
Tiếp theo, mình sẽ gọi component này ở bất cứ đâu, props có tên `render` được truyền vào sẽ có giá trị là một function và tham số của function đó sẽ là giá trị trả về.

```

1  default function App(props) {
2      return (
3          <Counter
4              render={(data) => {
5                  //Nhận giá trị trả về từ Counter
6                  //qua props render.
7                  const { count, increment, decrement } = data;
8                  return (
9                      <>
10                         <p>Giá trị {count}</p>
11                         <button onClick={increment}>Tăng</button>
12                         <button onClick={decrement}>Giảm</button>
13                     </>
14                 );
15             }}
16          />
17      );
18  }

```

Lúc này mình sẽ hiển thị count và các nút tăng giảm, khi bạn click vào nút tăng giảm nó sẽ gọi hàm tương ứng. Và đây là kết quả :



19. Higher-Order Components trong ReactJS

Trong bài viết này ta sẽ tìm hiểu về Higher-Order Components trong React JS, đây là tính năng rất hay trong việc tái sử dụng tài nguyên.

Trong một dự án ReactJS việc tái sử dụng các component cũng như phân chia các component sao cho hợp lý là điều tất yếu. Khi các component được nhóm với nhau thành các phần riêng biệt, tái sử dụng các component sẽ giúp quá trình duy trì dự án được dễ dàng hơn sau này. Chúng ta có phương pháp đó là **Higher-Order Components** cho phép làm những việc này một cách đơn giản.

Table of Content

- [1. Higher-Order Components là gì ?](#)
- [2. Triển khai ví dụ](#)

45. Higher-Order Components là gì ?

Higher-Order Components(HOC) theo định nghĩa nó là một function nhận vào một component và trả về một component. **Higher-Order Components** không phải là một tính năng trong React hoặc bất kỳ ngôn ngữ lập trình nào khác, mà là một phương pháp phát triển component.

Để hiểu rõ hơn tại sao phải sử dụng nó thì ta sẽ làm một ví dụ đơn giản như sau:

Ở đây mình xây dựng 1 component có chức năng hiển thị ra một ảnh và mỗi khi di con trỏ chuột vào thì ảnh sẽ bị mờ đi.

```
1    import React from "react";
2
3    const Image = (props) => {
4      return ;
5    };
6
7    export default class HoverComponent extends React.Component {
8      constructor(props) {
9        super(props);
```

```

10     this.state = {
11         opacity: 1,
12     };
13     //bind this
14     this.onMouseLeave = this.onMouseLeave.bind(this);
15     this.onMouseEnter = this.onMouseEnter.bind(this);
16 }
17 //Được gọi khi chuột được di vào
18 onMouseEnter() {
19     this.setState({
20         opacity: 0.5,
21     });
22 }
23 //Được gọi khi chuột được rời đi
24 onMouseLeave() {
25     this.setState({
26         opacity: 1,
27     });
28 }
29 render() {
30     return (
31         <div
32             style={{ opacity: this.state.opacity }}
33             onMouseEnter={this.onMouseEnter}
34             onMouseLeave={this.onMouseLeave}
35         >
36             <Image />
37         </div>

```

```

38         );
39     }
40 }

```

Trong chương trình trên, khi muốn hiển thị nhiều ảnh thì phải xây dựng rất nhiều `HoverComponent` chỉ nhằm 1 mục đích làm mờ ảnh. Điều này là không cần thiết. Vậy làm sao để làm mờ nhiều ảnh mà không cần phải viết lại `HoverComponent` cho mỗi ảnh. Chúng ta sẽ đi tìm cách giải quyết cho bài toán này ở phần ví dụ nhé.

46. Triển khai ví dụ

Ta sẽ đi giải quyết bài toán ở đầu bài cho, lúc này sẽ cần sử dụng **Higher Order Componets(HOC)** có chức năng làm mờ ảnh. Cùng bắt đầu xây dựng một HOC có tên `withHoverOpacity` :

```

1    //Đây được gọi là một HOC, nó nhận vào 1 component
2    //và trả ra một component
3    const withHoverOpacity = (ImageComponent) => {
4        return class extends React.Component {
5            constructor(props) {
6                super(props);
7                this.state = {
8                    opacity: 1,
9                };
10           //bind this
11           this.onMouseLeave = this.onMouseLeave.bind(this);
12           this.onMouseEnter = this.onMouseEnter.bind(this);
13       }
14       //Được gọi khi chuột được di vào
15       onMouseEnter() {
16           this.setState({
17               opacity: 0.5,
18           });

```

```

19     }
20     //Được gọi khi chuột được rời đi
21     onMouseLeave() {
22         this.setState({
23             opacity: 1,
24         });
25     }
26     render() {
27         return (
28             <div
29                 style={{ opacity: this.state.opacity }}
30                 onMouseEnter={this.onMouseEnter}
31                 onMouseLeave={this.onMouseLeave}
32             >
33                 <ImageComponent />
34             </div>
35         );
36     }
37 };
38 };

```

Một HOC nhận vào một component và trả về một component, bên trên mình đã xây dựng thành công một HOC. Tiếp theo, để sử dụng nó bạn chỉ cần gọi nó.

```

1     //Các component là các ảnh cần Hover
2     const Image1 = (props) => {
3         return ;
4     };
5     const Image2 = (props) => {
6         return (

```



```

7      
11 );
12 };
13
14 //Lúc này mình truyền vào HOC một component
15 //và mình sẽ nhận vào một component mới
16
17 //Ở đây mình có thể hiển thị rất nhiều ảnh
18 // mà không cần phải xây dựng component hỗ trợ việc làm
19 //mờ ảnh quá nhiều
20 const ImageWithHoverOpacity1 = withHoverOpacity(Image1);
21 const ImageWithHoverOpacity2 = withHoverOpacity(Image2);

```

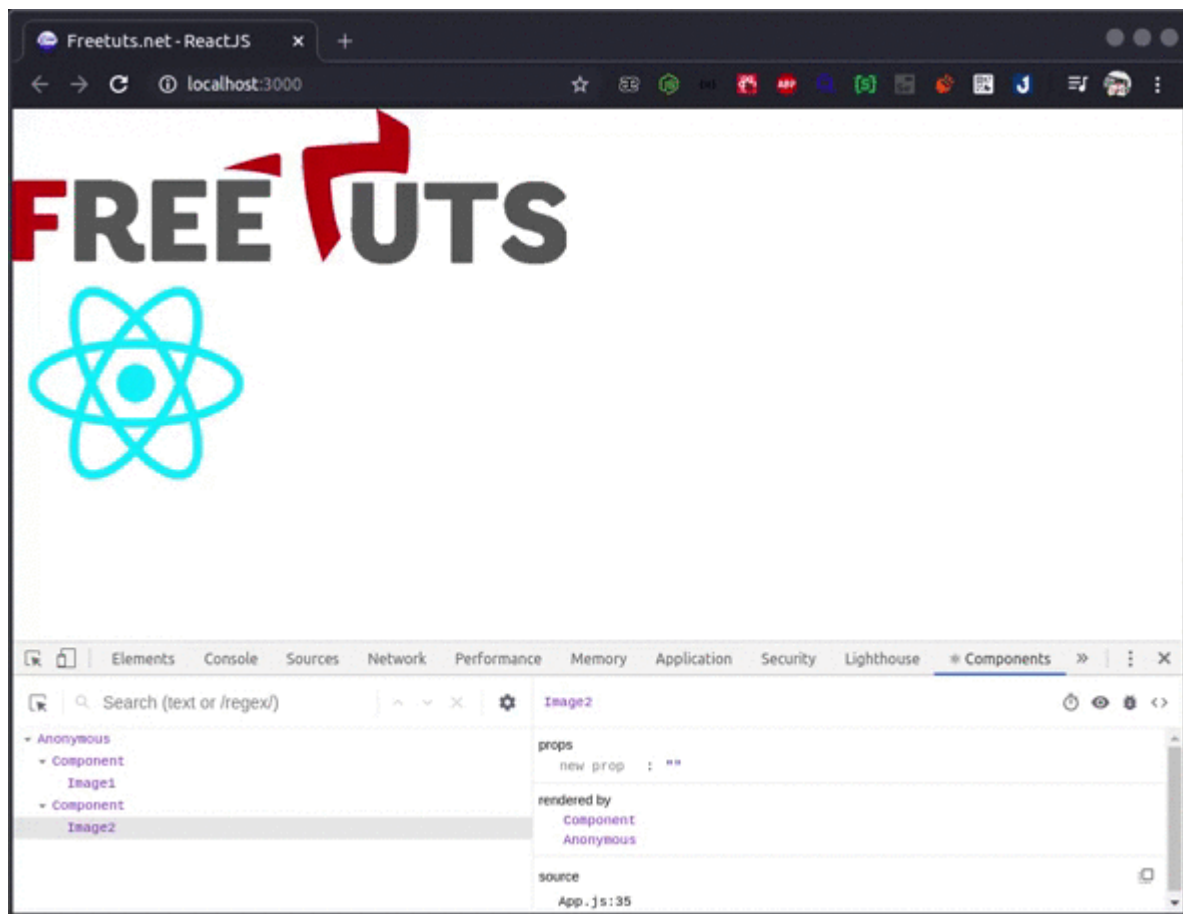
Lúc này component `ImageWithHoverOpacity1`, `ImageWithHoverOpacity2` là 2 component đã được thêm hiệu ứng làm mờ ảnh bằng HOC. Bây giờ, ta chỉ cần render nó ra:

```

1 //Hiển thị component
2 export default function () {
3     return (
4         <>
5             <ImageWithHoverOpacity1 />
6             <ImageWithHoverOpacity2 />
7         </>
8     );
9 }

```

Lúc này ta sẽ nhận được kết quả gần giống như đầu bài, nhưng các đoạn mã sẽ được tái sử dụng nhiều hơn. Khởi chạy ứng dụng và đây là kết quả:



20. Giới thiệu Hooks trong React JS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về **Hook trong React**, và giới thiệu các hooks hay được sử dụng trong quá trình làm việc với React.

Khi làm việc với các React Component chúng ta cần phải thao tác rất nhiều với state, props hay life cycle. Và kể từ phiên bản 16.8 trở đi React cung cấp một chức năng mới đó là **React Hooks**, chức năng này cho phép thay thế việc sử dụng state thông thường bằng các khái niệm mới như `useState`, `useEffect`..

Việc sử dụng React Hooks cho phép sử dụng state, hay các tính năng khác của React mà không cần phải viết một class component dài dòng.

Table of Content

- 1. React Hooks là gì?
- 2. Hooks trong React JS cơ bản
 - `useState()`
 - `useEffect()`
 - `useContext()`
 - `useReducer()`

47. React Hooks là gì?



Chúng ta có thể hiểu **React Hooks** là một chức năng được xây dựng trong React cho phép chúng ta có thể sử dụng state và **life cycle** bên trong một functional components. **Hooks** đem lại một vài lợi ích khi làm việc như :

- Cải thiện hiệu suất làm việc bằng cách có thể tái sử dụng code.
- Các thành phần được trình bày khoa học hơn.
- Sử dụng một cách linh hoạt trong component tree.

React Hooks đem lại cho functional **components** các tính năng cần thiết của component, nó có thể thay thế gần như hoàn toàn việc sử dụng class components. Ở đây mình có ví dụ để chứng minh điều đó.

Giả sử khi mình muốn xây dựng một component cho phép random các số nguyên từ 1 - 100. Khi chúng ta viết bằng class component thông thường sử dụng **state** thì các đoạn mã khá dài dòng:

```
1    import React, { Component } from 'react';
2
3    export default class RandomNumberComponent extends Component {
4      constructor(props) {
5        super(props)
6        //Khởi tạo state
7        this.state = {
8          number: 0
9        }
10       this.randomNumber = this.randomNumber.bind(this)
11     }
12     randomNumber = () => {
13       const number = Math.round(Math.random() * 100)
14       //Cập nhật state mới
15       this.setState({
16         number
17       })
18     }
```

```

19     render() {
20         return (
21             <div style = {{padding: '10%'}}>
22                 <b>{this.state.number}</b> <hr />
23                 <button onClick={this.randomNumber}>Random</button>
24             </div>
25         );
26     }
27 }

```

Nhưng khi chúng ta sử dụng **React Hooks** (cụ thể ở ví dụ bên dưới là sử dụng hook `useState`) sẽ nhanh hơn rất nhiều.

```

1     import React, { useState } from "react";
2
3     export default function RandomNumberComponent(props) {
4         const [number, setNumber] = useState(0)
5
6         return (
7             <div style={{ padding: "10%" }}>
8                 <b>{number}</b> <hr />
9                 <button onClick={() => {
10                     setNumber(Math.round(Math.random() * 100))
11                 }}>Random</button>
12             </div>
13         );
14     }

```

Hai cách viết trên đều có một chức năng giống nhau nhưng khi chúng ta sử dụng **React Hooks** sẽ giúp giảm các đoạn mã và tài nguyên.

