GIÁO TRÌNH REACTJS

Module 01: Cài đặt và kiến thức căn bản

Bài 01: React.js là gì?

- React.js là một thư viện Javascript để xây dựng giao diện người dùng, được phát triển ban đầu từ Facebook và đóng góp bởi cộng đồng lập trình viên trên thế giới.
- React.js có thể sử dụng để viết phần nền cho single-page hoặc ứng dụng di động, giúp trang lấy và xử lý dữ liệu được tối ưu hơn các phương pháp khác.
- React.js thích hợp với các ứng dụng lớn, khả năng mở rộng ở tương lai.
- Điểm mạnh của React.js dễ thấy nhất có lẽ là phần Component:
    - React.js chia nhỏ các phần của trang thành từng phần riêng biệt để xử lý, gọi là component, giúp dễ quản lý, dễ sử dụng ở nhiều nơi.
    - Mỗi khi dữ liệu được cập nhật mới, thay vì thay đổi cả trang, thì React.js sẽ giúp thay đổi chỉ component liên quan, việc này sẽ tối ưu rất nhiều thời gian làm mới dữ liệu.
    - Đặc biệt là hầu hết các component thường dùng đã được phát triển và chia sẻ, chỉ cần cài đặt và sử dụng.
    - …
- Điểm "không mạnh" của React.js có lẽ là ... khó học đối với một số bạn mới tiếp xúc lần đầu, hoặc với những bạn đã quen với cách viết các thư viện khác như jQuery.

Bài 02: Cài đặt React.js

Để cài đặt React.js có rất nhiều cách, sau đây chúng ta cùng nhau tìm hiểu 2 cách sau:

- Sử dụng trực tiếp React.js từ CDN thông qua tag <script>
- Cài đặt React.js bằng lệnh

Cách 1: Cài đặt qua CDN

Tạo file index.html với nội dung sau:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello React</title>
    <script
src="https://unpkg.com/react@16/umd/react.development.js"
></script>
    <script
src="https://unpkg.com/react-dom@16/umd/react-dom.develop
ment.js"></script>
    <script
src="https://unpkg.com/@babel/standalone/babel.min.js"></
script>
  </head>
  <body>
    <div id="root"></div>

    <script type="text/babel">
      ReactDOM.render(
        <h1>Hello, React!</h1>,
        document.getElementById('root')
      );
    </script>
  </body>
</html>
```

- , React sẽ load nội dung tới id này.
- `react.development.js` và `react-dom.development.js`, là 2 file thư viện của React.
- `babel.min.js`: cần thiết cho cấu trúc JSX (Javascript XML - viết cấu trúc XML trong Javascript, như trên là cấu trúc <h1>Hello, world!</h1>), nếu bạn viết React mà không dùng cấu trúc JSX thì không cần thêm <script> này.

- Chỉ với 1 file HTML như trên thôi, là bạn có thể chạy được nội dung React rồi đó.

Cách 2: Cài đặt React.js bằng lệnh

Để cài đặt một project React hoàn chỉnh (liên kết file, sử dụng component riêng biệt, tạo thư viện tùy ý, ...) ta thực hiện các bước sau:

Bước 1 - Cài đặt môi trường Node.js

Download và cài đặt Nodejs tại link sau: https://nodejs.org/en/

Bước 2 - Cài đặt bộ cài app React

Có thể cài ở bất kỳ ổ đĩa hay thư mục nào bằng lệnh cmd

```
npm install -g create-react-app
```

Bước 3 - Tạo project React.js

Trỏ vào thư mục muốn cài đặt (Dùng câu lệnh cd)

Chạy lệnh tạo project với tên react-project (tên project tùy ý) như bên dưới

```
create-react-app react-project
```

Bước 4 - Thực thi lệnh chạy

Trỏ vào thư mục react-project và chạy project

```
cd react-project
```

```
npm start
```

Bài 03: React.js render HTML

- Render là quá trình hiển thị nội dung lên trình duyệt cho người

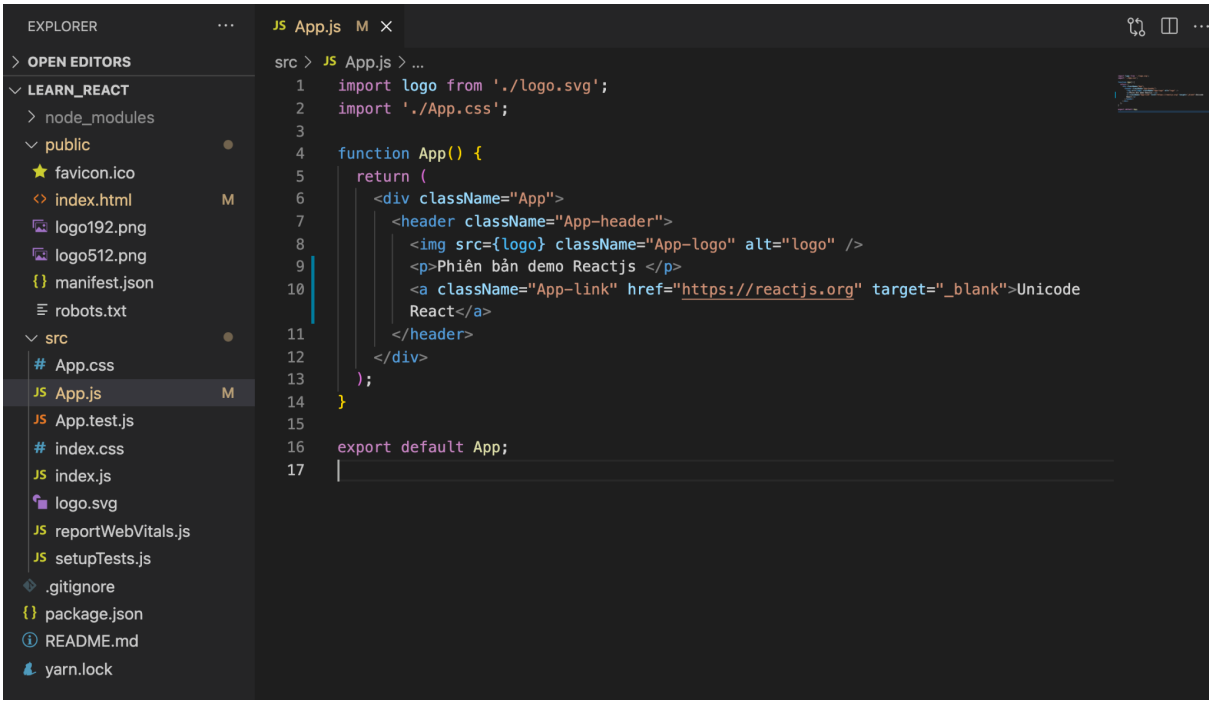
dùng nhìn thấy

- Với React.js thì nội dung layout bạn viết không phải nằm ở trang HTML, mà được viết bên trong file Javascript, file HTML chỉ là cầu nối giúp nội dung Javascript "liên kết" với trình duyệt
- Xem ví dụ sau

```
<div id="root"></div>

<script type="text/babel">
  ReactDOM.render(
    <h1>Hello, React!</h1>,
    document.getElementById('root')
  );
</script>
```

Cấu trúc thư mục project React.js



- **node_modules**: chứa nội dung cài đặt, tất cả cài đặt sẽ được lưu tại đây
- **public**: chứa tất cả file output, là các file sẽ tương tác trực tiếp với trình duyệt như: HTML, image,…
- **src**: chứa tất cả các file input, đây là các file mà chúng ta sẽ code nội dung, thao tác phần lớn ở những file này, gồm các file Javascript, CSS,...

Cách React.js Render HTML lên trình duyệt

Để hiểu rõ hơn về render trong React, ta xem xét nội dung 3 file:

- /public/index.html
- /src/App.js
- /src/index.js

Để cho dễ hiểu, ta lần lượt viết lại nội dung các file như sau:

File /public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Ta thấy file này không chứa bất kỳ nội dung nào hiển thị ngoài trình duyệt

File /src/App.js

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>Hello, React!.</p>
      </header>
    </div>
  );
}

export default App;
```

