REACTJS 01

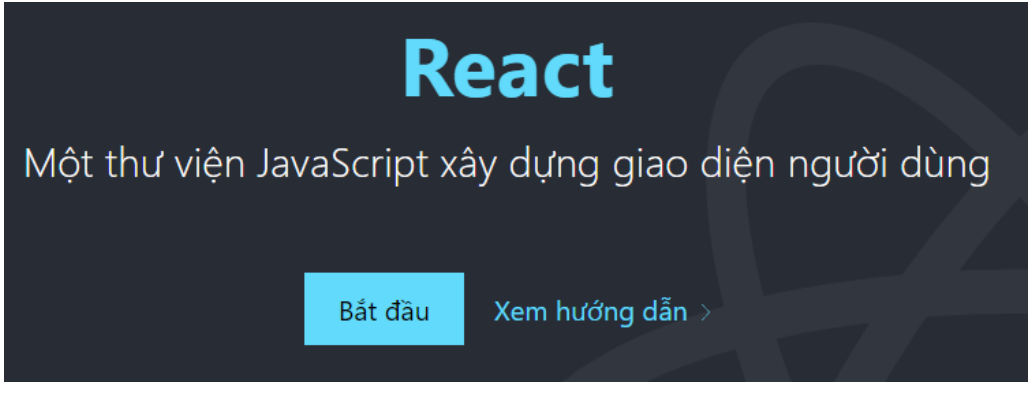
FDP 5.0

Nội dung

- Giới thiệu
- ES6
- React Render HTML
- JSX (Javascript XML)
- Component
- Prop & State

GIỚI THIỆU

- Một thư viện JavaScript xây dựng giao diện người dùng
- Được sử dụng để xây dựng các SPA (Single Page Application)
- Cho phép người dùng tạo các các thành phần UI(User Interface) có thể tái sử dụng.



GIỚI THIỆU

- Tài liệu chính thức tại https://reactjs.org/ (EN) hoặc https://vi.reactjs.org/ (VI)
- Để sử dụng ReactJS có 2 cách
  - Nhúng trực tiếp script của ReactJS vào trong mã HTML
  - Cài đặt môi trường, bao gồm: Node.js và NPM (Node Package Manager)

GIỚI THIỆU

Cách 1

- Nhúng các đoạn script sau vào file *.html

```
<script src="https://unpkg.com/react@16/umd/react.production.min.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>
<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
```

- Tạo 1 thẻ div để bọc toàn bộ mã ReactJS render ra.

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

- Một đoạn script phía cuối cùng trước thẻ body để thực hiện ứng dụng ReactJS

GIỚI THIỆU

Cách 1

```
<!DOCTYPE html>
<html>
  <script src="https://unpkg.com/react@16/umd/react.production.min.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
  <body>

    <div id="mydiv"></div>

    <script type="text/babel">
      class Hello extends React.Component {
        render() {
          return <h1>Hello World!</h1>
        }
      }

      ReactDOM.render(<Hello />, document.getElementById('mydiv'))
    </script>
  </body>
</html>
```

GIỚI THIỆU

Cách 2

- Cài đặt môi trường
  - **Tải nodejs tại** https://nodejs.org/en/
  - **Cài đặt NodeJS vào trong máy tính**
- Tại thư mục của Project sử dụng command line để tạo ứng dụng ReactJS.
  - Gõ lệnh: npx create-react-app <<tên_project>>
- Gõ lệnh cd <<tên_project>> để di chuyển đến thư mục project
- Gõ lệnh npm start để khởi chạy project

ES6

- ES6 là viết tắt của ECMAScript 6
- ECMAScript được tạo ra để chuẩn hóa JavaScript và ES6 là phiên bản thứ 6 của ECMAScript, nó được xuất bản vào năm 2015 và còn được gọi là ECMAScript 2015.
- Tại sao nên biết ES6? React sử dụng ES6 và bạn nên làm quen với một số tính năng mới như:
  - Class
  - Arrow function
  - Variable (let, const, var)