48. Hooks trong React JS cơ bản

Chúng ta có 10 hooks được xây dựng trong phiên bản React từ 16.8 trở đi. Nhưng trong bài này mình sẽ chỉ ra các hooks cơ bản hay được sử dụng bao gồm:

- `useState()`
- `useEffect()`
- `useContext()`
- `useReducer()`

Ở đây có 4 hooks cơ bản hay được sử dụng, bây giờ chúng ta sẽ đi tìm hiểu cơ bản nhất về các hooks này. Mình sẽ giới thiệu chi tiết cũng như ví dụ về từng hooks ở các bài viết tiếp theo.

xliii. `useState()`

Việc sử dụng `useState()` cho phép chúng ta có thể làm việc với state bên trong functional component mà không cần chuyển nó về class component. Ở ví dụ bên trên mình cũng đã sử dụng `useState()` để cập nhật state. Chúng ta có thể sử dụng nó bằng cú pháp:

```
1 const [tenSate, hamCapNhatState] = useState(giaTriBanDauCuaState);
```

Đây là một hooks được sử dụng hầu như trong tất cả các functional component.

xliv. `useEffect()`

`useEffect()` là function nắm bắt tất cả các sự thay đổi của code. Trong một function component, việc sử dụng life cycle không React hỗ trợ, bởi vậy rất khó để debug, cũng như nắm bắt được quá trình khởi chạy của component.

`useEffect()` sinh ra để làm điều này, nó được khởi chạy khi giá trị của một biến nào đó thay đổi, hay component đã được render ra,...`useEffect()` có thể thay thế hoàn toàn các life cycle trong class component. Chúng ta có thể sử dụng nó bằng cú pháp :

```
1 useEffect(functionDuocKhoiChay, arrayChuaCacGiaTriThayDoi)
```

xlvi. `useContext()`

`useContext()` cho phép nhận về giá trị của context mỗi khi nó thay đổi. Bạn có thể tham khảo bài viết về [React Context](#) để hiểu rõ hơn.

```
1 const giaTriCuaContext = useContext(TenContext);
```

xlvi. useReducer()

Hook `useReducer` được sử dụng để xử lý các state phức tạp và việc chia sẻ state giữa các component. Ở đây chúng ta có cú pháp.

```
1  const [state, dispatch] = useReducer(reducer, initialArg, init);
```

Tất cả các hooks mà được giới thiệu ở trên, sẽ được giới thiệu chi tiết và ví dụ cụ thể ở loạt bài viết tiếp theo.

21. Tìm hiểu React Hook useState

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về **React Hook useState**. Từ phiên bản 16.8 trở đi React cho ra mắt một tính năng mới đó là React Hooks, nó cho phép chúng ta làm việc với state, life cycle, và hàng loạt tính năng khác trong một functional component.

React Hooks bao gồm rất 10 hooks khác nhau, trong phạm vi bài viết này chúng ta sẽ tìm hiểu và xây dựng các ví dụ cụ thể bằng một hooks quan trọng và được sử dụng rất nhiều đó là **useState**.

Table of Content

- [1. useState trong React](#)
- [2. Xây dựng ví dụ](#)
 - [Ẩn/Hiện một component](#)
 - [Lấy dữ liệu từ API và hiển thị](#)

49. useState trong React

useState là một hook cho phép chúng ta quản lý các state trong một functional component, Bằng cách gọi **React.useState** bên trong một functional component, bạn đã có thể làm việc với state một cách nhanh chóng.

Để sử dụng **useState**, trước tiên chúng ta cần import nó vào component.

```
import React, { useState } from "react";
```

Tiếp theo, bạn chỉ cần sử dụng **useState** bằng cú pháp:

```
1 const [nameState, funcUpdate] = React.useState(defaultState)
```

Ở đây chúng ta có 3 đối số cần chú ý đến:

- **nameState**: đây là giá trị mặc định của state.
- **funcUpdate**: function dùng để cập nhật state. Giả sử mình muốn cập nhật giá trị của state mình chỉ cần gọi **funcUpdate('giá trị mới của state')**.
- **defaultState**: giá trị mặc định của state khi được khởi tạo lần đầu.

Khi bạn sử dụng state trong một class component, state được khởi tạo đó luôn luôn là một object, chúng ta chỉ có thể lưu trữ giá trị trong object đó.

```
1 // Khởi tạo một state trong
2 // class component
3 // Giá trị luôn là một object
4 this.state = {
5   website: 'freetuts.net'
6 }
```

Nhưng với hooks, chúng ta có thể lưu trữ bất cứ kiểu dữ liệu nào trong state như object, number, string,... Khi bạn thực hiện gọi `useState` đồng nghĩa với nó là state đã được khởi tạo thành công.

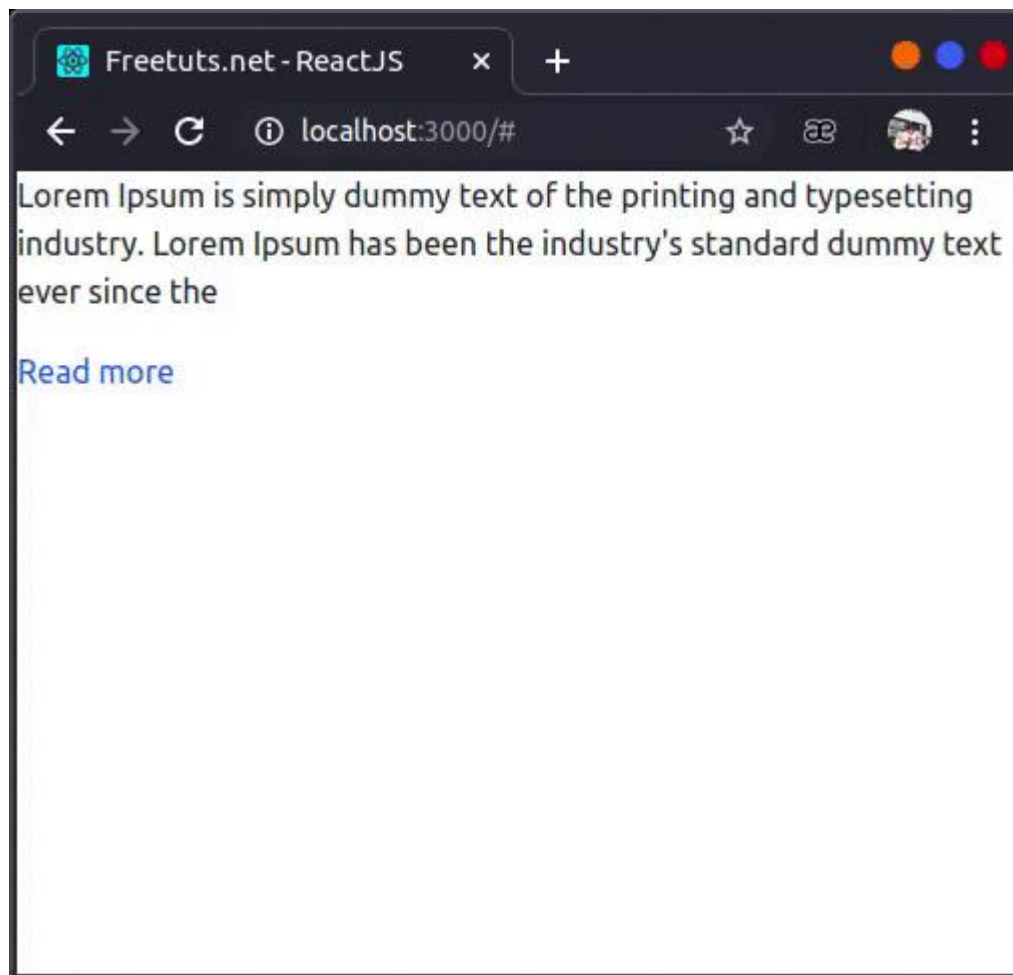
```
1 // Khởi tạo state trong functional component
2 // sử dụng React hook: useState
3
4 // Khởi tạo state có kiểu dữ liệu là string
5 const [website, setWebsite] = React.useState('freetuts.net');
6
7 // Kiểu dữ liệu là boolean
8 const [isLoading, setLoading] = React.useState(false);
9
10 .....
```

50. Xây dựng ví dụ

Chúng ta sẽ cùng nhau đi xây dựng ví dụ cụ thể, ở đây chúng ta sẽ từng bước đi xây dựng các ví dụ đơn giản nhất.

xlvi. Ẩn/Hiện một component

Một ví dụ cho phép ẩn/hiện các đoạn văn bản bằng cách click vào "đọc thêm".



Để triển khai ví dụ chúng ta sẽ thực hiện từng bước như sau:

```
1 //Gọi React và useState
2 import React, { useState } from "react";
3
4 //Component hiển thị thêm nội dung
5 const MoreContent = () => {
6   return (
7     <p>
8
9       1500s, when an unknown printer took a galley of type and scrambled it to
10      make a type specimen book. It has survived not only five centuries, but
```

```

11     also the leap into electronic typesetting, remaining essentially
12     unchanged. It was popularised in the 1960s with the release of Letraset
13     sheets containing Lorem Ipsum passages, and more recently with desktop
14     publishing software like Aldus PageMaker including versions of Lorem
15     Ipsum.
16   </p>
17   );
18 };
19
20 export default function App(props) {
21   // Sử dụng useState
22   // isShow là một state
23   // setShow là function giúp cập nhật state
24   // Giá trị mặc định ban đầu của state là false
25   const [isShow, setShow] = useState(false);
26   return (
27     <div>
28       <p>Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lo
29
30       {isShow === true ? <MoreContent /> : ''}
31       {isShow === false ? <a href="/#" onClick={() => {
32         // Khi click vào button
33         // cập nhật state bằng cách gọi hàm
34         // setShow đã được khai báo bên trên/
35         setShow(true)
36       }}>Read more</a> : ''}
37     </div>
38   );

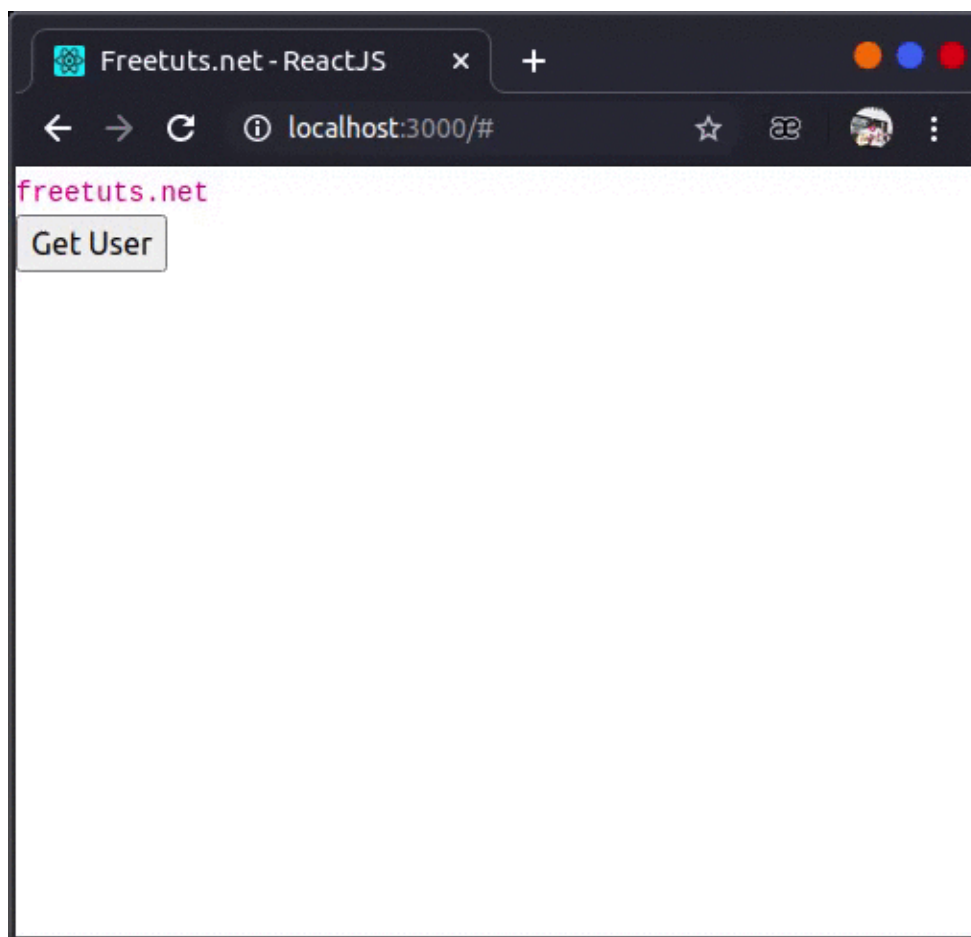
```

xlvi. Lấy dữ liệu từ API và hiển thị

Để tiến hành triển khai ví dụ chúng ta cần tham khảo thêm các bài viết về:

- [Higher-Order Components trong ReactJS.](#)
- [Tìm hiểu về Fragments trong ReactJS.](#)
- [Tìm hiểu về List và Keys trong ReactJS](#)

Sau khi đã hiểu hết về các phần kiến thức cần sử dụng, chúng ta sẽ tiến hành xây dựng ứng dụng cho phép lấy dữ liệu người dùng về từ API và hiển thị ra trình duyệt.