Ta thấy đoạn code màu xanh bên trên chính là nội dung được hiển thị ngoài trình duyệt.
File này liên kết với `/public/index.html` thông qua `/src/index.js`

File /src/index.js

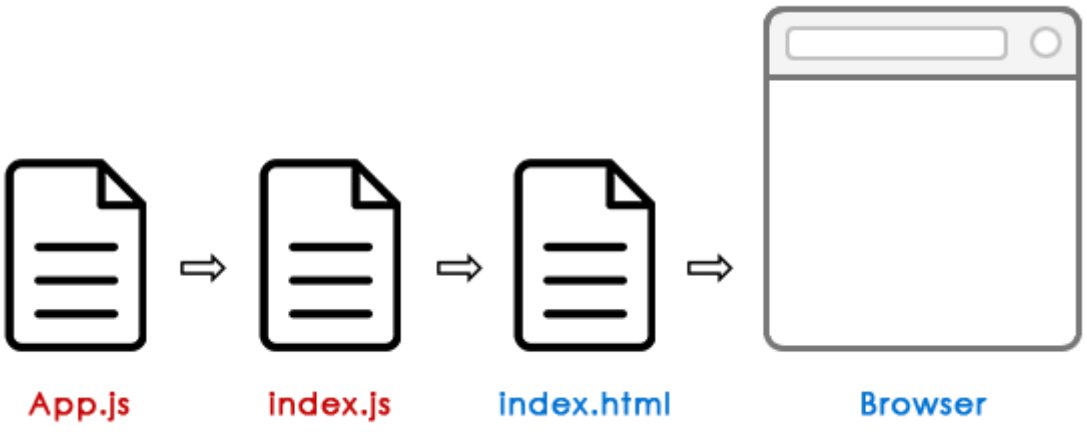
```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App'; //Load function App từ App.js
import * as serviceWorker from './serviceWorker';

ReactDOM.render(<App />,
document.getElementById('root'));

serviceWorker.unregister();
```

- Import thư viện React và ReactDOM: import React from 'react'; import ReactDOM from 'react-dom';
- Import './index.css'; import nội dung file index.css
- Import App from './App'; import nội dung file App.js
- ReactDOM.render(<App />, document.getElementById('root')); render nội dung function App tới id="root"

Nội dung trên có nghĩa là file /src/index.js lấy nội dung từ function App của file /src/App.js render ra nội dung trả về id="root" của file /public/index.html, sau đó hiển thị nội dung này ra trình duyệt



Tới đây các bạn sẽ thắc mắc tại sao có thể liên kết được với file /public/index.html trong khi bên trong file `/public/index.html` không chứa bất kỳ liên kết Javascript nào? Câu trả lời nằm file `/src/serviceWorker.js`, nội dung file này được import trực tiếp bên trong /src/index.js đóng vai trò như một Route điều hướng, liên kết các file.

Bài 04: JSX

JSX (Javascript XML), là cấu trúc XML được viết bên trong cấu trúc Javascript (HTML cũng được viết theo cấu trúc XML nhé), do đó có thể hiểu đơn giản hơn JXS giống như cách viết HTML bên trong cấu trúc Javascript

Vậy tại sao cần JSX? các bạn thử xem 2 cấu trúc sau nhé

React	JSX
`const element = React.createElement( 'h1', {className: 'block'}, 'Hello, world!' );`	`const element = ( <h1 className="block">Hello, world!</h1> );`

- Để render thẻ <h1> ta thấy rõ ràng cách viết JSX ngắn gọn hơn, và nó gần giống với cấu trúc HTML quen thuộc.
- Đây chỉ mới có 1 thẻ <h1>, nếu nội dung là một file HTML hoàn chỉnh thì cấu trúc bên trái sẽ rất phức tạp.
- Ngoài ra JSX chống tấn công kiểu injection, do mặc định React DOM sẽ loại bỏ những ký tự đặc biệt trong bất kì giá trị nào được nhúng vào JSX trước khi được render, việc này giúp ngăn chặn phương thức tấn công XSS (cross site scripting).
- Đó là lý do chúng ta nên viết React theo cấu trúc JSX.

Khác biệt giữa HTML và JSX

Mô tả	Cấu trúc HTML	Cấu trúc JSX
Tên Class	`<tag class="">`	`<tag className="">`
Thuộc tính value <input />	`<input value="" />`	`<input defaultValue="" />`
Thuộc tính for của <label>	`<label for="">`	`<label htmlFor="">`
Giá trị của <select><option>	`<option value="">`	`<option value={}>`
Inline CSS	`<tag style="width: 10%">`	`<tag style={{ width: '10%' }}>`
Sự kiện	`<tag onclick="functionName()">`	`<tag onClick={functionName}>`
Truyền giá trị		`<img src={path}> Hello {name}!`

Công cụ chuyển đổi HTML sang JSX:
https://transform.tools/html-to-jsx

Bài 05: Component

- React chia nhỏ các phần của trang thành từng phần riêng biệt, gọi là component, ví dụ như các phần: header, footer, sidebar, navigation, itemList, …
- Tính chất của component giống như một hàm (function) Javascript, có thể tái sử dụng ở nhiều nơi khác nhau.

Cách viết Component trong React.js

- Viết dưới dạng function (hoặc Class - nói ở bài sau), và function luôn được return.
- Bên trong return luôn tồn tại tag bao ngoài tất cả (tag wrap).
- Sử dụng thư viện React.DOM để render một component.
- Function render phải có cấu trúc XML, viết dưới dạng <Tên /> hoặc <Tên></Tên>

```
<div id="member"></div>

<script type="text/babel">
  function Member(){
    return(
      <div className="member">
        <h2>Nguyễn Văn A</h2>
        <p>Tuổi: 25</p>
      </div>
    )
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

- Function Member() đóng vai trò là một component, nhiệm vụ là tạo một đối tượng member với tên và tuổi.
- ReactDOM.render giữ vai trò render nội dung ra trình duyệt, dựa vào id="member".

Cách tạo và Include Component

- Giả sử ta có nhiều component khác nhau trong một trang, mỗi component có một nhiệm vụ riêng, Ví dụ:
  - Component avatar dành để xử lý hình ảnh member.
  - Component memberInfo dành để hiển thị thông tin member như tên, tuổi.
  - Component comment dành để quản ý bình luận của member.
- Nhiệm vụ của ta là làm sao viết kết hợp các component này lại để hiển thị thông tin cho member, ta làm như sau:

```
<div id="member"></div>

<script type="text/babel">
  function Avatar(){
```

```
    return(
      <div className="avatar">
        <img src="img_avatar.jpg" alt="" />
      </div>
    )
  };

  function MemberInfo(){
    return(
      <div className="info">
        <h2>Nguyễn Văn A</h2>
        <p>Tuổi: 25</p>
      </div>
    )
  };

  function Comment(){
    return(
      <div className="comment">
        Lorem ipsum dolor sit amet, consectetur
adipiscing elit proin sit amet neque.
      </div>
    )
  };

  function Member(){
    return(
      <div className="member">
        <Avatar />
        <MemberInfo />
        <Comment />
      </div>
    )
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
```

```
</script>
```

- Với ví dụ trên, các bạn sẽ thấy việc tạo component rất đơn giản, chỉ cần tách các phần riêng biệt cho từng chức năng khác nhau, là ta đã có được các component, việc này thuận lợi trong việc xử lý các phần riêng mà không bị ảnh hưởng hay bị phân tâm với các component khác.
- Khi làm thực tế, ta cần tách từng component ra riêng một file Javascript để tiện sử dụng, cách làm như thế nào sẽ được giới thiệu sau nhé.

Bài 06: Props

Component thực chất là một function Javascript (hoặc class), phương thức truyền giá trị giữa các function này được gọi là props, chúng được React xử lý và trả kết quả hiển thị trên trình duyệt.