- Class là 1 loại function nhưng thay vì dùng từ khóa **function** để khởi tạo nó, thì chúng ta sử dụng từ khóa **class** và các thuộc tính của nó được gán bên trong 1 phương thức có tên là **constructor()**
- **Ví dụ**

```
class Car {
  constructor(name) {
    this.brand = name;
  }
}

mycar = new Car("Ford");
```

Lưu ý: phương thức **constructor** được gọi tự động khi đối tượng được khởi tạo

- Một phương thức trong class sẽ chịu trách nhiệm cho một nhiệm vụ bất kỳ của. Phương thức là 1 hàm có thể có hoặc không có tham số đầu vào.
- Ví dụ

```
class Car {
  constructor(name) {
    this.brand = name;
  }

  present() {
    return 'I have a ' + this.brand;
  }
}

mycar = new Car("Ford");
mycar.present();
```

- Để tạo 1 class kế thừa - sử dụng từ khóa **extends**
- Một class kế thừa từ 1 class khác sẽ được thừa kế tất cả phương thức từ class khác đó
- Ví dụ

```
class Model extends Car {
  constructor(name, mod) {
    super(name);
    this.model = mod;
  }
  show() {
      return this.present() + ', it is a ' + this.model
  }
}
mycar = new Model("Ford", "Mustang");
mycar.show();
```

```
class Car {
  constructor(name) {
    this.brand = name;
  }

  present() {
    return 'I have a ' + this.brand;
  }
}
```

- Phương thức **super()** bên trong constructor dùng để chỉ đến class cha (class được kế thừa)
- Bằng cách gọi đến phương thức **super()** bên trong phương thức **constructor** chúng ta có thể sử dụng các thuộc tính và phương thức trong class cha.

- Đây là 1 hàm được tạo ra trong ES6
- Cho phép viết cú pháp hàm trông ngắn gọn hơn.
- Ví dụ:

Trước:

```
hello = function() {
  return "Hello World!";
}
```

Với Arrow Function:

```
hello = () => {
  return "Hello World!";
}
```

- Trước khi ES6 ra đời để khởi tạo 1 biến chúng ta dùng từ khóa **var**.
  - Nếu bạn sử dụng var bên ngoài một chức năng, nó thuộc về phạm vi toàn cục.
  - Nếu bạn sử dụng var bên trong của một chức năng, nó thuộc về chức năng đó.
  - Nếu bạn sử dụng var bên trong một khối, tức là vòng lặp for, thì biến vẫn có sẵn bên ngoài khối đó.
- **Var** có phạm vi chức năng , không phải phạm vi khối .

- **let** có phạm vi là khối. Nghĩa là biến được tạo bằng từ khóa **let** sẽ chỉ có giá trị khi biến đó nằm trong 1 khối, ví dụ như 1 vòng lặp for, hoặc 1 c
- Ví dụ

```
const x = 5.6;
```

```
let x = 5.6;
```

React Render HTML

- Mục tiêu của React là làm cho HTML chỉ render trong một webpage.
- React hiển thị HTML cho trang web bằng cách sử dụng một hàm có tên **ReactDOM.render()**

React Render HTML

- **ReactDOM.render()** hàm này nhận vào 2 đối số
  - HTML code
  - HTML element
- Ví dụ

Hiển thị một đoạn bên trong phần tử "gốc":

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

Kết quả được hiển thị trong `<div id="root">` phần tử:

```
<body>

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

</body>
```

JSX (Javascript XML)

- **JSX** là gì?
  - Viết tắt của JavaScript XML.
  - Cho phép chúng ta viết HTML trong React.
  - Giúp viết và thêm HTML trong React dễ dàng hơn.
- JSX cho phép viết các phần tử HTML bằng JavaScript và đặt chúng trong DOM mà không cần bất kỳ **createElement()** và / hoặc **appendChild()** phương thức nào
- Bạn không bắt buộc phải sử dụng JSX, nhưng JSX giúp bạn viết các ứng dụng React dễ dàng hơn.