Trước tiên, chúng ta cần cài thêm một module cho phép gọi api có tên `axios` bằng cách sử dụng dòng lệnh:

```
1 npm i axios
```

Bạn có thể tham khảo thêm về module này bằng cách truy cập vào trang tài liệu chính thức của nó. Tiếp theo, chúng ta sẽ tạo một component và import các thư viện cần thiết vào:

```
1 //Gọi React và useState
2 import React, { useState } from "react";
3 import axios from "axios"; //Sử dụng axios
```

Khởi tạo component giúp hiển thị danh sách người dùng:

```
1 //Component hiển thị danh sách người dùng
2 const ShowUser = (props) => {
3   //Lấy giá trị của props listUser
4   const { listUser } = props;
5
6   // Render ra list user
7   // React.Fragment cho phép bọc JSX lại.
8   // List Keys : chỉ định key, giúp loại bỏ cảnh báo.
9   return (
10     <div>
11       {listUser.map((user, index) => {
12         return (
13           <React.Fragment key={user.id}>
14             <ul>
15               <li>{user.name}</li>
16               <li>{user.email}</li>
17             </ul>
18             <hr />
19           </React.Fragment>
20         );
21       })}
```

```

22     </div>
23   );
24 };

```

Sau đó khởi tạo component gọi API và hiển thị giao diện.

```

1   export default function App(props) {
2     //Khai báo state, sử dụng hook: useState
3     const [listUser, setListUser] = useState([]);
4     const [isLoading, setLoading] = useState(false);
5
6     //API chứa dữ liệu người dùng
7     const getUserAPI =
8       "https://5df8a4c6e9f79e0014b6a587.mockapi.io/freetuts/users";
9
10    //Hàm fetch API để lấy dữ liệu ng. dùng
11    const getUser = () => {
12      //Cập nhật lại giá trị của state loading
13      setLoading(true);
14
15      //Thực hiện gọi api
16      axios
17        .get(getUserAPI)
18        .then((res) => {
19          //Cập nhật giá trị của state listUser
20          setListUser(res.data);
21        })
22        .catch((err) => {
23          //Trường hợp xảy ra lỗi
24          alert("Không thể kết nối tới server");
25        });
26    };
27  }

```

```

24         })
25         .finally(() => {
26             // Câu lệnh trong này được khởi chạy mỗi khi call API xong
27             // bất kể thất bại hay không.
28             // ...
29             setLoading(false); //Cập nhật giá trị của state isLoading
30         });
31     };
32
33     return (
34         <>
35         <code>freetuts.net</code> <br />
36         {isLoading ? "loading..." : <button onClick={getUser}>Get User</button>}
37         <ShowUser listUser={listUser} />
38         </>
39     );
40 }
41

```

Ở đây mình viết luôn vào file `src/App.js` của dự án nên sau khi gộp các phần ở trên, chúng ta sẽ có file `App.js` hoàn chỉnh:

```

1     //Gọi React và useState
2     import React, { useState } from "react";
3     import axios from "axios"; //Sử dụng axios
4
5     //Component hiển thị danh sách người dùng
6     const ShowUser = (props) => {
7         //Lấy giá trị của props listUser
8         const { listUser } = props;

```

```

9
10    // Render ra list user
11    // React.Fragment cho phép bọc JSX lại.
12    // List Keys : chỉ định key, giúp loại bỏ cảnh báo.
13    return (
14        <div>
15            {listUser.map((user, index) => {
16                return (
17                    <React.Fragment key={user.id}>
18                        <ul>
19                            <li>{user.name}</li>
20                            <li>{user.email}</li>
21                        </ul>
22                        <hr />
23                    </React.Fragment>
24                );
25            })}
26        </div>
27    );
28 };
29
30 export default function App(props) {
31     //Khai báo state, sử dụng hook: useState
32     const [listUser, setListUser] = useState([]);
33     const [isLoading, setLoading] = useState(false);
34
35     //API chứa dữ liệu người dùng
36     const getUserAPI =

```



```

37     "<a href='\"https://5df8a4c6e9f79e0014b6a587.mockapi.io/freetuts/users\"'>https://5df8a4c6e9f79e0014b6a587.mockapi.io/freetuts/users</a>";
38
39     //Hàm fetch API để lấy dữ liệu ng. dùng
40     const getUser = () => {
41         //Cập nhật lại giá trị của state loading
42         setLoading(true);
43
44         //Thực hiện gọi api
45         axios
46             .get(getUserAPI)
47             .then((res) => {
48                 //Cập nhật giá trị của state listUser
49                 setListUser(res.data);
50             })
51             .catch((err) => {
52                 //Trường hợp xảy ra lỗi
53                 alert("Không thể kết nối tới server");
54             })
55             .finally(() => {
56                 // Câu lệnh trong này được khởi chạy mỗi khi call API xong
57                 // bất kể thất bại hay không.
58                 // ...
59                 setLoading(false); //Cập nhật giá trị của state isLoading
60             });
61     };
62
63     return (
64         <>

```

```
65         <code>freetuts.net</code> <br />
66         {isLoading ? "loading..." : <button onClick={getUser}>Get User</button>}
67         <ShowUser listUser={listUser} />
68     </>
69 );
70 }
```

Khởi chạy dự án bằng cách gõ dòng lệnh:

```
1 npm start
```

22. useEffect trong React Hooks

Trong bài viết này chúng ta sẽ đi tìm hiểu về **useEffect trong React Hooks**. Ở các bài trước chúng ta đã cùng nhau đi tìm hiểu về `useState` hooks. Bài viết này sẽ tiếp nối series bằng một hook cũng khá quan trọng đó là **useEffect**, nó cho phép chúng ta làm việc với life cycle trong functional component.

Table of Content

- 1. useEffect trong React Hooks
- 2. Sử dụng useEffect
 - Sử dụng useEffect như `componentDidMount`
 - Sử dụng useEffect như `componentDidUpdate`
 - Sử dụng useEffect như `componentWillUnmount`

51. useEffect trong React Hooks

`useEffect` là một hook trong React Hooks cho phép chúng ta làm việc với các **life cycle** ở functional component. Có thể hiểu đơn giản rằng `useEffect Hook` là của 3 phương thức `componentDidMount`, `componentDidUpdate`, và `componentWillUnmount` kết hợp lại với nhau.

`Lifecycle` là một phần rất quan trọng trong một component. Trong một vài trường hợp chúng ta cần phải fetch data từ API khi component đã được render, hay thực hiện hành động nào đó khi một component được update. Bởi vậy có thể thấy rằng phương thức quan trọng và hay được sử dụng nhất trong lifecycle đó là `componentDidMount`, `componentDidUpdate`.

Nhưng trong một functional component không thể làm việc với các life cycle này bằng cách thông thường, bởi vậy `useEffect` Hooks sinh ra để làm điều này.

useEffect cho phép chúng ta xử lý các logic trong các vòng đời của component và được gọi mỗi khi có bất cứ sự thay đổi nào trong một componnet.

52. Sử dụng useEffect

Chúng ta sẽ đi vào tìm hiểu sâu hơn về **useEffect** bằng cách đi qua các ví dụ cụ thể. Để sử dụng `useEffect` bạn cần phải import nó vào trong component cần sử dụng:

```
1 import React, { useEffect } from 'react'
```

và sử dụng nó bằng cú pháp:

```
1    useEffect(effectFunction, arrayDependencies)
```

trong đó `arrayDependencies` là một mảng các giá trị, khi giá trị trong mảng này thay đổi thì hàm `effectFunction` sẽ được gọi. Trước khi đi tiếp vào phần sau bạn có thể tham khảo thêm bài viết về [Tìm hiểu về Life cycle trong ReactJS](#).

xlix. Sử dụng `useEffect` như `componentDidMount`

Chúng ta có thể bắt sự kiện `componentDidMount` trong một functional component bằng cách sử dụng `useEffect` và chỉ định `arrayDependencies` là một mảng rỗng:

```
1    useEffect(effectFunction, [])
```

Nó thường được sử dụng để gọi API khi component đã được render. Ở đây mình có một ví dụ về gọi API.

```
1    //Gọi React và useState
2    import React, { useState, useEffect } from "react";
3    import axios from "axios"; //Sử dụng axios
4
5    //Component hiển thị danh sách người dùng
6    const ShowUser = (props) => {
7        //Lấy giá trị của props listUser
8        const { listUser } = props;
9
10       // Render ra list user
11       // React.Fragment cho phép bọc JSX lại.
12       // List Keys : chỉ định key, giúp loại bỏ cảnh báo.
13       return (
14           <div>
15               {listUser.map((user, index) => {
16                   return (
17                       <React.Fragment key={user.id}>
```

```

18         <ul>
19             <li>{user.name}</li>
20             <li>{user.email}</li>
21         </ul>
22         <hr />
23     </React.Fragment>
24     );
25     })}
26 </div>
27 );
28 };
29
30 export default function App(props) {
31     //Khai báo state, sử dụng hook: useState
32     const [listUser, setListUser] = useState([]);
33
34     //Sử dụng useEffect hook như componentDidMount
35     useEffect(() => {
36         const getUserAPI = 'https://5df8a4c6e9f79e0014b6a587.mockapi.io/freetuts/users';
37
38         //Gọi API bằng axios
39         axios.get(getUserAPI).then((res) => {
40             // Cập nhật lại listUser bằng
41             // Bạn có thể xem lại bài viết về useState()
42             setListUser(res.data);
43         }).catch((err) => {
44             //Trường hợp xảy ra lỗi
45             console.log(err);

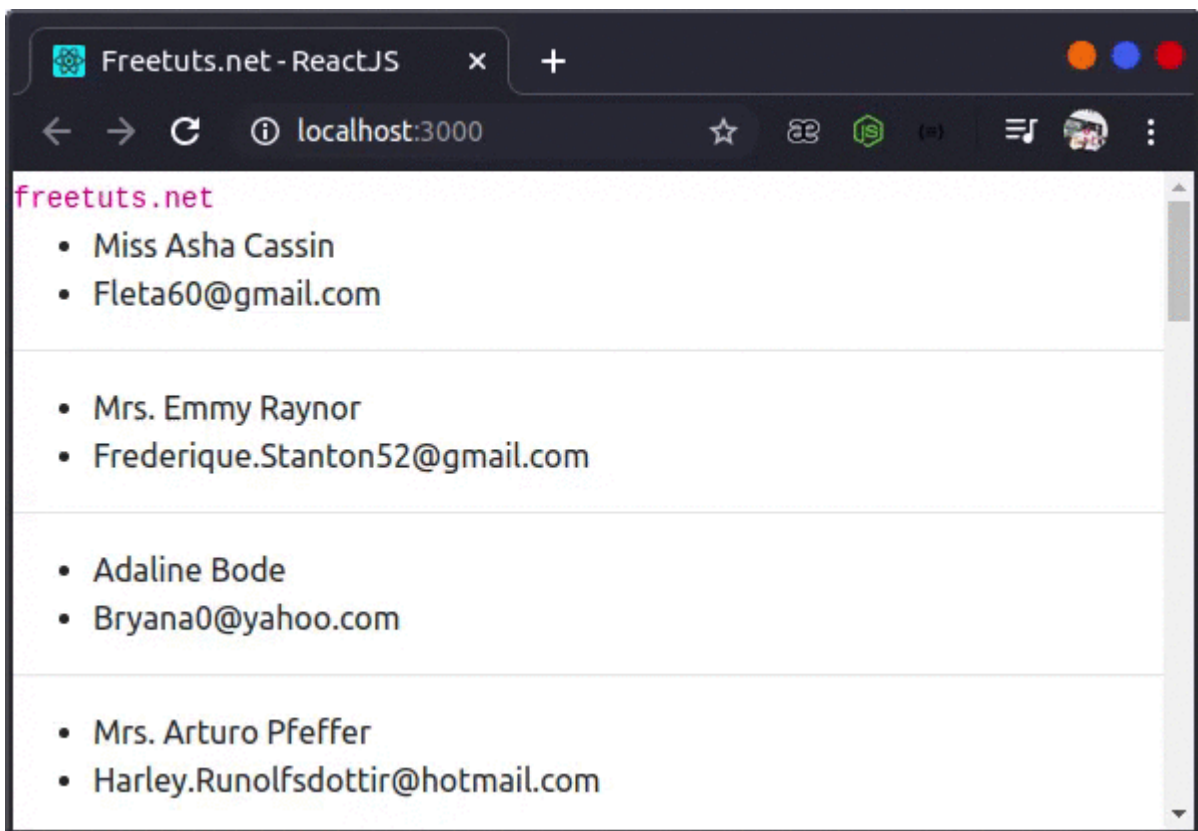
```

```

46         alert("Xảy ra lỗi");
47     })
48
49     }, [])
50
51     return (
52         <>
53             <code>freetuts.net</code> <br />
54
55             <ShowUser listUser={listUser} />
56         </>
57     );
58 }