```
<div id="member"></div>

<script type="text/babel">
  function Member(props){
    return(
      <div className="member">
        <h2>{ props.name }</h2>
        <p>Tuổi: { props.age }</p>
      </div>
    )
  };

  ReactDOM.render(
    <Member name="Nguyễn Văn A" age="25" />,
    document.getElementById('member')
  );
</script>
```

Ở component Member ta đã sử dụng props, giống như cách báo cho hệ thống React biết nơi nào xuất hiện props thì nơi đó sẽ được liên kết với các biến (name, age), khi render thì các tham số (Nguyễn Văn A, 25) sẽ được truyền vào các biến này

Tái sử dụng Component trong cùng khu vực

Giả sử ta cần render thêm một member với thông tin "Trần Thị B", "18" tuổi, thay vì viết thêm một component mới, thì ta có thể sử dụng lại component đã có như sau

```
<div id="member"></div>

<script type="text/babel">
  function Member(props){
    return(
      <div className="member">
        <h2>{ props.name }</h2>
        <p>Tuổi: { props.age }</p>
      </div>
    )
  };

  var memberlist = (
    <div>
      <Member name="Nguyễn Văn A" age="25" />
      <Member name="Trần Thị B" age="18" />
    </div>
  );

  ReactDOM.render(
    memberlist,
    document.getElementById('member')
  );
</script>
```

Với cách khai báo một biến mới memberlist như trên, ta có thể sử dụng nhiều component cùng lúc, việc này sẽ giúp giải quyết nhiều cấu trúc sau này.

Tái sử dụng Component ở nhiều nơi khác nhau

Member mới có thông tin "Trần Thị B", "18" tuổi, tuy nhiên thông tin này được sử dụng ở một khu vực khác, không thuộc về id="member" thì ta chỉ việc thêm một render mới

```
<div id="member"></div>
<div id="client"></div>

<script type="text/babel">
  function Member(props){
    return(
      <div className="member">
        <h2>{ props.name }</h2>
        <p>Tuổi: { props.age }</p>
      </div>
    )
  };

  ReactDOM.render(
    <Member name="Nguyễn Văn A" age="25" />,
    document.getElementById('member')
  );

  ReactDOM.render(
    <Member name="Trần Thị B" age="18" />,
    document.getElementById('client')
  );
</script>
```

Bài 07: Props xử lý dữ liệu

Bài học này sẽ giúp bạn xử lý dữ liệu truyền vào cho một component

Cùng xem đoạn code sau:

```
<div id="member"></div>

<script type="text/babel">
  function Member(props){
    return(
      <div className="member">
        <h2>{ props.name }</h2>
        <p>Tuổi: { props.age }</p>
```

```
      </div>
    )
  };

  ReactDOM.render(
    <Member name="Nguyễn Văn A" age="25" />,
    document.getElementById('member')
  );
</script>
```

- Thay vì viết dữ liệu trực tiếp bên trong phần render, ta sẽ tách riêng phần dữ liệu ra một dữ liệu của một đối tượng (data Object).
- Khi này ta cần xử lý lại phần render sao cho có thể lấy được dữ liệu, thao tác rất đơn giản như sau:

```
<div id="member"></div>

<script type="text/babel">
  function Member(props){
    return(
      <div className="member">
        <h2>{ props.name }</h2>
        <p>Tuổi: { props.age }</p>
      </div>
    )
  };

  const info = {
    name: 'Nguyễn Văn A',
    age: '25'
  };

  ReactDOM.render(
    <Member name={info.name} age={info.age} />,
    document.getElementById('member')
  );
</script>
```

- **const info** khai báo giá trị cho một object.
- **info.name** lấy giá trị name của object.
- **info.age** lấy giá trị age của object.

Bài 08: Props- liên kết data trong Component

Bài học trước ta đã biết các xử lý data đơn giản cho một component, bài học này sẽ giúp các bạn hình dung được các dữ liệu được truyền giữa các component như thế nào

Xem lại đoạn code sau:

```
<div id="member"></div>

<script type="text/babel">
  function Avatar(){
    return(
      <div className="avatar">
        <img src="img_avatar.jpg" alt="" />
      </div>
    )
  };

  function MemberInfo(){
    return(
      <div className="info">
        <h2>Nguyễn Văn A</h2>
        <p>Tuổi: 25</p>
      </div>
    )
  };

  function Comment(){
    return(
      <div className="comment">
        Lorem ipsum dolor sit amet, consectetur
adipiscing elit proin sit amet neque.
      </div>
    )
```

```
  };

  function Member(){
    return(
      <div className="member">
        <Avatar />
        <MemberInfo />
        <Comment />
      </div>
    )
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

Tạo một data object như sau:

```
const member = {
  text: 'Lorem ipsum dolor sit amet, consectetur
adipiscing elit proin sit amet neque.',
  info: {
    path: 'img_avatar.jpg',
    name: 'Nguyễn Văn A',
    age: 25
  }
}
```

Lần lượt tiến hành xử lý dữ liệu trong các function như sau:

```
function Avatar(props){
  return(
    <div className="avatar">
      <img src={props.info.path} alt="" />
    </div>
  )
```

```
};

function MemberInfo(props){
  return(
    <div className="info">
      <h2>{props.info.name}</h2>
      <p>Tuổi: {props.info.age}</p>
    </div>
  )
};

function Comment(props){
  return(
    <div className="comment">
      {props.text}
    </div>
  )
};

function Member(){
  return(
    <div className="member">
      <Avatar info={member.info} />
      <MemberInfo info={member.info} />
      <Comment text={member.text} />
    </div>
  )
};
```

- **{props.info.path}** props sẽ lấy giá trị path từ info của dữ liệu, tương tự cho name và age.
- **{props.text}** giá trị text được lấy trực tiếp.
- **<Avatar info={member.info} />** cần khai báo các giá trị dữ liệu tương ứng, tương tự cho function **<MemberInfo />** và **<Comment />**

Xử lý dữ liệu khi các component lồng vào nhau

- Trường hợp này còn gọi là: ***tái sử dụng component***

- Để hiểu rõ hơn việc truyền dữ liệu giữa các component, ta tiến hành lồng function Avatar(props) vào bên trong MemberInfo(props), ta làm như sau (nhớ xóa bỏ function Avatar(props) bên trong Member()):

```
function Avatar(props){
  return(
    <div className="avatar">
      <img src={props.info.path} alt="" />
    </div>
  )
};

function MemberInfo(props){
  return(
    <div className="info">
      <Avatar info={member.info} />
      <h2>{props.info.name}</h2>
      <p>Tuổi: {props.info.age}</p>
    </div>
  )
};

function Comment(props){
  return(
    <div className="comment">
      {props.text}
    </div>
  )
};

function Member(){
  return(
    <div className="member">
      <MemberInfo info={member.info} />
      <Comment text={member.text} />
    </div>
  )
};
```

- Việc di chuyển <Avatar info={member.info} /> từ function Member() vào bên trong MemberInfo(props). Khi này kết quả vẫn không thay đổi.
- Thay vì sử dụng member.info của function <Avatar />, ta sẽ liên kết <Avatar /> với <MemberInfor /> bằng một props.avatar, với avatar là một tên bất kỳ được sử dụng riêng bên trong function <Avatar />, ta viết như sau:

```
function Avatar(props){
  return(
    <div className="avatar">
      <img src={props.avatar.path} alt="" />
    </div>
  )
};

function MemberInfo(props){
  return(
    <div className="info">
      <Avatar avatar={props.info} />
      <h2>{props.info.name}</h2>
      <p>Tuổi: {props.info.age}</p>
    </div>
  )
};
```

