**Báo cáo: Các Thành Phần Cơ Bản trong ReactJS**

**Mục Lục**

1. **Giới thiệu về ReactJS**
2. **JSX trong ReactJS**
   * Khái niệm JSX
   * Cách JSX hoạt động
   * Lợi ích của JSX
   * Quy tắc viết JSX
   * Biểu thức trong JSX
   * Tích hợp CSS trong JSX
   * Ví dụ minh họa
3. **Components trong ReactJS**
   * Khái niệm Component
   * Vai trò của Component trong kiến trúc React
   * Phân loại Component
4. **Class Component**
   * Khái niệm
   * Đặc điểm
   * Vòng đời của Class Component
   * Các phương thức phổ biến trong Class Component
   * Cách tạo Class Component
   * Quản lý State và Props trong Class Component
   * Ví dụ minh họa chi tiết
5. **Function Component**
   * Khái niệm
   * Ưu điểm so với Class Component
   * Cách tạo Function Component
   * Hooks trong Function Component (useState, useEffect, useContext)
   * Quản lý Props và State trong Function Component
   * Ví dụ minh họa chi tiết
6. **So sánh Class Component và Function Component**
7. **Các Best Practices khi làm việc với Components**
8. **Lỗi thường gặp với JSX và Components và cách khắc phục**
9. **Kết luận**

**1. Giới thiệu về ReactJS**

ReactJS là một thư viện JavaScript mã nguồn mở do Facebook phát triển để xây dựng giao diện người dùng (UI) cho các ứng dụng web hiện đại. React giúp xây dựng các UI phức tạp bằng cách chia nhỏ thành nhiều thành phần độc lập, dễ quản lý.

**1.1 Đặc điểm nổi bật**

* Tư duy theo Component: Tách UI thành các phần nhỏ, dễ tái sử dụng.
* Virtual DOM: Tăng hiệu suất cập nhật giao diện.
* One-way Data Binding: Dữ liệu chảy một chiều giúp dễ kiểm soát.
* Khả năng tích hợp linh hoạt: Có thể kết hợp với các framework khác như Angular, Vue hoặc thư viện UI.

**1.2 Ứng dụng phổ biến của ReactJS**

* Xây dựng Single Page Application (SPA).
* Xây dựng các ứng dụng web có tính tương tác cao.
* Phát triển giao diện cho các ứng dụng di động thông qua React Native.

**2. JSX trong ReactJS**

**2.1 Khái niệm JSX**

JSX (JavaScript XML) là một cú pháp đặc biệt trong React, cho phép viết cấu trúc HTML ngay trong mã JavaScript. JSX giúp code dễ đọc hơn và tích hợp logic JavaScript trực tiếp vào phần giao diện.

**2.2 Cách JSX hoạt động**

JSX không phải là HTML. Trình biên dịch (Babel) chuyển đổi JSX thành các lệnh React.createElement().

**Ví dụ:**

const element = <h1>Hello, React!</h1>;

// Biên dịch thành:

const element = React.createElement('h1', null, 'Hello, React!');

**2.3 Lợi ích của JSX**

* Dễ đọc, dễ duy trì.
* Hỗ trợ logic JavaScript trong giao diện.
* Giúp code rõ ràng, gọn gàng**.**

**2.4 Quy tắc viết JSX**

* Phải có một phần tử cha bao bọc tất cả.
* Thuộc tính dùng camelCase, ví dụ: className, onClick.
* Biểu thức JavaScript đặt trong { }.

**2.5 Biểu thức trong JSX**

Có thể dùng biểu thức như biến, hàm, toán tử điều kiện trong { }.  
**Ví dụ:**

const name = 'ReactJS';

const element = <h1>{name ? `Xin chào ${name}` : 'Chào bạn'}</h1>;

**2.6 Tích hợp CSS trong JSX**

**Có 3 cách:**

* Inline style: style={{color: 'red'}}
* Import CSS file: import './style.css'
* CSS Module: import styles from './style.module.css'

**3. Components trong ReactJS**

**3.1 Khái niệm Component**

Component là đơn vị cơ bản trong React, cho phép chia nhỏ giao diện thành các phần độc lập có thể tái sử dụng.

**3.2 Vai trò**

* Tăng tính tái sử dụng.
* Dễ bảo trì.
* Giúp tách logic và giao diện rõ ràng.

**3.3 Phân loại**

* Class Component: Được định nghĩa bằng class, hỗ trợ state và lifecycle.
* Function Component: Định nghĩa bằng function, dùng Hooks để quản lý state và lifecycle.

**4. Class Component**

**4.1 Khái niệm**

Class Component là component được định nghĩa bằng cú pháp class của ES6 và kế thừa từ React.Component.

**4.2 Đặc điểm**

* Có thể dùng state.
* Có các phương thức vòng đời.

**4.3 Vòng đời Class Component**

* Mounting: constructor, render, componentDidMount
* Updating: shouldComponentUpdate, render, componentDidUpdate
* Unmounting: componentWillUnmount

**4.4 Các phương thức phổ biến**

* render()
* setState()
* componentDidMount()

**4.5 Cách tạo Class Component**

class Welcome extends React.Component {

render() {

return <h1>Xin chào {this.props.name}</h1>;

}

}

4.6 Quản lý State và Props

* State: Dữ liệu bên trong component, có thể thay đổi.
* Props: Dữ liệu truyền từ component cha sang component con**.**

**4.7 Ví dụ minh họa chi tiết**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

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

render() {

return (

<div>

<p>Giá trị: {this.state.count}</p>

<button onClick={this.increment}>Tăng</button>

</div>

);

}

}

**5. Function Component**

**5.1 Khái niệm**

Function Component là component được định nghĩa bằng hàm, trả về JSX.

**5.2 Ưu điểm**

* Dễ viết, dễ hiểu.
* Hiệu suất tốt.

**5.3 Hooks trong Function Component**

* useState: Quản lý state.
* useEffect: Quản lý side effect.
* useContext: Truyền dữ liệu toàn cục.

**5.4 Ví dụ chi tiết**

function Counter() {

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

useEffect(() => console.log('Cập nhật:', count), [count]);

return (

<div>

<p>Giá trị: {count}</p>

<button onClick={() => setCount(count + 1)}>Tăng</button>

</div>

);

}

**6. So sánh Class Component và Function Component**

| Tiêu chí | Class Component | Function Component |
| --- | --- | --- |
| Cú pháp | Dài hơn | Ngắn gọn |
| State | Có | Có (Hooks) |
| Lifecycle | Có phương thức riêng | Dùng Hooks |
| Hiệu suất | Thấp hơn | Cao hơn |

**7. Best Practices khi dùng Components**

* Tách component thành các file riêng.
* Đặt tên component chữ cái đầu viết hoa.
* Dùng PropTypes để kiểm tra dữ liệu.
* Giảm thiểu logic trong JSX.

**8. Lỗi thường gặp**

* Sai đường dẫn import.
* Quên export default component.
* Không có thẻ cha bao bọc trong JSX.

**9. Kết luận**

JSX và Components là hai yếu tố quan trọng trong React. Hiểu rõ cách hoạt động và sử dụng hợp lý sẽ giúp phát triển ứng dụng web mạnh mẽ, dễ bảo trì và mở rộng.