```

chúng ta sẽ nhận được kết quả như hình:



1. Sử dụng *useEffect* như *componentDidUpdate*

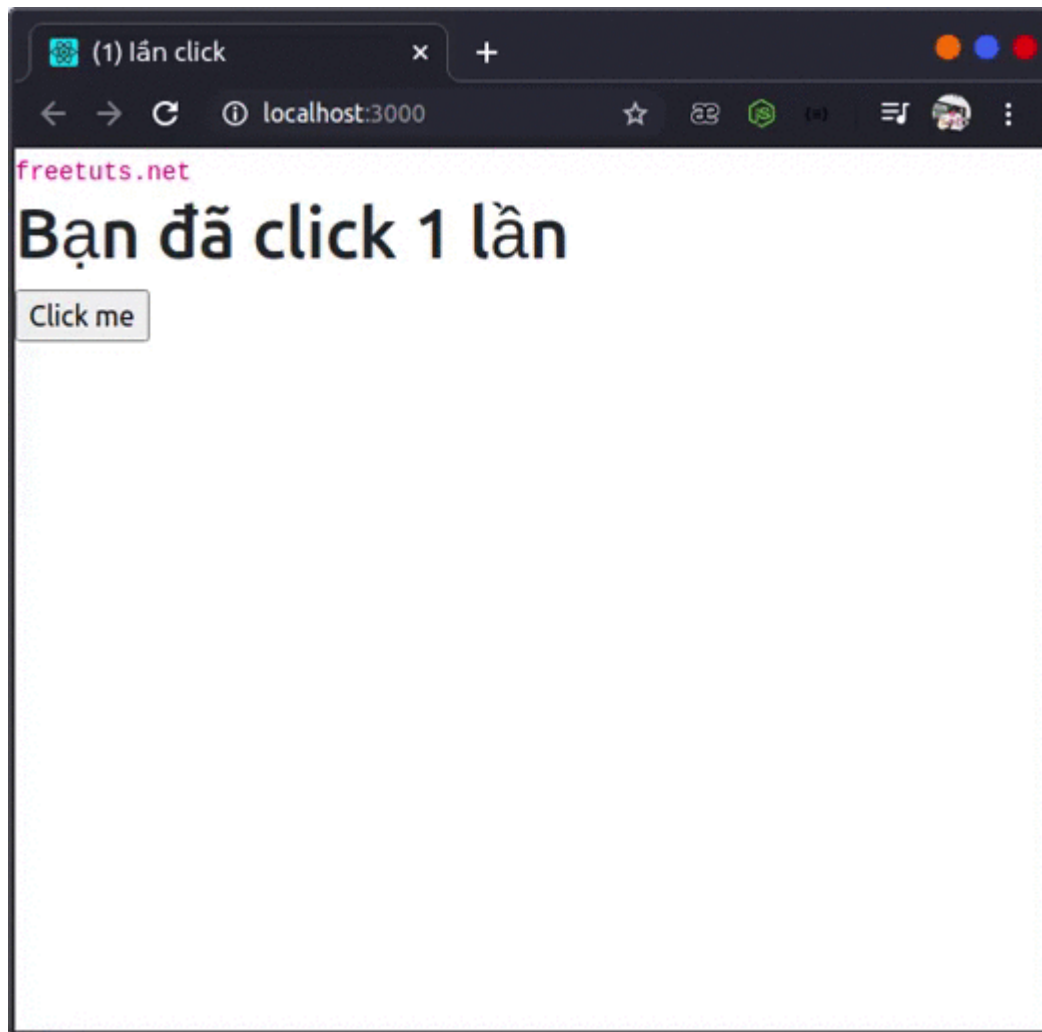
Để sử dụng `componentDidUpdate` trong functional component chúng ta sẽ chỉ định giá trị của tham số `arrayDependencies` là `null`.

```
1    useEffect(effectFunction)
```

Ở đây chúng ta sẽ xây dựng ví dụ về thay đổi tiêu đề mỗi khi component thực hiện update.

```
1    //Gọi React và useState
2    import React, { useState, useEffect } from "react";
3
4
5    export default function App() {
6      const [count, setCount] = useState(1);
7      //Sử dụng useEffect() như componentDidUpdate
8      useEffect(() => {
9        document.title = `${count}) lần click`
10      })
11      return (
12        <>
13          <code>freetuts.net</code> <br />
14          <h1>Bạn đã click {count} lần</h1>
15          <button onClick = {() => {
16            setCount(count+1)
17          }}>Click me</button>
18        </>
19      );
20    }
```

Và đây là kết quả.



li. Sử dụng `useEffect` như `componentUnWillMount`

Lifecycle thứ 3 mà chúng ta có thể sử dụng được với `useEffect` là `componentUnWillMount`, lifecycle này được khởi chạy khi mỗi khi component chuẩn bị được xóa khỏi tree dom. Để sử dụng nó bằng `useEffect` chúng ta chỉ cần return về 1 function trong `effectFunction`, function được return đó sẽ đóng vai trò như là `componentUnWillMount`. Cùng xem xét ví dụ bên dưới:

```
1   import React, { useEffect, useState } from 'react';
2
3
4   function LifecycleDemo() {
5
6     useEffect(() => {
```

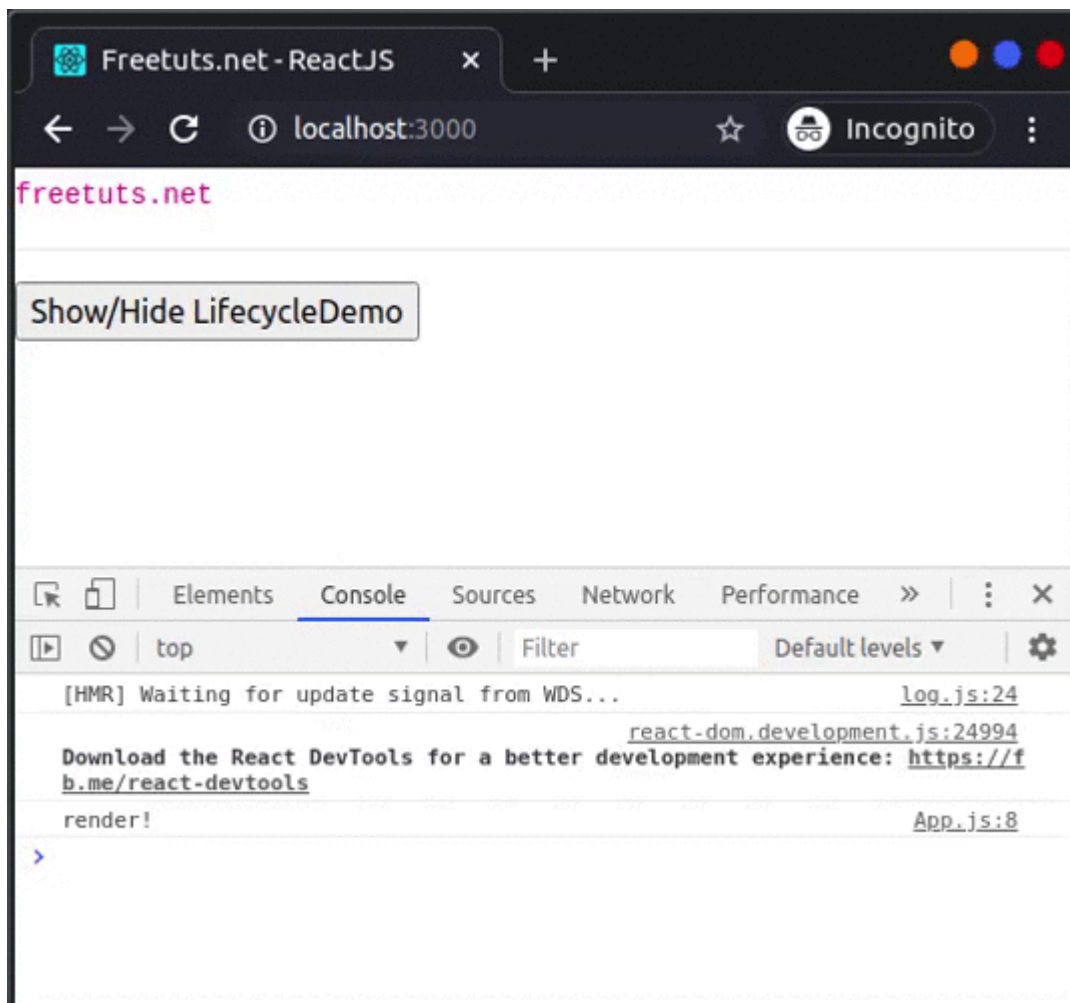


```

7      //Được gọi khi component render
8      console.log('render!');
9      // useEffect trả về một hàm ,
10     // hàm trả về đó là đóng vai trò như
11     // là componentWillUnmount
12     return () => console.log('unmounting...');
13 })
14
15     return (<code>freetuts.net</code>);
16 }
17
18 export default function App() {
19     const [mounted, setMounted] = useState(true);
20     const toggle = () => setMounted(!mounted);
21     return (
22         <>
23             {mounted && <LifecycleDemo/>} <hr />
24             <button onClick={toggle}>Show/Hide LifecycleDemo</button>
25         </>
26     );
27 }

```

mở **Dev Tools -> Console** lên và bạn sẽ thấy sự thay đổi ở mỗi lần component bị xóa bỏ khỏi tree dom.



23. useContext trong React Hook

Trong bài viết này chúng ta sẽ cùng nhau **đi tìm hiểu về useContext trong React Hooks**. Đây là bài viết tiếp nối các bài viết về React Hooks trước đó. Để hiểu rõ hơn về các khái niệm trong bài viết này chúng ta có thể tham khảo thêm về bài viết [React Context API](#).

Table of Content

- [1. useContext là gì ?](#)
- [2. Ví dụ về useContext trong React](#)

53. useContext là gì ?

useContext là một hooks trong React Hooks cho phép chúng ta có thể làm việc với React Context trong một functional component. Giả sử khi chúng ta muốn lấy giá trị của context trong class component:

```
1  class MyClass extends React.Component {
2    render() {
3      //Lấy giá trị của context
4      const value = this.context;
5    }
6  }
7  MyClass.contextType = MyContext;
```

Bạn cũng có thể lấy giá trị của context trong functional componetn bằng cách sử dụng **useContext**.

```
1  import AppContext from './appContext.js';
2
3  const MyClass = () => {
4    //Lấy giá trị của context
5    const value = useContext(AppContext);
6    return (
7      ...
8    );
9  };
```

```
9    }
```

Để sử dụng `useContext` chúng ta sẽ truyền vào hooks này một tham số duy nhất đó là `Context Object` (được tạo bởi `React.createContext`).

```
1    const AppContext = React.createContext({ foo: 'bar' });
```

khi cần lấy giá trị của context trong functional component chúng ta sẽ sử dụng:

```
1    const context = useContext(AppContext);
```

Phần tiếp theo chúng ta sẽ đi vào xây dựng các ví dụ liên quan đến `useContext` dựa vào các kiến thức đã học ở các phần trước.