- **avatar** ta có thể đặt tên bất kỳ, miễn sao ở function **<MemberInfor />** ta gọi kết quả trả về là props.info là được. Điều này tiện lợi trong việc xử lý các giá trị props ở các component mà không cần quan tâm tới các giá trị props ở các component khác.
- Code hoàn chỉnh như sau:

```
<div id="member"></div>

<script type="text/babel">
  function Avatar(props){
    return(
      <div className="avatar">
        <img src={props.avatar.path} alt="" />
```

```
      </div>
    )
  };

  function MemberInfo(props){
    return(
      <div className="info">
        <Avatar avatar={props.info} />
        <h2>{props.info.name}</h2>
        <p>Tuổi: {props.info.age}</p>
      </div>
    )
  };

  function Comment(props){
    return(
      <div className="comment">
        {props.text}
      </div>
    )
  };

  function Member(){
    return(
      <div className="member">
        <MemberInfo info={member.info} />
        <Comment text={member.text} />
      </div>
    )
  };

  const member = {
    text: 'Lorem ipsum dolor sit amet, consectetur
adipiscing elit proin sit amet neque.',
    info: {
      path: 'img_avatar.jpg',
      name: 'Nguyễn Văn A',
      age: 25
    }
```

```
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

Ta cũng có thể làm tương tự cho component MemberInfo(props) và Member(), ta tiếp tục viết lại code như sau:

```
function MemberInfo(props){
  return(
    <div className="info">
      <Avatar avatar={props.memberInfo} />
      <h2>{props.memberInfo.name}</h2>
      <p>Tuổi: {props.memberInfo.age}</p>
    </div>
  )
};

function Member(){
  return(
    <div className="member">
      <MemberInfo memberInfo={member.info} />
      <Comment text={member.text} />
    </div>
  )
};
```

Với memberInfo là tên bất kỳ bên trong component MemberInfo(props).

Xử lý dữ liệu tại phần render

- Hiện tại giá trị dữ liệu object const member được gọi trực tiếp bên trong Member(), ta có thể gọi giá trị này tại phần render, mục đích để tách biệt rõ các component chỉ nên xử lý các thao tác logic, phần dữ liệu nên đặt tại render
- Ta viết lại component Member() và phần render như sau:

```
function Member(props){
  return(
    <div className="member">
      <MemberInfo memberInfo={props.info} />
      <Comment text={props.text} />
    </div>
  )
};

ReactDOM.render(
  <Member
    info={member.info}
    text={member.text}
  />,
  document.getElementById('member')
);
```

- Qua bài này ta thấy giá trị của Props không thay đổi, chỉ truyền từ component này sang component khác, giá trị dữ liệu phụ thuộc vào dữ liệu object khai báo ban đầu, ví dụ đơn giản: muốn thay đổi giá trị dữ liệu "Nguyễn Văn A" thành "Nguyễn Thị B" bên trong component MemberInfo(props) thì ta làm như thế nào? tất nhiên với props ta không thể thực hiện được.
- Để giải quyết bài toán dữ liệu có thể thay đổi được trong quá trình truyền dữ liệu giữa các component, React cho ra một khái niệm khác, đó là State, bài học sau ta sẽ làm quen với State nhé

Bài 09: Props với Arrow Function

- Arrow Function cấu trúc có những vượt trội hơn so với cách sử dụng cũ, sử dụng tham số cũng linh hoạt hơn.
- Bài học này sẽ giúp các bạn chuyển đổi cách viết một props theo function thông thường sang arrow function.

Cách viết cũ	Arrow Function
`function Member(props){` `  return(`	`var Member = (props) => {` `  return(`

```
    <div
className="member">
      <h2>{ props.name
}</h2>
      <p>Tuổi: { props.age
}</p>
    </div>
  )
};
```

```
    <div
className="member">
      <h2>{ props.name
}</h2>
      <p>Tuổi: { props.age
}</p>
    </div>
  )
};
```

- Nhìn vào so sánh cấu trúc bên trên, ta thấy cấu trúc bên trong function không có gì thay đổi, chỉ thay đổi cách khai báo function
- Nên sử dụng Arrow function thay cho cách viết cũ

Ví dụ Props với Arrow Function

```
<div id="member"></div>

<script type="text/babel">
  var Member = (props) => {
    return(
      <div className="member">
        <h2>{ props.name }</h2>
        <p>Tuổi: { props.age }</p>
      </div>
    )
  };

  ReactDOM.render(
    <Member name="Nguyễn Văn A" age="25" />,
    document.getElementById('member')
  );
</script>
```

Bài 10: Props với Classes

Trong phiên bản Javascript ES6, chúng ta có thể chuyển đổi một function sang dạng Class, việc này có nhiều ưu điểm hơn so với cách viết function, thao tác với các object khá tốt

Cấu trúc Function	Cấu trúc Classes

function Member(props){
  return(
  <div
className="member">
   { props.name
}
  Tuổi: { props.age
}
 	
)
};
	class Member extends
	React.Component {
	  render() {
	  return(
	  <div
	className="member">
	   {
	this.props.name }
	  Tuổi: {
	this.props.age }
	
	  )
	  }
	};

Nhìn vào so sánh cấu trúc bên trên, ta thấy dễ dàng chuyển đổi một function sang Class:

- function Member(props) thành class Member extends React.Component
- Thêm render() {}
- Ở mỗi props thêm this

Ví dụ Props với Classes

```
<div id="member"></div>

<script type="text/babel">
  class Member extends React.Component {
    render() {
      return(
        <div className="member">
          <h2>{ this.props.name }</h2>
          <p>Tuổi: { this.props.age }</p>
        </div>
      )
    }
```

```
  };

  ReactDOM.render(
    <Member name="Nguyễn Văn A" age="25" />,
    document.getElementById('member')
  );
</script>
```

Bài 11: State trong React.js

- Ở bài học trước tôi đã chia sẻ: giá trị của Props không thay đổi, chỉ truyền từ component này sang component khác.
- Tuy nhiên trong thực tế, bản thân component đôi lúc cũng cần xử lý dữ liệu riêng của nó, ví dụ thay đổi thông tin dữ liệu như name, ... do đó React đã cho ra đời khái niệm State để giải quyết bài toán về dữ liệu bên trong component.

Cấu trúc chung của State

Cấu trúc chung của một component sử dụng với State được viết dưới dạng Classes (hoặc arrow function) như bên dưới.

```
class ClassName extends React.Component {
  constructor(props) {
    super(props);
    this.state = {};
  }

  render() {
    return (
      ...
    );
  }
}
```

- ClassName tên Class.
- extends React.Component phương thức kế thừa component (thực ra là kế thừa class đã tồn tại).
- constructor(props) hàm khởi tạo đối tượng cho một class, mỗi

class chỉ chứa một hàm khởi tạo duy nhất.

- super(props) gọi lại constructor trong React.Component, khi này ta mới có thể sử dụng phương thức this được.
- this.state phải là một Object.
- render giống như cách dùng của props

```
<div id="member"></div>

<script type="text/babel">
  class Member extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        name: "Bùi Văn Tèo",
        age: 25
      };
    };

    render() {
      return (
        <div>
          <h2>{this.state.name}</h2>
          <p>Tuổi: {this.state.age}</p>
        </div>
      );
    };
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

Nội dung trên không có gì đặc biệt, chỉ là khai báo State với constructor, gán giá trị ban đầu cho State là this.state, sau đó render nội dung đã được khai báo.