JSX (Javascript XML)

❖ Ví dụ so sánh giữa có JSX và không có JSX

JSX:

```
const myelement = <h1>I Love JSX!</h1>;

ReactDOM.render(myelement, document.getElementById('root'));
```

Không có JSX:

```
const myelement = React.createElement('h1', {}, 'I do not use JSX!');

ReactDOM.render(myelement, document.getElementById('root'));
```

JSX (Javascript XML)

- Với JSX, bạn có thể viết các biểu thức bên trong dấu ngoặc nhọn **{ }**
- Biểu thức có thể là một biến React hoặc thuộc tính hoặc bất kỳ biểu thức JavaScript hợp lệ nào khác. JSX sẽ thực thi biểu thức và trả về kết quả.

Thực thi biểu thức `5 + 5` :

```
const myelement = <h1>React is {5 + 5} times better with JSX</h1>;
```

JSX (Javascript XML)

- Để viết HTML trên nhiều dòng, hãy đặt HTML bên trong dấu ngoặc đơn.

Tạo một danh sách với ba mục danh sách:

```
const myelement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
```

JSX (Javascript XML)

- Mã HTML phải được bao bọc trong MỘT phần tử cấp cao nhất.
- Vì vậy, nếu bạn muốn viết hai tiêu đề, bạn phải đặt chúng bên trong một phần tử mẹ, giống như một phần tử **div**
- Ví dụ

Gói hai tiêu đề bên trong một phần tử DIV:

```
const myelement = (
  <div>
    <h1>I am a Header.</h1>
    <h1>I am a Header too.</h1>
  </div>
);
```

JSX (Javascript XML)

- JSX tuân theo các quy tắc XML và do đó các phần tử HTML phải được đóng đúng cách.
- Ví dụ

Đóng các phần tử trống bằng />

```
const myelement = <input type="text" />;
```

Component

- Component là một thành phần độc lập có thể là class hoặc function, chứa các logic code Javascript và trả về mã HTML thông qua 1 hàm **render()**
- Component có 2 loại là **Class Component** và **Function Component**
- 1 component sẽ kế thừa 1 class là **React.Component** và yêu cầu 1 hàm **render()** để xuất ra mã HTML
- Ví dụ

Tạo một thành phần Class được gọi là Car

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}

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

Props và State

- Props là các đối số được truyền vào các thành phần React.
- Props sẽ được truyền qua các component khác nhau như là 1 thuộc tính của HTML
- React Props giống như các đối số hàm trong JavaScript và các thuộc tính trong HTML.

Thí dụ

Thêm thuộc tính "thương hiệu" vào phần tử Xe:

```
const myelement = <Car brand="Ford" />;
```

Thí dụ

Sử dụng thuộc tính thương hiệu trong thành phần:

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand}!</h2>;
  }
}
```

Component

- Đối tượng **state** là nơi bạn lưu trữ các giá trị thuộc tính thuộc về thành phần.
- **State** chỉ tồn tại trong compent đó để sử dụng state ở component khác cần truyền **state** thông qua **props**
- Ví dụ

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {brand: "Ford"};
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

Component

- **State** có thể thay đổi được giá trị. Để thay đổi giá trị của nó, chúng ta sử dụng phương thức: **setState({tên_state: "giá trị"})**

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  changeColor = () => {
    this.setState({color: "blue"});
  }
```

```
render() {
  return (
    <div>
      <h1>My {this.state.brand}</h1>
      <p>
        It is a {this.state.color}
        {this.state.model}
        from {this.state.year}.
      </p>
      <button
        type="button"
        onClick={this.changeColor}
      >Change color</button>
    </div>
  );
  }
}
```

Thank you and
happy learning !!!