54. Ví dụ về useContext trong React

Ở đây chúng ta sẽ đi xây dựng một bộ đếm sử dụng React Context, sử dụng hook `useContext` để lấy giá trị của context.

Ví dụ này chúng ta sẽ viết trực tiếp vào file `src/App.js`, ở đây chúng ta sẽ tiến hành khởi tạo một Context.

```
1    const CounterContext = React.createContext();
```

Tiếp theo, chúng ta sẽ bọc các component cần chia sẻ state bằng `Provider`, và truyền giá vào props `value` giá trị của context.

```
1    export default class App extends Component {
2      constructor(props) {
3        super(props);
4        this.state = {
5          count: 1,
6        };
7      }
8      //Cập nhật giá trị của state count - Tăng
9      countUp() {
10        this.setState({
11          count: this.state.count + 1,
```

```

12     });
13 }
14     //Cập nhật giá trị của state count - Giảm
15     countDown() {
16         this.setState({
17             count: this.state.count - 1,
18         });
19     }
20     render() {
21         return (
22             <CounterContext.Provider
23                 value={{
24                     count: this.state.count,
25                     countUp: this.countUp.bind(this),
26                     countDown: this.countDown.bind(this),
27                 }}
28             >
29                 <Counter />
30             </CounterContext.Provider>
31         );
32     }
33 }

```

Bên trên chúng ta đã gán giá trị của **Context** là một object bao gồm **count**, và 2 hàm cho phép tăng giảm giá trị của count. Đồng thời, bên trong **Provider** chúng ta đã gọi component **Counter**.

```

1     const Counter = () => {
2         // Sử dụng useContext để lấy giá trị của context
3         // Tham số truyền vào là object CounterContext.
4         const counter = useContext(CounterContext);

```

```

5
6     return (
7         <div>
8             <h1>{counter.count}</h1>
9             <button
10                onClick={() => {
11                    counter.countUp();
12                }}
13            >
14                Tăng
15            </button>
16            <button
17                onClick={() => {
18                    counter.countDown();
19                }}
20            >
21                Giảm
22            </button>
23        </div>
24    );
25 };

```

Componet **Counter** có nhiệm vụ hiển thị giá trị của context, và gọi các hành động liên quan như tăng giảm. Ở trên, chúng ta đã sử dụng **useContext** để lấy giá trị của **CouterContext**. Khi một hành động tương ứng được thực thi thì giá trị của context sẽ được thay đổi.

Cuối cùng, chúng ta sẽ có file **App.js** hoàn chỉnh như sau:

```

1     import React, { Component, useContext } from "react";
2     // Khởi tạo context mới.
3     const CounterContext = React.createContext();

```

```

4    const Counter = () => {
5      // Sử dụng useContext để lấy giá trị của context
6      // Tham số truyền vào là object CounterContext.
7      const counter = useContext(CounterContext);
8
9      return (
10         <div>
11           <code>freetuts.net</code>
12           <h1>{counter.count}</h1>
13           <button
14             onClick={() => {
15               counter.countUp();
16             }}
17           >
18             Tăng
19           </button>
20           <button
21             onClick={() => {
22               counter.countDown();
23             }}
24           >
25             Giảm
26           </button>
27         </div>
28       );
29     };
30     export default class App extends Component {
31       constructor(props) {

```

```

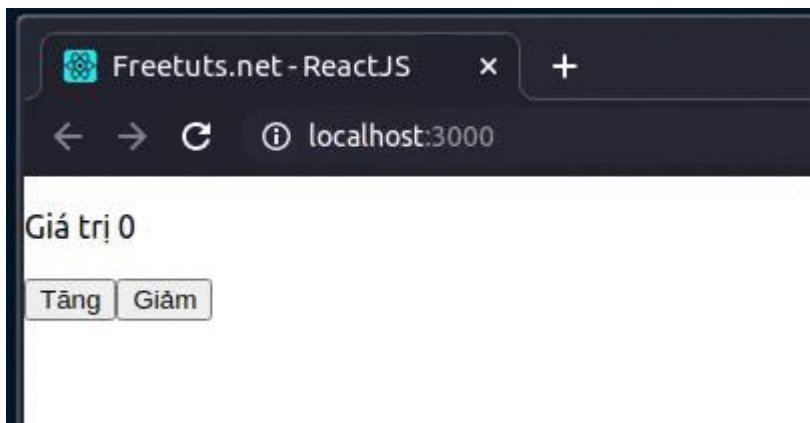
32     super(props);
33     this.state = {
34         count: 1,
35     };
36 }
37 //Cập nhật giá trị của state count - Tăng
38 countUp() {
39     this.setState({
40         count: this.state.count + 1,
41     });
42 }
43 //Cập nhật giá trị của state count - Giảm
44 countDown() {
45     this.setState({
46         count: this.state.count - 1,
47     });
48 }
49 render() {
50     return (
51         <CounterContext.Provider
52             value={{
53                 count: this.state.count,
54                 countUp: this.countUp.bind(this),
55                 countDown: this.countDown.bind(this),
56             }}
57         >
58         <Counter />
59     </CounterContext.Provider>

```



```
60      );  
61    }  
62  }
```

Khởi chạy project chúng ta sẽ nhận được kết quả như hình.



Chúng ta có thể tách các đoạn code thành các file riêng biệt để dễ dàng quản lí hơn sau này. Ví dụ bên trên khá quen thuộc và đơn giản. Chủ yếu dựa trên kiến thức về React Context, bạn nên xem lại bài viết về Context API trong ReactJS để có thể hiểu rõ hơn về phần này.

24. Xây dựng Hook trong React JS (React Custom Hook)

Bài viết này sẽ hướng dẫn cách để tự tay đi xây dựng một hooks - custom hooks trong React JS.

React Hooks là một tính năng mới trong React 16.8. Cho phép sử dụng state và các tính năng khác trong React mà không cần viết một class component. Ngoài sử dụng các hooks mặc định như `useState`, `useEffect`,... chúng ta còn có thể tự tạo một hooks tùy theo chức năng của nó.

Table of Content

- 1. Custom Hooks là gì?
- 2. Khi nào dùng Custom Hooks
- 3. Tự xây dựng một custom hook

55. Custom Hooks là gì?



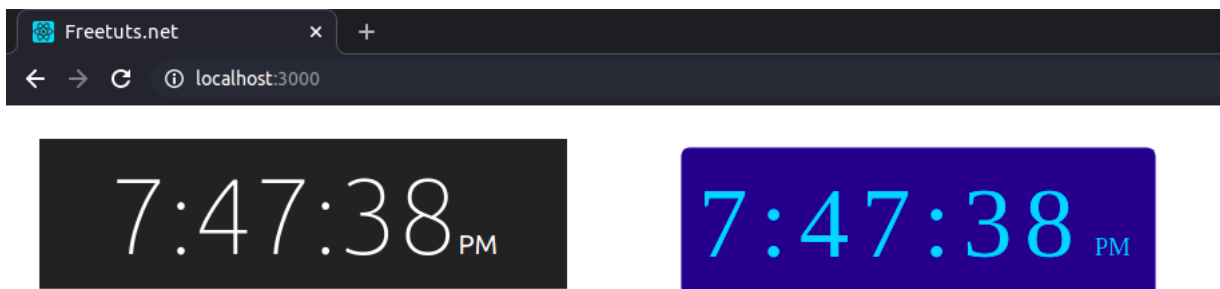
Custom Hooks là những hooks mà do lập trình viên tự định nghĩa với mục đích thực hiện một chức năng nào đó, nó thường được sử dụng để chia sẻ logic giữa các components.

React cũng định nghĩa cho chúng ta các hooks như `useState`, `useEffect`, `useContext`,... cho phép chúng ta làm việc dễ dàng hơn. Để tự định nghĩa một hooks cho riêng mình, chúng ta chỉ cần xây dựng 1 hàm nhận và trả về các giá trị.

Khi đặt tên một custom hooks phải có từ khóa `use` ở đầu, ví dụ như: `useClick()`, `useClock()`, `useQuery()`,...

56. Khi nào dùng Custom Hooks

Custom Hooks rất hay được sử dụng trong quá trình triển khai ứng dụng. Trong trường hợp chúng ta muốn tách biệt các phần xử lý logic riêng ra khỏi UI, hay chia sẻ logic giữa các component. Giả sử trong trường hợp muốn xây dựng 2 components đều có 1 chức năng là hiển thị ngày giờ hiện tại nhưng khác nhau ở giao diện như bên dưới.



Thông thường, chúng ta sẽ viết tất cả các logic xử lý giờ ở trong component như này.

```
1   import React, { useState } from "react";
2   import "./Clock1.css";
3   function Clock1() {
4     const [time, setTime] = useState("");
5     const [ampm, setampm] = useState("");
6     const updateTime = function () {
7       let dateInfo = new Date();
8       let hour = 0;
9       /* Lấy giá trị của giờ */
10      if (dateInfo.getHours() === 0) {
11        hour = 12;
12      } else if (dateInfo.getHours() > 12) {
```

```

13         hour = dateInfo.getHours() - 12;
14     } else {
15         hour = dateInfo.getHours();
16     }
17     /* Lấy giá trị của phút */
18     let minutes =
19         dateInfo.getMinutes() < 10
20         ? parseInt("0" + dateInfo.getMinutes())
21         : dateInfo.getMinutes();
22
23     /* Lấy giá trị của giây */
24     let seconds =
25         dateInfo.getSeconds() < 10
26         ? "0" + dateInfo.getSeconds()
27         : dateInfo.getSeconds();
28
29     /* Định dạng ngày */
30     let ampm = dateInfo.getHours() >= 12 ? "PM" : "AM";
31
32     /* Cập nhật state */
33     setampm(ampm)
34     setTime(`${hour}:${minutes}:${seconds}`);
35 };
36 setInterval(function () {
37     updateTime();
38 }, 1000);
39 return (
40     <div className="time">

```

```

41         <span className="hms">{time}</span>
42         <span className="ampm">{ampm}</span>
43     </div>
44 );
45 }
46
47 export default Clock1;

```

Rồi lại lặp lại logic này ở **Clock2**, có thể nhận thấy cách viết này không tối ưu một chút nào. Giả sử, dự án có 10 cái đồng hồ thì cần phải lặp lại logic này ở mỗi components. Điều này là không cần thiết và tốn thời gian. Giải pháp ở đây là sử dụng custom hooks.

57. Tự xây dựng một custom hook

Vậy làm sao để tự mình xây dựng một custom hooks, rất đơn giản chỉ cần tách phần xử lý logic ra một function.

Chúng ta sẽ có một custom hooks thay cho cách viết dài dòng ở ví dụ trên. Trong thư mục `src` tạo một thư mục có tên `hooks`, thư mục này sẽ chứa tất cả các *custom hooks*. Đây là một hooks có tên `useClock()` có nhiệm vụ trả về thời gian hiện tại.

```

1    // src/hooks/useClock.js
2    import { useState } from "react";
3
4
5    export default function useClock() {
6        const [time, setTime] = useState("");
7        const [ampm, setampm] = useState("");
8
9        // Function cập nhật thời gian.
10       const updateTime = function () {
11           let dateInfo = new Date();
12           let hour = 0;

```

```

13      /* Lấy giá trị của giờ */
14      if (dateInfo.getHours() === 0) {
15          hour = 12;
16      } else if (dateInfo.getHours() > 12) {
17          hour = dateInfo.getHours() - 12;
18      } else {
19          hour = dateInfo.getHours();
20      }
21      /* Lấy giá trị của phút */
22      let minutes =
23          dateInfo.getMinutes() < 10
24              ? parseInt("0" + dateInfo.getMinutes())
25              : dateInfo.getMinutes();
26
27      /* Lấy giá trị của giây */
28      let seconds =
29          dateInfo.getSeconds() < 10
30              ? "0" + dateInfo.getSeconds()
31              : dateInfo.getSeconds();
32
33      /* Định dạng ngày */
34      let ampm = dateInfo.getHours() >= 12 ? "PM" : "AM";
35
36      /* Cập nhật state */
37      setampm(ampm)
38      setTime(`${hour}:${minutes}:${seconds}`);
39  };
40

```

```

41         setInterval(function () {
42             updateTime();
43         }, 1000);
44
45         return [time, ampm]
46     }

```

Khi muốn sử dụng hooks này chỉ cần import nó vào component và gọi như các hooks thông thường. React dự vào tên để xem đây là một hooks bởi vậy nên đặt tên đúng định dạng là **use + nameHooks**.

```

// src/components/Clock2.js

import React from 'react'

import './Clock2.css';

//Import Hooks

import useClock from '../hooks/useClock'

function Clock2() {

    //Gọi custom hook để sử dụng

    const [time, ampm] = useClock()

    return (

```

```
<div id="MyClockDisplay" className="clock">

  {time}

  <span>{ampm}</span>

</div>

)

}

export default Clock2
```

Vậy là chúng ta có thể sử dụng logic trong nhiều component khác nhau mà không cần lặp lại các đoạn logic phức tạp, chỉ cần import và gọi là có thể sử dụng. Với cộng đồng sử dụng rộng lớn thì ngoài các hooks có sẵn trong React, còn có các custom hooks do cộng đồng đóng góp. Có thể nói ReactJS là một thư viện khá mạnh và hữu ích cho việc lập trình front-end.