Cấu trúc đơn giản của State

Chúng ta cũng có thể viết một State đơn giản như sau:

```
class ClassName extends React.Component {
  state = {};

  render() {
    return (
      ...
    );
  }
}
```

Xem ví dụ sau:

```
<div id="member"></div>

<script type="text/babel">
  class Member extends React.Component {
    state = {
      name: "Bùi Văn Tèo",
      age: 25
    };

    render() {
      return (
        <div>
          <h2>{this.state.name}</h2>
          <p>Tuổi: {this.state.age}</p>
        </div>
      );
    };
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

- Ta thấy nội dung trên rất đơn giản để khai báo một State
- Vậy tại sao không sử dụng State đơn giản như trên cho gọn, cần gì phải dùng constructor cho rắc rối? câu trả lời là constructor là phương thức (method) luôn luôn được gọi đầu tiên mỗi khi khởi tạo một Classes, nó đảm bảo mọi thứ đã sẵn sàng trước khi bắt đầu một Classes. Tuy nhiên, tùy vào cách sử dụng mà bạn có thể dùng constructor hay không nhé.

Bài 12: setState trong React.js

Bài trước ta đã biết cách viết một State là như thế nào, bài này sẽ giúp các bạn cách cập nhật dữ liệu của component bằng cách sử dụng setState.

Thay đổi dữ liệu bên trong component với State

Muốn thay đổi hay cập nhật nội dung của một State, chúng ta sử dụng **this.setState**, khi này dữ liệu thay đổi sẽ được cập nhật tới State hiện tại.
Để hiểu rõ hơn về **this.setState**, chúng ta xem ví dụ sau:

```
<div id="member"></div>

<script type="text/babel">
  class Member extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        name: "Bùi Văn Tèo",
        age: 25
      };
    };

    changeName = () => {
      this.setState({
        name: "Bùi Văn Tý",
        age: 27
      })
    };
```

```
    render() {
      return (
        <div>
          <h2>{this.state.name}</h2>
          <p>Tuổi: {this.state.age}</p>
          <button type="button"
onClick={this.changeName}>Change name</button>
        </div>
      );
    };
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

- Nội dung trên ta đã dùng this.setState để thay đổi name của State.
- Khi click vào button sẽ gọi function changeName, khi này this.setState sẽ cập nhật lại dữ liệu mới cho State.

Ví dụ về setState

Ta sẽ thử sử dụng setState với dữ liệu Object nhiều phần tử để biết thêm nhiều trường hợp khác nhau:

```
<div id="member"></div>

<script type="text/babel">
  class Member extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        users: [
          { name: "Bùi Văn Tèo", age: 25 },
          { name: "Trần Văn An", age: 29 },
          { name: "Nguyễn Thị Bưởi", age: 22 }
```

```
        ],
        job: "IT Communicator"
      };
    };

    changeName = () => {
      this.setState({
        users: [
          { name: "Bùi Văn Tý", age: 27 },
          { name: "Trần Văn An", age: 29 },
          { name: "Nguyễn Thị Bưởi", age: 22 }
        ],
        job: "Bridge System Engineer"
      })
    };

    render() {
      return (
        <div>
          <h2>{this.state.users[0].name}</h2>
          <p>Tuổi: {this.state.users[0].age}</p>
          <p>Nghề nghiệp: {this.state.job}</p>
          <button type="button"
onClick={this.changeName}>Change Data</button>
        </div>
      );
    };
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

Bài 13: Vòng đời Component trong React.js

Component có thể được viết theo kiểu của vòng đời này như sau:

- Tạo vòng đời (Mount trên DOM)
- Cập nhật vòng đời (Updating)
- Kết thúc vòng đời (Unmount từ DOM)

Quá trình trên được gọi là Component lifecycle (vòng đời của một component).

Các phương thức làm việc

Vòng đời	**Câu lệnh**	**Mô tả**
Tạo vòng đời	componentWillMount()	Phương thức này sẽ được gọi trước khi render. Không nên sử dụng setState() tại đây vì chưa có DOM để tương tác.
	componentDidMount()	Phương thức này sẽ được gọi một lần duy nhất sau khi component được render xong, thường sử dụng setState() vì khi này DOM đã được tạo, thường được dùng khi muốn: ● Get data từ server. ● Lấy giá trị cố định của thành phần như: width, height, offset, … ● Khai báo setInterval cho hành động lặp lại. ● Khai báo sự kiện như: load, scroll, resize, ...
Cập nhật vòng đời	shouldComponentUpdate()	Phương thức này xác định component nên được update hay không, phương thức này có 2 giá trị:

		● true (mặc định) khi này sẽ tiếp tục thực hiện componentWillUpdate() và componentDidUpdate(). ● false sẽ dừng thực hiện componentWillUpdate() và componentDidUpdate().
	componentWillUpdate()	Phương thức này được gọi một lần duy nhất sau shouldComponentUpdate(), sử dụng trước khi re-render component.
	componentDidUpdate()	Chúng ta gọi phương thức này sau khi DOM đã được update xong, tại đây thường được dùng khi muốn: ● Xử lý dữ liệu được lấy từ server. ● Thay đổi giao diện dựa vào dữ liệu nhận được.
Kết thúc vòng đời	componentWillUnMount()	Phương thức này sẽ được gọi một lần duy nhất trước khi component được loại bỏ từ DOM, những thứ đã khai báo tại componentDidMount() cần phải được hủy. Đây cũng là phương thức kết thúc vòng đời của component.

Thực hành với vòng đời Component

Tạo một vòng đời (lifecycle) cho một bộ đếm (counter) tăng từ 0 cho đến 3, khi đạt đến giá trị 3 thì sẽ xóa bộ đếm để kết thúc vòng đời, ta lần lượt thực hiện các bước sau:

- Khai báo ban đầu, render giá trị đầu tiên.
- Cho giá trị bắt đầu tăng sau mỗi 1 giây (cập nhật State).
- Loại bỏ component ra khỏi DOM khi giá trị đạt đến 3 (Xóa component để xem kết thúc vòng đời).

Ta có đoạn code sau:

```
<div id="counter"></div>

<script type="text/babel">
  class Counter extends React.Component {
    constructor(props) {
      super(props);
      this.state = { count: 1 };
    };

    render() {
      return (
        <h2>{this.state.count}</h2>
      );
      console.log('count = ' + this.state.count);
    };
  };

  ReactDOM.render(
    <Counter />,
    document.getElementById('counter')
  );
</script>
```

Cho giá trị bắt đầu tăng sau mỗi 1 giây

- Tạo function count để cập nhật giá trị với setState.
- Cần setInterval để giá trị tự động cập nhật function count.
- Do component đã được render xong, nên setInterval ta đặt bên

trong componentDidMount (như đã nói ở table bên trên)

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 1 };
  };

  countFnc = () => {
    this.setState({
      count: this.state.count + 1
    });
  };

  componentDidMount() {
    setInterval(() => this.countFnc(), 1000);
    console.log("componentDidMount");
  };

  render() {
    console.log('count = ' + this.state.count);
    return (
      <h2>{this.state.count}</h2>
    );
  };
};

ReactDOM.render(
  <Counter />,
  document.getElementById('counter')
);
```

Giá trị sẽ được tăng 1 đơn vị trên giây, Do chưa có điều kiện kết thúc nên giá trị sẽ được tăng vô hạn.

Loại component ra khỏi DOM khi giá trị đạt đến 3

- Ta tận dụng việc gọi một lần duy nhất của phương thức componentDidUpdate để loại bỏ component ra khỏi DOM khi giá

trị đạt đến 3.

- Sử dụng ReactDOM.unmountComponentAtNode() để loại bỏ component ra khỏi DOM.
- Sau khi Component đã được loại bỏ khỏi DOM, ta cần sử dụng phương thức componentWillUnmount() để xóa interval đã đăng ký từ componentDidMount(), nếu không sẽ báo lỗi vì vòng đời không được kết thúc.