25. Redux là gì? Tại sao lại ứng dụng trong ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về Redux trong ReactJS.

Chắc hẳn trong quá trình làm việc với ReactJS thì việc quản lí các state giữa các component là vấn đề khá phức tạp. Mặc dù nhà phát triển của React cũng đã cho ra mắt [React Context](#) cho phép chúng ta làm điều này, nhưng nó có một vài hạn chế nhất định. Bởi vậy, chúng ta sẽ thay thế React Context bằng một thư viện mới có tên **Redux** với nhiều ưu điểm cũng như quản lí dễ dàng hơn.

Ở trong bài viết này chúng ta sẽ đi giới thiệu về Redux cũng như các phần liên quan đến nó một cách chi tiết. **Redux** cũng là phần khá "đau não" với các bạn mới làm quen với React. Vậy nên, cùng đi qua các phần căn bản nhất nhé.

Table of Content

- [1. Cần hiểu trước khi tìm hiểu Redux là gì](#)
 - [State là gì?](#)
 - [Props là gì?](#)
- [2. Vậy Redux là gì?](#)
 - [Tại sao chúng ta phải sử dụng Redux?](#)
 - [Redux hoạt động như thế nào?](#)

58. Cần hiểu trước khi tìm hiểu Redux là gì

lùi. State là gì?

Để trả lời cho câu hỏi Redux là gì? Trước tiên, chúng ta sẽ đi tìm hiểu về khái niệm state trong React.

State được định nghĩa là một object có thể được sử dụng để chứa dữ liệu hoặc thông tin về components. Trong một React Component, state chỉ tồn tại trong phạm vi của components chứa nó, mỗi khi state thay đổi thì components đó sẽ được render lại.

Đây là một state được khởi tạo trong React component.

```
1    import React, { Component } from 'react';  
2  
3    export default class App extends Component {
```

```

4      constructor(props) {
5          super(props)
6          //Khởi tạo giá trị ban đầu của state
7          this.state = {
8              website: 'freetuts.net',
9              title  : 'Freetuts.net'
10         }
11     }
12     render() {
13         return (
14             ...
15         );
16     }
17 }

```

Khi chúng ta muốn thay đổi giá trị của state bạn chỉ cần cập nhật giá trị mới cho state đó bằng cách gọi hàm `setState()`.

```

1    //Cập nhật giá trị của state
2    this.setState({
3        website: 'freetuts.net'
4    })

```

Vì phạm vi của state chỉ tồn tại trong một component, trong một vài trường hợp chúng ta muốn chia sẻ state cho nhiều component cùng một lúc thì khá phức tạp. Bởi vậy **Redux** ra đời để làm điều này, nó giúp bạn chia sẻ state tới nhiều components.

liii. Props là gì?

Props là một object được truyền vào trong một components, mỗi components sẽ nhận vào props và trả về react element. Props cho phép chúng ta giao tiếp giữa các components với nhau bằng cách truyền tham số qua lại giữa các components.

```

1    const App = () => <Welcome name="Freetuts"></Welcome>

```

Trên đây là cú pháp truyền props giữa 2 component, cụ thể trong ví dụ trên là từ `App` sang `Welcome`. Đây là phần khá đơn giản nên chúng ta chỉ nhắc lại một chút.

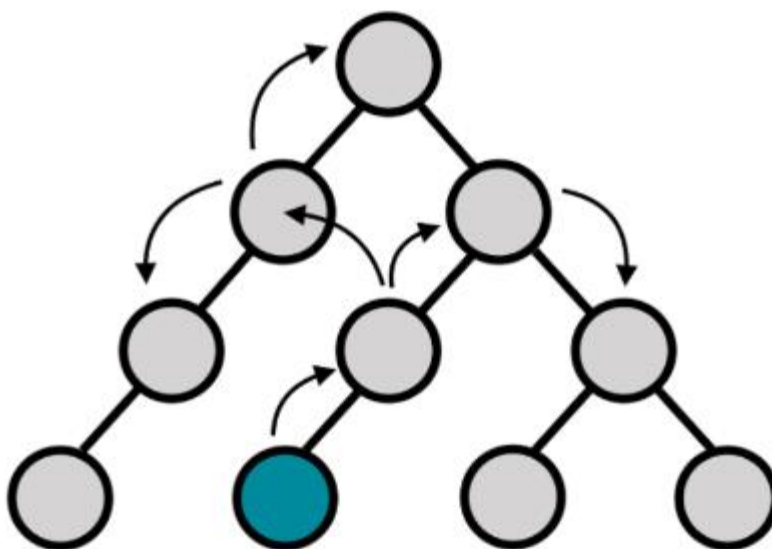
59. Vậy Redux là gì?

Redux là một thư viện cho phép chúng ta quản lý state trong một ứng dụng javascript. Nếu kết hợp nó trong React JS thì sẽ như hổ mọc thêm cánh.

liv. Tại sao chúng ta phải sử dụng Redux?

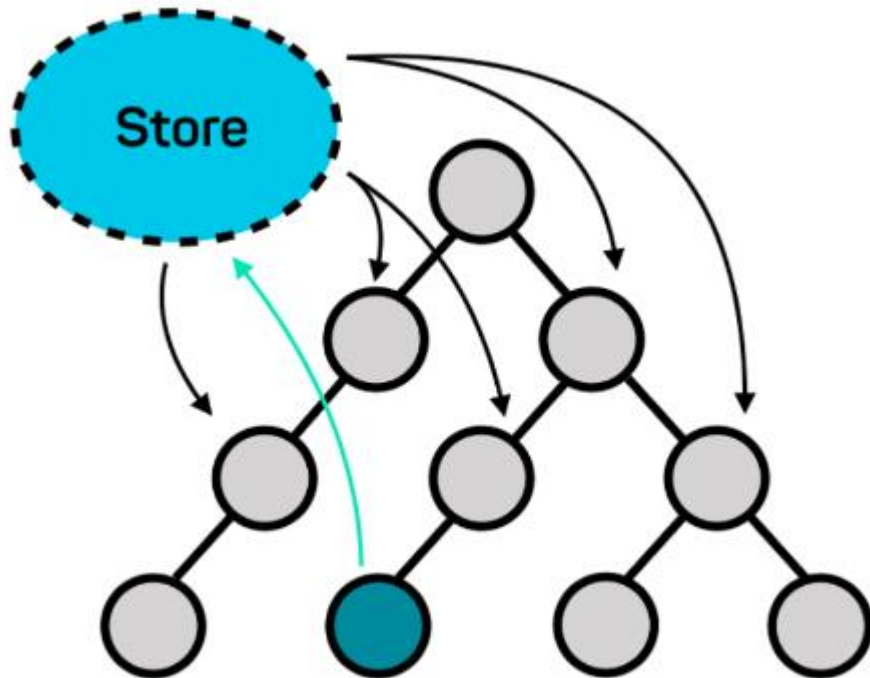
Redux sẽ giải quyết một bài toán khá là quan trọng đó là chia sẻ state. Như đã đề cập ở trên thì việc chia sẻ state giữa các component với nhau theo các thức truyền qua props là khá phức tạp và rắc rối.

Giả sử khi chúng ta muốn truyền dữ liệu từ component A sang component C thì bắt buộc phải thông qua component B.



Như hình bên trên khi chúng ta cần chia sẻ dữ liệu giữa các component với nhau bằng cách sử dụng props thì bắt buộc phải thông qua các component trung gian. Điều này khá phức tạp và dễ gây nhầm lẫn.

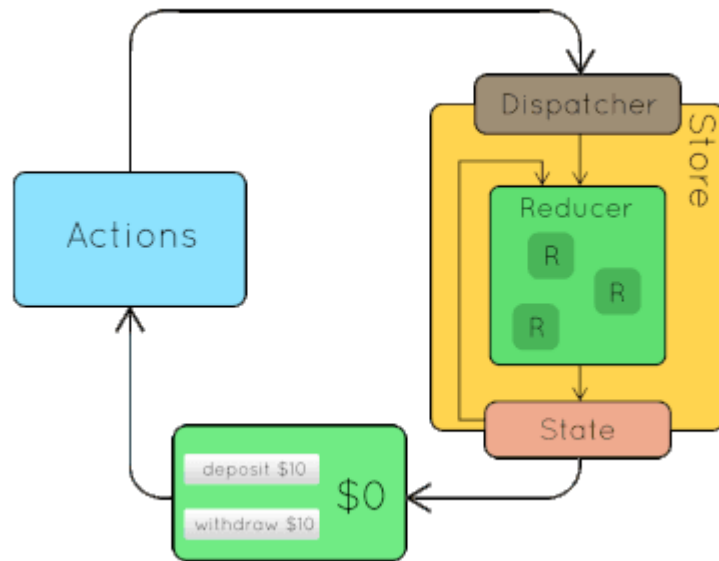
Để có thể truyền dữ liệu một cách tối ưu và đơn giản hơn chúng ta sẽ lưu dữ liệu vào một store, từ đó cấp phát dữ liệu cho các component cần thiết. Lúc này store sẽ đóng vai trò trung gian, nó có nhiệm vụ chứa và phân phát dữ liệu. Cùng xem hình minh họa bên dưới.



Giả sử bạn muốn chia sẻ dữ liệu từ component A tới component C thì chỉ cần đẩy state vào Store và Store sẽ cấp phát dữ liệu cho component C. Đây là mô hình mà Redux sử dụng, qua đó việc kiểm soát dữ liệu sẽ dễ dàng và tối ưu hơn.

lv. Redux hoạt động như thế nào?

Store được coi là phần quan trọng nhất trong **Redux**, nó có nhiệm vụ lưu trữ và phân phát dữ liệu cho các component. Trong store bao gồm các thành phần như *dispatcher* (có nhiệm vụ kích hoạt các action bên trong reducer), *reducer* có nhiệm vụ xử lý các hành động được gửi đến. Sau đây là mô hình cách thức hoạt động của **Redux**.



Sau khi một *action* được thực thi, *dispatcher* sẽ được kích hoạt và gửi đến *reducer* một *action*. Lúc này *reducer* thực hiện hành động dựa vào *action* được gửi đến. Sau đó, đồng thời lưu lại giá trị của *state* mới vào trong *store* và trả về *state* mới đó.

Giả sử ở đây mình có 1 đoạn mã thực hiện tăng giảm giá trị của state thông qua *redux*.

```

1   import redux from 'redux';
2   //Reducer
3   const counter = (state = 0, action) => {
4       //Kiểm tra điều kiện
5       switch (action.type) {
6           case 'INCREMENT':
7               return state + 1;
8       }
9       return state;
10  };
11
12  //Đây là store
13  const store = redux.createStore(counter);

```

14

15 //Thực hiện dispatch

16 store.dispatch({type : 'INCREMENT'})

Ở ví dụ trên khi dispatch được thực thi thì lúc này nó sẽ gửi đến cho reducer một action có *type* là *INCREMENT*, *reducer* kiểm tra *action* và tiến hành tăng giá trị của state và trả về state mới.

26. React Router cơ bản

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về **React Router trong ReactJS**. Việc sử dụng React Router cho phép chúng ta có thể tùy biến URL trong một dự án ReactJS.

Table of Content

- [1. React Router là gì ?](#)
- [2. Sử dụng React Router trong ReactJS](#)
- [3. Xây dựng ví dụ](#)

60. React Router là gì ?

SPA (single page application) hiện nay được coi là một xu thế để xây dựng một trang web bởi nhiều tính năng ưu việt, có rất nhiều thư viện cho phép xây dựng một trang SPA phổ biến nhất đó là ReactJS. Khi một trang web được xây dựng theo hướng SPA thì tất cả các UI của trang web sẽ được render ra một trang duy nhất, tùy vào từng trường hợp mà component sẽ được render.