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 1 };
  };

  countFnc = () => {
    this.setState({
      count: this.state.count + 1
    });
  };

  componentDidMount() {
    this.counterID = setInterval(() => this.countFnc(),
1000);
    console.log("componentDidMount");
  };

  componentDidUpdate(prevProps, prevState) {
    console.log("componentDidUpdate");
    if (this.state.count === 3) {

ReactDOM.unmountComponentAtNode(document.getElementById('
counter'));
    }
  };

  componentWillUnmount() {
    console.log("componentWillUnmount");
    clearInterval(this.counterID);
  };
```

```
  render() {
    console.log('count = ' + this.state.count);
    return (
      <h2>{this.state.count}</h2>
    );
  };
};

ReactDOM.render(
  <Counter />,
  document.getElementById('counter')
);
```

- Ta thấy kết quả đã dừng lại ở 3, khi này sau khi gọi phương thức componentWillUnmount() thì vòng đời đã kết thúc.
- Bên trên là một ví dụ đơn giản về cách tạo vòng đời component, trong thực tế tùy vào mục đích sử dụng mà ta có thể điều chỉnh vòng đời dừng tại đâu cho phù hợp, ví dụ nếu tạo một đồng hồ điện tử, thì ta sẽ không cho vòng đời kết thúc, khi đó sẽ không cần sử dụng componentWillUnmount().

Module 02: Xử lý trong React.js

Bài 14: Xử lý sự kiện trong React.js

Cách React.js xử lý sự kiện (event) (thật ra là JSX) có khác biệt so với viết HTML thông thường, xem so sánh sau sẽ rõ.

HTML thông thường	JSX

`<button onclick="fncName()">`
`  Click me!`
`</button>`
	`<button onClick={fncName}>`
	`  Click me!`
	`</button>`

- Thay đổi bên trên khác biệt ở phần onClick={} thay vì onclick="", (cẩn thận chữ C viết hoa).
- Thay đổi khác là ở phần gọi function:

- Khi bạn khai báo fncName() thì function sẽ được gọi khi render.
- Khi bạn khai báo fncName thì function sẽ được gọi khi button được click.

Thực hành sự kiện React.js với button

```
<div id="member"></div>

<script type="text/babel">
  const ActionButton = () => {
    const handleClick = () => {
      console.log('Button was clicked!');
    }

    return (
      <button onClick={handleClick}>
        Click me
      </button>
    );
  }

  ReactDOM.render(
    <ActionButton />,
    document.getElementById('member')
  );
</script>
```

Bài 15: Xử lý form trong React.js

- Ta có thể sử dụng các thành phần của form trong trang HTML, tuy nhiên để tận dụng ưu điểm của React, cũng như có thể xử lý dễ dàng với dữ liệu, React chọn cách viết các thành phần form bên trong file Javascript, kỹ thuật này được gọi là "controlled component".
- Các giá trị thay đổi trong form sẽ được giữ trong State của component, và được cập nhật thay đổi thông qua setState.
- Ta lần lượt xem các ví dụ về <input />, <textarea>, <select><option>

Làm việc với Input

```
<div id="root"></div>

<script type="text/babel">
  class ActionButton extends React.Component {
    constructor(props) {
      super(props);
      this.state = {value: ''};
    };

    handleChange = (event) => {
      this.setState({value: event.target.value});
    };

    handleSubmit = (event) => {
      alert('Giá trị đã được submit: ' +
this.state.value);
      event.preventDefault();
    };

    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <label>
            Name:
            <input type="text" value={this.state.value}
onChange={this.handleChange} />
          </label>
          <input type="submit" value="Submit" />
        </form>
      );
    };
  };

  ReactDOM.render(
    <ActionButton />,
    document.getElementById('root')
  );
```

```
</script>
```

- Ban đầu value của <input /> sẽ lấy giá trị khởi tạo của state là rỗng.
- onChange={this.handleChange} sẽ sử dụng setState để lấy giá trị thay đổi của <input />.
- Khi nhấn Submit sẽ kích hoạt onSubmit của <form>, khi này sẽ tiến hành xử lý function handleSubmit, là gọi một thông báo với giá trị vừa nhập.

Làm việc với Textarea

```
<div id="root"></div>

<script type="text/babel">
  class ActionButton extends React.Component {
    constructor(props) {
      super(props);
      this.state = {value: 'Your message here'};
    };

    handleChange = (event) => {
      this.setState({value: event.target.value});
    };

    handleSubmit = (event) => {
      alert('Giá trị đã được submit: ' +
this.state.value);
      event.preventDefault();
    };

    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <label>
            Message:
            <textarea type="text"
value={this.state.value} onChange={this.handleChange} />
          </label>
```

```
          <input type="submit" value="Submit" />
        </form>
      );
    };
  };

  ReactDOM.render(
    <ActionButton />,
    document.getElementById('root')
  );
</script>
```

Làm việc với Select

- Cách xử lý cho <select><option> có một điểm khác biệt hơn so với xử lý bên file HTML, đó là ta sẽ sử dụng thuộc tính value ngay tại <select>, việc này sẽ tiện lợi ở chỗ: State chỉ cần tương tác với một value của <select>.
- Phần xử lý dữ liệu sẽ giống như xử lý cho <input />.

```
<div id="root"></div>

<script type="text/babel">
  class ActionButton extends React.Component {
    constructor(props) {
      super(props);
      this.state = {value: 'javascript'};
    };

    handleChange = (event) => {
      this.setState({value: event.target.value});
    };

    handleSubmit = (event) => {
      alert('Giá trị đã được submit: ' +
this.state.value);
      event.preventDefault();
    };
```

```
    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <label>
            Chọn khóa học:
            <select value={this.state.value}
onChange={this.handleChange}>
              <option value="html">HTML</option>
              <option value="css">CSS</option>
              <option
value="javascript">JAVASCRIPT</option>
              <option value="react.js">REACT.JS</option>
            </select>
          </label>
          <input type="submit" value="Submit" />
        </form>
      );
    };
  };

  ReactDOM.render(
    <ActionButton />,
    document.getElementById('root')
  );
</script>
```

Bài 16: Xử lý form kết hợp nhiều thành phần trong React.js

Bài học này sẽ giúp các bạn biết cách kết hợp xử lý các thành phần trong <form>

Xử lý 2 thành phần input

```
<div id="root"></div>

<script type="text/babel">
  class ActionButton extends React.Component {
    constructor(props) {
      super(props);
```

```
      this.state = {
        isGoing: true,
        guestName: "Bùi Văn Tèo"
      };
    };

    handleInputChange = (event) => {
      const target = event.target;
      const value = target.type === 'checkbox' ?
target.checked : target.value;
      const name = target.name;

      this.setState({
        [name]: value
      });
    }

    handleSubmit = (event) => {
      alert('Giá trị đã được submit: '
        + this.state.isGoing + ' và '
        + this.state.guestName
      );
      event.preventDefault();
    };

    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <label>
            Tham gia:
            <input
              name="isGoing"
              type="checkbox"
              checked={this.state.isGoing}
              onChange={this.handleInputChange} />
          </label>
          <br />

          <label>
```

```
            Ghi rõ họ tên:
            <input
              name="guestName"
              type="text"
              value={this.state.guestName}
              onChange={this.handleInputChange} />
          </label>
          <input type="submit" value="Submit" />
        </form>
      );
    };
  };

  ReactDOM.render(
    <ActionButton />,
    document.getElementById('root')
  );
</script>
```

- Ở phần constructor khai báo 2 giá trị đầu tiên của <input type="checkbox" /> và <input type="text" />.
- Trong function handleInputChange xử lý giá trị true và false của <input type="checkbox" /> đồng thời lấy giá trị name của <input type="text" />.
- Cũng trong function handleInputChange sử dụng setState để thực hiện cập nhật giá trị mới cho State.
- Function handleSubmit giữ vai trò gọi 2 giá trị mới của <input type="checkbox" /> và <input type="text" />.

Xử lý nhiều thành phần trong form React.js

Tương tự như trên, dựa theo name và value ta có thể sử lý nhiều thành phần của <form> cùng lúc như sau

```
<div id="root"></div>

<script type="text/babel">
  class ActionButton extends React.Component {
    constructor(props) {
```

```
      super(props);
      this.state = {
        isGoing: true,
        guestName: 'Bùi Văn Tèo',
        course: 'react.js',
        message: 'Your message here'
      };
    };

    handleInputChange = (event) => {
      const target = event.target;
      const value = target.type === 'checkbox' ?
target.checked : target.value;
      const name = target.name;

      this.setState({
        [name]: value
      });
    }

    handleSubmit = (event) => {
      alert('Tham gia: '
        + this.state.isGoing + ', họ tên: '
        + this.state.guestName + ', '
        + this.state.course + ' và '
        + this.state.message
      );
      event.preventDefault();
    };

    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <label>
            Tham gia:
            <input
              name="isGoing"
              type="checkbox"
              checked={this.state.isGoing}
```

```
            onChange={this.handleInputChange} />
          </label>
          <br />

          <label>
            Ghi rõ họ tên:
            <input
              name="guestName"
              type="text"
              value={this.state.guestName}
              onChange={this.handleInputChange} />
          </label>
          <br />

          <label>
            Chọn khóa học:
            <select
              name="course"
              value={this.state.course}
              onChange={this.handleInputChange}
            >
              <option value="html">HTML</option>
              <option value="css">CSS</option>
              <option
value="javascript">JAVASCRIPT</option>
              <option value="react.js">REACT.JS</option>
            </select>
          </label>
          <br />

          <label>
            Message:
            <textarea
              name="message"
              type="text"
              value={this.state.message}
              onChange={this.handleInputChange}
            />
          </label>
```

```
          <br />

          <input type="submit" value="Submit" />
        </form>
      );
    };
  };

  ReactDOM.render(
    <ActionButton />,
    document.getElementById('root')
  );
</script>
```

Bài 17: Render có điều kiện trong React.js

- Trong React, chúng ta có thể tạo nhiều component khác nhau, khi đó có thể render bất kỳ component nào ta muốn, bằng cách sử dụng điều kiện tại phần render.
- Cách sử dụng câu điều kiện (câu điều kiện if else) tại phần render giống như cách sử dụng trong Javascript, React sẽ dựa vào câu điều kiện để tạo thành phần đại diện cho State hiện tại, sau đó sẽ dựa vào setState để cập nhập lại giao diện.

Thực hành Render có điều kiện

Chúng ta sẽ tạo một ứng dụng sao cho:
- Nếu chưa login (hoặc logout) thì sẽ hiển thị "Please sign in".
- Nếu đã login thì sẽ hiển thị "Welcome back!".

Ta lần lượt thực hiện các bước sau:
- Tạo 2 component UserGreeting (hiển thị nội dung khi login) và GuestGreeting (hiển thị nội dung khi chưa login hoặc logout).
- Tạo 1 function Greeting điều khiển 2 component User và Guess tuy theo điều kiện user có login hay không.
- Tạo 2 component hiện thị nội dung cho button: LogoutButton khi đã login và LoginButton khi chưa login hoặc logout.
- Tạo 2 function handleLoginClick và handleLogoutClick để xử lý click button.
- Việc còn lại cuối cùng là tiến hành xử lý câu điều kiện bên trong

phần render để trả đúng trạng thái user có login hay không.

2 component UserGreeting và GuestGreeting

```
const UserGreeting = (props) => {
  return <h2>Welcome back!</h2>;
}

const GuestGreeting = (props) => {
  return <h2>Please sign in.</h2>;
}
```

Function Greeting

Nội dung function này lọc điều kiện login:

- Nếu login thì trả về component UserGreeting
- Nếu chưa login hoặc logout thì trả về component GuestGreeting

```
const Greeting = (props) => {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

2 component LoginButton và LogoutButton

Nội dung 2 component này dùng để điều khiển hiển thị của button: nếu là login thì button sẽ hiển thị Logout và ngược lại.

```
const LoginButton = (props) => {
  return (
    <button onClick={props.onClick}>
      Login
    </button>
  );
}
```

```
const LogoutButton = (props) => {
  return (
    <button onClick={props.onClick}>
      Logout
    </button>
  );
}
```

2 function handleLoginClick và handleLogoutClick

Nội dung 2 function này sử dụng setState để cập nhật trạng thái login hay logout.

```
handleLoginClick = () =>  {
this.setState({isLoggedIn: true});
}

handleLogoutClick = () =>  {
this.setState({isLoggedIn: false});
}
```

Tổng hợp code, ta được kết quả

```
<div id="root"></div>

<script type="text/babel">
  class LoginControl extends React.Component {
    constructor(props) {
      super(props);
      this.state = {isLoggedIn: false};
    };

    handleLoginClick = () => {
      this.setState({isLoggedIn: true});
    };
```

```
    handleLogoutClick = () => {
      this.setState({isLoggedIn: false});
    };

    render() {
      const isLoggedIn = this.state.isLoggedIn;
      let button; /* Khai báo biến button sử dụng cho if
*/

      if (isLoggedIn) {
        button = <LogoutButton
onClick={this.handleLogoutClick} />;
      } else {
        button = <LoginButton
onClick={this.handleLoginClick} />;
      }

      return (
        <div>
          <Greeting isLoggedIn={isLoggedIn} />
          {button}
        </div>
      );
    };
  };

  const UserGreeting = (props) => {
    return <h2>Welcome back!</h2>;
  };

  const GuestGreeting = (props) => {
    return <h2>Please sign up.</h2>;
  };

  const Greeting = (props) => {
    const isLoggedIn = props.isLoggedIn;
    if (isLoggedIn) {
      return <UserGreeting />;
    }
```

```
    return <GuestGreeting />;
  };

  const LoginButton = (props) => {
    return (
      <button onClick={props.onClick}>
        Login
      </button>
    );
  };

  const LogoutButton = (props) => {
    return (
      <button onClick={props.onClick}>
        Logout
      </button>
    );
  };

  ReactDOM.render(
    <LoginControl />,
    document.getElementById('root')
  );
</script>
```

Render có điều kiện bên trong cấu trúc JSX

Render có điều kiện có thể được viết bên trong một cấu trúc JSX, như sau:

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' :
'not'}</b> logged in.
    </div>
  );
}
```

Hoặc viết theo cách sau:

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn ? (
        <LogoutButton onClick={this.handleLogoutClick} />
      ) : (
        <LoginButton onClick={this.handleLoginClick} />
      )}
    </div>
  );
}
```

Bài 18: Chuyển Array - Object thành danh sách

Dữ liệu dưới dạng Array, Object làm sao có thể chuyển đổi sang cấu trúc là một danh sách (list) của HTML? nội dung bài học này sẽ giúp các bạn làm được việc đó

Render giá trị Array

Để lấy giá trị là một Array, đối với Javascript ta làm như sau:

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => number * 2);
console.log(doubled);
```

Tương tự như vậy, để render giá trị Array ra một list ta làm như sau:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
    <li>{number * 2}</li>
);

ReactDOM.render(
  <ul>{listItems}</ul>,
```

```
  document.getElementById('root')
);
```

Render giá trị Array viết dạng component

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li>{number * 2}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Ta thấy kết quả vẫn hiển thị như mong muốn, tuy nhiên lúc này khi xem output console (Nhấn F12), thì sẽ thấy thông báo Warning: Each child in a list should have a unique "key" prop..

Thuộc tính key cần được thêm khi tạo list, xem bài tiếp theo nhé

Bài 19: List và keys

- key giúp React xác định item nào được thay đổi, được thêm, hoặc được xóa.
- key dành cho mỗi phần tử trong list, do đó keys của mỗi phần tử phải là duy nhất.

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
```

```
    <li key={number.toString()}>
      {number * 2}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
};

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

- Xem lại kết quả console khi này, ta sẽ thấy không còn cảnh báo nữa.
- Do key mỗi phần tử là duy nhất, nên key thường là giá trị id, trường hợp không có id hoặc giá trị nào đó duy nhất thì ta có thể sử dụng index như là giải pháp cuối cùng.

Sử dụng id làm keys

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);
```

Sử dụng index làm keys

- Sử dụng trong trường hợp Array không có giá trị duy nhất như id.
- Tuy nhiên không khuyến khích sử dụng index trong trường hợp thứ tự các mục có khả năng thay đổi, vì có khả năng sẽ làm ảnh hưởng đến State của component.