Ngoài ra, chúng ta có thể sử dụng URL làm điều kiện xem xét rằng liệu component nào sẽ được render. Trong ReactJS, React Router là thư viện được xây dựng để thực hiện điều này.

React Router là một thư viện cho việc điều hướng URL tiêu chuẩn trong React, Nó cho phép chúng ta có thể đồng bộ UI với URL. Được thiết kế với API đơn giản, từ đó cho phép giải quyết các vấn đề về URL một cách nhanh chóng.

61. Sử dụng React Router trong ReactJS

Để sử dụng React Router chúng ta cần phải cài đặt thư viện này vào trong dự án React bằng cách sử dụng NPM:

```
1 npm install react-router-dom
```

Sau khi cài đặt thành công, trong trường hợp cần dùng đến React Router bạn chỉ cần import nó component đó.

```
1 import { BrowserRouter, Route, Switch } from 'react-router-dom';
```

Để hiểu rõ hơn về React Router chúng ta sẽ đi xây dựng một ví dụ cụ thể về nó.

62. Xây dựng ví dụ

Trong ví dụ này, chúng ta sẽ đi xây dựng một trang landing page với nhiều trang khác nhau. Cấu trúc thư mục của `src` sẽ như sau:

```
src/  
  
---components/  
  
-----About.js  
  
-----Home.js  
  
-----Shop.js  
  
---index.js  
  
---App.js  
  
...more...
```

Trước tiên, chúng ta cần phải thiết lập app sử dụng **React Router**. Mọi thứ sẽ được render cần phải được bọc bên trong `BrowserRouter`, chúng ta sẽ lựa chọn component `App` bởi nó chính là component xử lý logic mặc định trong ReactJS. Trong file `index.js` của dự án chúng ta sẽ chỉnh sửa lại như sau:

```
1    // index.js  
2    ...  
3    import { BrowserRouter } from 'react-router-dom';  
4    ...  
5    ReactDOM.render(  
6      <BrowserRouter>  
7        <App/>  
8      </BrowserRouter>,  
9      document.getElementById('root')
```



```
10    );
```

Tiếp theo đó ở file `src/App.js` chúng ta cần phải sử dụng `Switch` để bọc các `Router` lại. Đây là điều bắt buộc, tất các các `Route` cần phải được bọc bởi `Switch`.

```
1    ...
2    // Ở đây chúng ta import 4 component được xây dựng trong thư mục src/components
3    // đó là Home, About, Shop, Error
4    import React from 'react'
5    import { Route, Switch } from 'react-router-dom';
6
7    import Home from './components/Home';
8    import About from './components/About';
9    import Shop from './components/Shop'
10   import Error from './components/Error'
11
12
13   export default function App() {
14       return (
15           <>
16               <Switch>
17                   <Route path="/" component={Home} exact />
18                   <Route path="/about" component={About} />
19                   <Route path="/shop" component={Shop} />
20                   <Route component={Error} />
21               </Switch>
22           </>
23       )
24   }
```

Route có nhiệm vụ render component theo path được chỉ định. Trong trường hợp ở trên **Route** có path là `/` có thêm một props nữa là **exact** bởi hầu hết các path đều thông qua `/`. Khi một route không có thuộc tính **path** thì render component khi URL không tồn tại.

Bên trên trong file **App.js** chúng ta đã import 3 component đó là **Home**, **About**, **Shop** , **Error** trong thư mục **src/components**, bây giờ chúng ta cần phải xây dựng 3 componet đó.

```
1 //file: components/Home.js
2
3 import React from "react";
4 import { Link } from 'react-router-dom';
5
6 export default function Home() {
7   return (
8     <div>
9       <h1> Home Page</h1>
10      <Link to="/about">About / </Link>
11      <Link to="/shop">Shop / </Link>
12      <Link to="/404">404 / </Link>
13    </div>
14  );
15 };
```

```
1 //file: components/About.js
2
3 import React from "react";
4 import { Link } from 'react-router-dom';
5
6 export default function About() {
7   return (
8     <div>
```

```

9      <h1> About Page</h1>
10      <Link to="/">Home Page </Link>
11  </div>
12  );
13  };

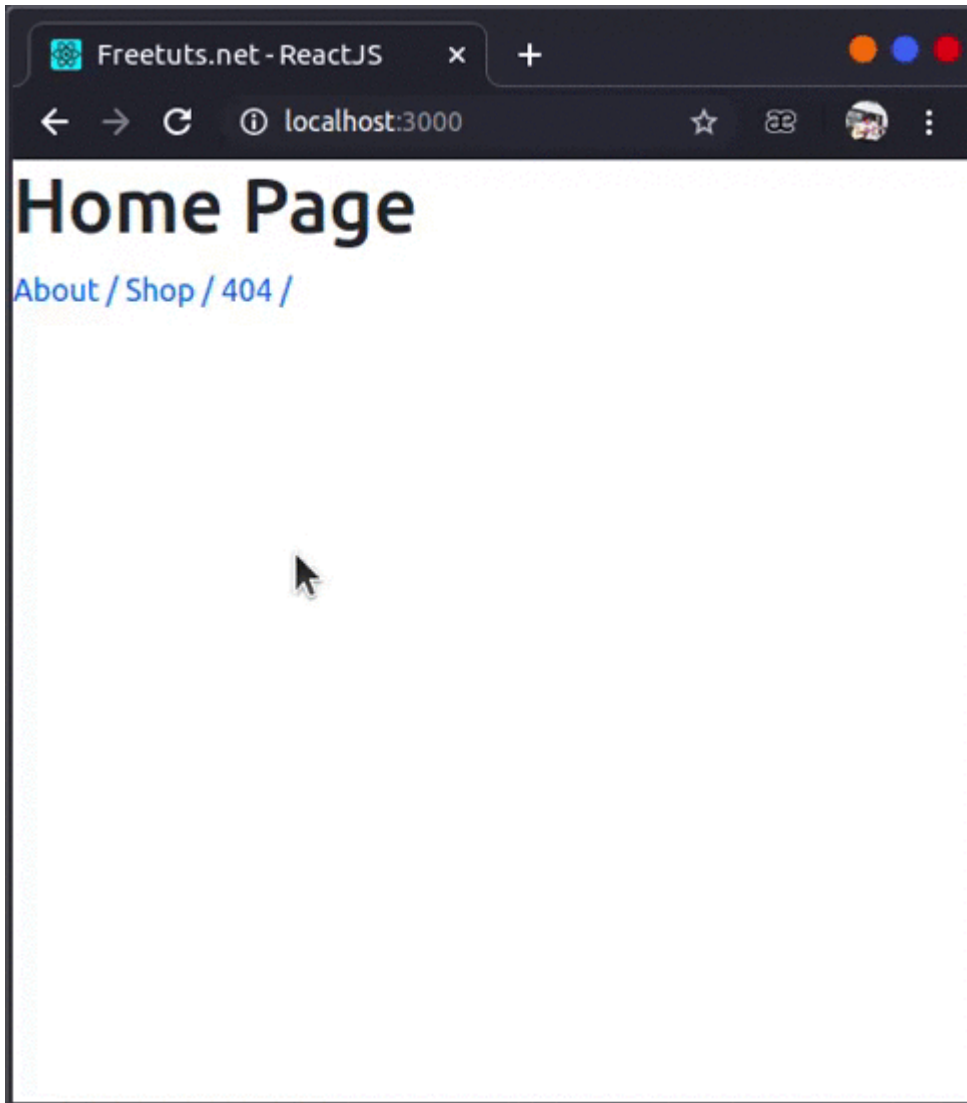
1  //file: components/Shop.js
2
3  import React from "react";
4  import { Link } from 'react-router-dom';
5
6  export default function Shop() {
7      return (
8          <div>
9              <h1> Shop Page</h1>
10              <Link to="/">Home Page </Link>
11          </div>
12      );
13  };

1  //file: components/Error.js
2
3  import React from "react";
4  import { Link } from 'react-router-dom';
5
6  export default function Shop() {
7      return (
8          <div>
9              <h1> 404 - Error Page</h1>
10              <Link to="/">Home Page </Link>

```

```
11         </div>
12     );
13 };
```

Component Link cho phép chúng ta chuyển qua lại giữa các component thông qua URL, nó tương tự như thẻ `a` trong html. Khởi chạy ứng dụng và chúng ta sẽ thấy sự thay đổi.



Sau khi thực hiện các bước cài đặt chúng ta có thể click vào các link để chuyển hướng qua lại trong SPA cụ thể là trong ReactJS rồi.

27. Tích hợp Redux vào ReactJS

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về cách tích hợp Redux vào trong một dự án ReactJS.

Redux là một thư viện hỗ trợ chúng ta quản lí các state trong các ứng dụng javascript. Có thể khẳng định đây là một thư viện khá quan trọng nhưng đối với những người mới làm quen với Redux việc tích hợp nó vào dự án React không đơn giản.

Nội dung bài viết sẽ chỉ ra từng bước để tích hợp Redux vào ReactJS một cách chi tiết nhất.

Chúng ta nên tham khảo thêm bài viết về [Redux là gì? Tại sao lại ứng dụng trong ReactJS](#) để có thể hiểu rõ các khái niệm được giới thiệu ở trong bài viết này.

Table of Content

- [1. Cài đặt Redux](#)
- [2. Tích hợp Redux vào ReactJS](#)
 - [Khởi tạo các hằng](#)
 - [Khởi tạo actions](#)
 - [Khởi tạo reducers](#)
 - [Tích hợp Redux](#)
 - [Lấy và cập nhật giá trị của state từ Store](#)

63. Cài đặt Redux

Sau khi khởi tạo một dự án ReactJS, để có thể sử dụng Redux chúng ta cần phải cài đặt 2 module là `redux` và `react-redux` bằng cách sử dụng npm:

```
1 npm install redux react-redux --save
```

Sau khi cài đặt thành công chúng ta có thể bắt đầu sử dụng redux trong dự án của mình rồi !

64. Tích hợp Redux vào ReactJS

Để hiểu rõ hơn về các bước tích hợp Redux chúng ta sẽ đi xây dựng bộ khung Redux cho một ứng dụng ghi chú.

Ở đây chúng ta sẽ chia nhỏ các phần của Redux ra nhiều thư mục. Bạn có thể tìm hiểu về các thành phần quan trọng của redux ở bài viết giới thiệu về Redux ở trên. Chúng ta sẽ làm việc trong thư mục `src`.

```
1  src/
2  ....const/
3      index.js
4  .....actions/
5      index.js
6  ....reducers/
7      ...
8      index.js
9  ----components
10
11  ....App.js
```

Mỗi thư mục sẽ có các nhiệm vụ khác nhau :

- `const`: chứa các hằng số cố định của dự án.
- `actions`: chứa các `actions` dùng để truyền vào hàm `dispatch`.
- `reducers`: chứa các `reducers` trong redux.