```
const todoItems = todos.map((todo, index) =>
```

```
  <li key={index}>
    {todo.text}
  </li>
);
```

Trích xuất component với Keys

Key chỉ có ý nghĩa trong ngữ cảnh có các mảng đang sử dụng, ví dụ như nếu trích xuất component ListItem, thì ta nên gán key ngay tại <ListItem key={} /> thay vì tại <li> bên trong function ListItem.

Cấu trúc đúng	Cấu trúc không đúng

`function ListItem(props) {`
`  const value =`
`props.value;`
`  return (`
`    /* Sai: không cần key ở`
`đây */`
`    <li`
`key={value.toString()}>`
`      {value}`
`    </li>`
`  );`
`};`
 	
`function NumberList(props)`
`{`
`  const numbers =`
`props.numbers;`
`  const listItems =`
`numbers.map((number) =>`
`    /* Sai: key nên được sử`
`dụng ở đây: */`
`    <ListItem value={number`
`* 2} />`
`  );`
`  return (`
	`function ListItem(props) {`
	`  return`
	`<li>{props.value}</li>;`
	`};`
	
	`function NumberList(props)`
	`{`
	`  const numbers =`
	`props.numbers;`
	`  const listItems =`
	`numbers.map((number) =>`
	`    /* Sử dụng đúng */`
	`    <ListItem`
	`key={number.toString()}`
	`value={number`
	`* 2} />`
	`  );`
	`  return (`
	`    <ul>`
	`      {listItems}`
	`    </ul>`
	`  );`
	`};`
	
	`const numbers = [1, 2, 3,`

```
    <ul>
      {listItems}
    </ul>
  );
};

const numbers = [1, 2, 3,
4, 5];
ReactDOM.render(
  <NumberList
numbers={numbers} />,

document.getElementById('ro
ot')
);
```

```
4, 5];
ReactDOM.render(
  <NumberList
numbers={numbers} />,

document.getElementById('ro
ot')
);
```

Keys phải giống nhau trong cùng một component

Key cần phải đồng nhất khi sử dụng trong cùng một component (hoặc trong cùng một function hay class).

```
function Tutorial(props) {
  const sidebar = (
    <ul>
      {props.posts.map((post) =>
        <li key={post.id}>
          {post.title}
        </li>
      )}
    </ul>
  );
  const content = props.posts.map((post) =>
    <div key={post.id}>
      <h2>{post.title}</h2>
      <p>{post.content}</p>
    </div>
  );
  return (
    <div>
```

```
      {sidebar}
      <hr />
      {content}
    </div>
  );
};

const posts = [
  {id: 1, title: 'Học Javascript', content: 'Chào mừng bạn đến Javascript!'},
  {id: 2, title: 'Học React.js', content: 'Bạn có thể bắt đầu học React.js sau khi cài đặt.'}
];
ReactDOM.render(
  <Tutorial posts={posts} />,
  document.getElementById('root')
);
```

Lưu ý: Key không phải là một giá trị của component, do đó nếu muốn sử dụng giá trị giống như key thì ta cần phải tạo một props, sử dụng với tên khác

```
const content = posts.map((post) =>
  <Post
    key={post.id}
    id={post.id}
    title={post.title} />
);
```

Bài 20: Nhúng map() và JSX

Thay vì khai báo một map() tách riêng phần render, ta hoàn toàn có thể sử dụng bên trong cấu trúc JSX của phần render.

Cùng phân tích qua bảng sau:

Map bên ngoài Render	Map bên trong JSX của Render

```
function ListItem(props) {
  return
<li>{props.value}</li>;
};

function NumberList(props)
{
  const numbers =
props.numbers;
  const listItems =
numbers.map((number) =>
    <ListItem
key={number.toString()}
                value={number
* 2} />
  );
  return (
    <ul>
      {listItems}
    </ul>
  );
};
```

```
function ListItem(props) {
  return
<li>{props.value}</li>;
};

function NumberList(props)
{
  const numbers =
props.numbers;
  return (
    <ul>
      {numbers.map((number)
=>
        <ListItem
key={number.toString()}

value={number * 2} />
      )}
    </ul>
  );
};
```

Ta thấy JSX cho phép chúng ta nhúng một biểu thức bên trong cặp dấu {}, việc này giúp code rõ ràng hơn.

Bài 21: State với map()

Bài học này sẽ áp dụng map() vào một state thực tế, mục đích giúp các bạn hình dung rõ hơn việc ứng dụng map() để list ra một danh sách từ dữ liệu ban đầu.

```
<div id="member"></div>

<script type="text/babel">
  class Member extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
```

```
        users: [
          {
            id: 1,
            name: "Bùi Văn Tèo",
            age: 25
          },
          {
            id: 2,
            name: "Trần Văn An",
            age: 29
          },
          {
            id: 3,
            name: "Nguyễn Thị Bưởi",
            age: 22
          }
        ],
        job: "IT Communicator"
      };
    };

    render() {
      return (
        <div>
          <h2>Tên: </h2>
          <p>Tuổi: </p>
        </div>
      );
    };
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

Ta sẽ áp dụng map() để list ra một danh sách bằng cách viết lại render như sau

```
render() {
    return (
      <div>
        <ul>
          {this.state.users.map(item => (
            <li key={item.id}>
              <h2>Tên: {item.name}</h2>
              <p>Tuổi: {item.age}</p>
            </li>
          ))}
        </ul>
      </div>
    );
  };
```

Bài 22: setState với map()

Bài học này sẽ giúp các bạn biết cách áp dụng map() vào setState để cập nhật một mục bất kỳ trong một mảng cho trước.

```
<div id="member"></div>

<script type="text/babel">
  class Member extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        users: [
          {
            id: 1,
            name: "Bùi Văn Tèo",
            age: 25
          },
          {
            id: 2,
            name: "Trần Văn An",
            age: 29
          },
          {
```

```
            id: 3,
            name: "Nguyễn Thị Bưởi",
            age: 22
          }
        ],
        job: "IT Communicator"
      };
    };

    render() {
      return (
      <div>
        <ul>
          {this.state.users.map(item => (
            <li key={item.id}>
              <h2>Tên: {item.name}</h2>
              <p>Tuổi: {item.age}</p>
            </li>
          ))}
        </ul>
      </div>
      );
    };
  };

  ReactDOM.render(
    <Member />,
    document.getElementById('member')
  );
</script>
```

Để cập nhật nội dung trên, ta cần một <button> và một function xử lý nội dung cập nhật, ta thêm nội dung trên như sau:

```
changeItem = () => {
    this.setState(

    );
  };
```

```
  render() {
    return (
      <div>
        <ul>
          {this.state.users.map(item => (
            <li key={item.id}>
              <h2>Tên: {item.name}</h2>
              <p>Tuổi: {item.age}</p>
            </li>
          ))}
        </ul>
        <button type="button"
onClick={this.changeItem}>Update</button>
      </div>
    );
  };
```

- Giả sử ta cần cập nhật name và age ở mục có id: 2.
- Ta chỉnh lại function changeItem như sau:

```
 changeItem = () => {
    let key = 2;

    this.setState(prevState => ({
      users: prevState.users.map(
        item => item.id === key? {
          ...item,
          name: 'Dương Minh Trí',
          age: 4
        }: item
      )
    }));
  };
```

- let key = 2;, thực tế trong việc update, khi click vào liên kết update sẽ trả về giá trị id của mục cần được update, key này sẽ đại diện cho id đó.
- item.id === key, nếu item.id bằng với key thì tiến hành cập nhật

nội dung, không thì trả về item.

- Ta thấy nội dung có id: 2 đã được cập nhật khi click vào button.
- Nội dung bài này sẽ được áp dụng rất nhiều khi bạn muốn cập nhật một mảng bất kỳ