Trước khi xây dựng ứng dụng dùng Redux chúng ta cần phải hiểu rõ các thành phần có trong ứng dụng, từ đó xây dựng mô hình triển khai Redux một cách khoa học hơn.

lvi. Khởi tạo các hằng

Chúng ta sẽ đi khởi tạo các hằng số hỗ trợ việc triển khai dự án. Trong file `const/index.js` chúng ta sẽ khởi tạo 1 hằng số hỗ trợ việc thêm ghi chú.

```
1  // const/index.js
2  export const ADD_NEW_NOTE = "ADD_NEW_NOTE";
```

Trong các dự án lớn sẽ có rất nhiều hằng số, bởi vậy việc khởi tạo ra một thư mục chứa các hằng số là điều hoàn toàn cần thiết.

lvii. Khởi tạo actions

Actions là một object chứa các hành động mà bạn muốn gửi đến `reducers`. Giả sử như chúng ta muốn thêm note thì chúng ta sẽ chỉ định nó bên trong `actions`. Khi muốn gửi `actions` đến `reducers` chỉ cần gọi `store.dispatch(actions)`. Ở đây chúng ta sẽ chỉ định các actions hỗ trợ việc thêm ghi chú như sau:

```
1 // actions/index.js
2 import { ADD_NEW_NOTE, REMOVE_NOTE, EDIT_NOTE } from "../const/index";
3 export const actAddNote = (content) => {
4   return {
5     type: ADD_NEW_NOTE,
6     content,
7   };
8 };
```

Mỗi `action` chúng ta cần phải chỉ định thuộc tính **type** có giá trị duy nhất. Bởi khi action gửi đến reducer nó sẽ dựa vào thuộc tính **action.type** để xác định mình nên làm gì với state.

lviii. Khởi tạo reducers

Reducers sẽ có nhiệm vụ thay đổi state của ứng dụng dựa trên từng hành động được gửi đi. Trong các dự án lớn chúng ta cần chia ra rất nhiều reducers khác nhau. Ở trong thư mục `src/reducers` sẽ chỉ khởi tạo 1 reducers có tên `noteReducer`.

```
1 // reducers/noteReducers.js
2 import { ADD_NEW_NOTE, REMOVE_NOTE, EDIT_NOTE } from "../const/index";
3
4 const noteReducers = (state = [], action) => {
5   switch (action.type) {
6     case ADD_NEW_NOTE:
7       const generateID = new Date().getTime();
8       state = [...state, { id: generateID, content: action.content }];
9       return state;
10    default:
11      return state;
```

```

12     }
13   };
14
15   export default noteReducers

```

Chúng ta sẽ gộp các reducer lại với nhau bằng hàm `combineReducer`.

```

1   // src/reducers/index.js
2
3   import {combineReducers} from 'redux'
4   import noteReducers from './noteReducer'
5
6   //Ở đây chúng ta có thể gộp nhiều reducers
7   // Ở ví dụ này mình chỉ có 1 reducers là noteReducers
8   export default combineReducers({
9     note: noteReducers
10  })

```

lix. Tích hợp Redux

Sau khi đã tạo ra các thành phần cần thiết trong ứng dụng React chúng ta cần phải tạo **Store** lưu trữ state. Chúng ta sẽ làm việc với file `src/index.js`

```

1   // src/index.js
2
3   import React from "react";
4   import ReactDOM from "react-dom";
5   import "./index.css";
6   import App from "./App";
7   import * as serviceWorker from "./serviceWorker";
8

```



```

9
10 import { Provider } from "react-redux";
11 import { createStore } from "redux";
12
13 //Gọi reducers
14 import reducers from "../reducers/index";
15 //Tạo store
16 const store = createStore(reducers);
17
18 ReactDOM.render(
19   <React.StrictMode>
20     <Provider store={store}>
21       <App />
22     </Provider>
23   </React.StrictMode>,
24   document.getElementById("root")
25 );
26
27 // If you want your app to work offline and load faster, you can change
28 // unregister() to register() below. Note this comes with some pitfalls.
29 // Learn more about service workers: https://bit.ly/CRA-PWA
30 serviceWorker.unregister();

```

Để các component khác có thể lấy dữ liệu chúng ta cần phải bọc các component vào trong **Provider**.

1x. Lấy và cập nhật giá trị của state từ Store

Sau khi đã hoàn thành xong tất cả các bước cài đặt Redux cho project, chúng ta có thể thực hiện lấy và cập nhật giá trị của state ở store về component. Giả sử ở đây chúng ta muốn tương tác với store ở component App.js sẽ thực hiện như sau:

```

1    // src/App.js
2
3    //Import kết nối tới react-redux
4    import { connect } from 'react-redux'
5    //Import action dùng để dispatch
6    import {actAddNote} from '../actions/index'
7
8    function App(props) {
9
10       return (
11         ...
12       );
13     }
14
15     //Gán dispatch thành props
16     const mapDispatchToProps = (dispatch) => {
17       return {
18         addNote: (content) => {
19           dispatch(actAddNote(content))
20         }
21       }
22     }
23
24     //Gán giá trị của state thành props
25     const mapStateToProps = (state, ownProps) => {
26       return {
27         note: state.note
28       }

```

```
29     }  
30  
31     //Export component với kết nối redux.  
32     export default connect(mapStateToProps, mapDispatchToProps)(App)
```

Để kết nối với redux ở trong component chúng ta cần phải import hàm kết nối. Ở đây có 2 hàm cực kì quan trọng giúp thao tác với state đó là:

- `mapStateToProps`: giúp chuyển state sang thành props sử dụng trong component.
- `mapDispatchToProps`: giúp chuyển dispatch trong redux thành props. Giả sử mình muốn thực hiện dispatch action `actAddNote` thì mình chỉ cần gọi `props.addNote()`

28. Cách đẩy ứng dụng ReactJS lên Heroku và Deploy trên đó

Trong bài này mình sẽ hướng dẫn cách đẩy ứng dụng ReactJS lên Heroku, cài đặt thêm các phần mềm cần thiết trước khi đẩy ứng dụng lên Heroku.

Cho bạn nào chưa biết thì Heroku mà một nền tảng đám mây hỗ trợ nhiều ngôn ngữ, giúp lập trình viên có thể xây dựng, triển khai, quản lý và mở rộng ứng dụng của mình. Ưu điểm của nó là không phải quan tâm đến việc vận hành máy chủ hay phần cứng. Dưới đây là các bước để **deploy ứng dụng ReactJS lên Heroku** đơn giản nhất.

Table of Content

- [1. Cài đặt phần mềm hỗ trợ đưa ứng dụng lên Heroku](#)

- [Window và MacOS](#)
- [Ubuntu](#)

- [2. Đưa ứng dụng ReactJS lên Heroku](#)

- [Thêm Buildpacks](#)
- [Đẩy source code lên Heroku](#)
- [Tinh chỉnh ứng dụng](#)

65. Cài đặt phần mềm hỗ trợ đưa ứng dụng lên Heroku

Heroku hỗ trợ rất nhiều cách thức để thực hiện deploy một ứng dụng. Bài viết này sẽ sử dụng Heroku Git bởi nó là cách đơn giản và quen thuộc nhất. Chúng ta cần cài đặt 2 phần mềm chính đó là *Git* và *Heroku CLI*.

lxi. Window và MacOS

Đối với Window và MacOS thì các bước cài đặt rất đơn giản chỉ cần download trên trang chủ và cài đặt.

- Download [Git](#)
- Download [Heroku CLI](#)

Các bước cài đặt giống như các phần mềm thông thường, sau khi cài đặt tiến hành restart máy và thực hiện các bước tiếp theo.

lxii. Ubuntu

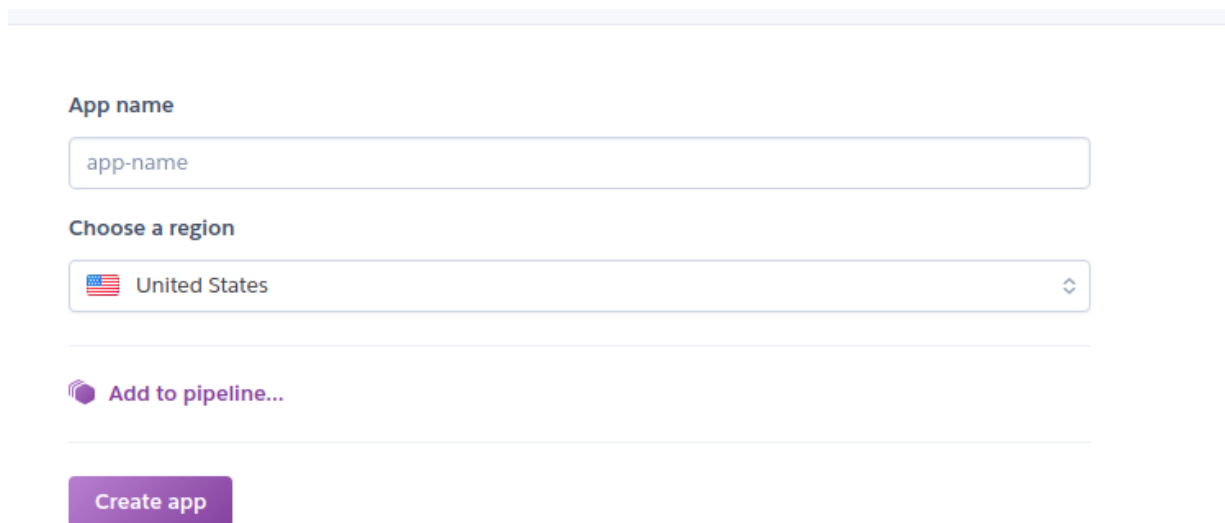
Đối với ubuntu thì sẽ chỉ cần cài đặt bằng các câu lệnh khá đơn giản.

- 1 `sudo apt install git`
- 2 `sudo snap install --classic heroku`

Chúng ta có thể tham khảo thêm bài viết về [Git căn bản](#) để có thể hiểu rõ hơn vì hướng dẫn này sử dụng git trong quá trình *deploy ứng dụng*.

66. Đưa ứng dụng ReactJS lên Heroku

Sau khi cài đặt các phần mềm cần thiết, chúng ta sẽ tiến hành tạo tài khoản Heroku, truy cập vào dashboard và tạo một app mới tại địa chỉ <https://dashboard.heroku.com/new-app>. **App name** sẽ là *subdomain* của bạn. URL sẽ có dạng là [appname].herokuapp.com.



Ngoài ra Heroku còn cho phép tùy chọn vị trí của máy chủ. Sau khi đã tùy chọn các bước, tiến hành click vào **Create App** để hoàn tất.

lxiii. Thêm Buildpacks

Đối với app *ReactJS* được khởi tạo bởi câu lệnh `create-react-app` thì trước khi đẩy source code lên app thì chúng ta cần phải thêm **buildpacks** cho nó. Ở phần **Settings -> Buildpacks -> Add Buildpacks** và điền vào đường dẫn buildpacks.

- 1 `https://github.com/mars/create-react-app-buildpack.git`

Buildpacks sẽ chứa các câu lệnh để biên dịch ứng dụng, nó sẽ được khởi chạy mỗi khi ứng dụng được thiết lập.

lxiv. Đẩy source code lên Heroku

Việc đầu tiên, chúng ta cần phải đăng nhập tài khoản heroku trên máy bằng terminal, sử dụng dòng lệnh.

```
heroku login
```

Sau đó, điền thông tin đăng nhập tài khoản heroku vừa dùng để khởi tạo lúc đầu vào. Khi đăng nhập thành công, chúng ta sẽ tiến hành deploy dự án lên. Giả sử dự án ReactJS cần deploy nằm ở trong thư mục `freetuts-react`. Mở terminal và truy cập vào thư mục `freetuts-react` bằng câu lệnh.

```
cd freetuts-react
```

Khởi tạo git và chỉ định remote tới app trên heroku.

```
git init
```

```
heroku git:remote -a [ten-app]
```

Tiếp theo, chúng ta sẽ deploy app lên heroku bằng cách push tới branch **master** của heroku.

```
git add *
```

```
git commit -a "first deploy"
```

```
git push heroku master
```

Sau khi hoàn tất, heroku sẽ cài đặt ứng dụng ReactJS tự động.

```

❌ tri@tri freetuts-react ↵ master git push heroku master
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 8 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (24/24), 200.74 KiB | 4.10 MiB/s, done.
Total 24 (delta 0), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:       NPM_CONFIG_LOGLEVEL=error
remote:       NODE_ENV=production
remote:       NODE_MODULES_CACHE=true
remote:       NODE_VERBOSE=false
remote:
remote: -----> Installing binaries
remote:       engines.node (package.json):  unspecified
remote:       engines.npm (package.json):    unspecified (use default)
remote:       engines.yarn (package.json):   unspecified (use default)
remote:

```

Đợi quá trình cài đặt hoàn tất, ứng dụng sẽ được chạy trên địa chỉ `[ten-app].herokuapp.com`.

lxv. Tinh chỉnh ứng dụng

Heroku cho phép tinh chỉnh cũng như theo dõi ứng dụng một cách đơn giản, sau đây là một vài chức năng mà hay dùng ở heroku.

Để theo dõi ứng dụng theo thời gian thực chúng ta dùng câu lệnh.

```
heroku logs --tail
```

Chỉnh sửa các biến môi trường của dự án ở phần **Settings -> Config Vars**, ở đây chúng ta có thể thêm, sửa, xóa các biến môi trường.

Config Vars

Hide Config Vars

There are no config vars for this app yet
[Learn about config vars](#) in the Dev Center.

WEBSITE

freetuts.net

Add

Phiên bản miễn phí của Heroku chỉ sử dụng ở mục đích học tập, bởi khi ứng dụng không được truy cập trong một khoảng thời gian thì Heroku tạm thời offline ứng dụng cho tới khi có truy cập thì ứng dụng sẽ được khởi chạy trở lại. Mục đích chính là để tiết kiệm tài nguyên hệ thống, đối với các ứng dụng trung bình trở lên thì chúng ta nên sử dụng cách khác để deploy ứng dụng của mình.