**TIÊU ĐỀ BÁO CÁO: NGHIÊN CỨU CÁC THÀNH PHẦN CƠ BẢN TRONG REACTJS: JSX VÀ COMPONENTS**

**LỜI MỞ ĐẦU**

ReactJS, thường được gọi đơn giản là React, là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook (nay là Meta) vào năm 2013. Nó nhanh chóng trở thành một trong những lựa chọn hàng đầu cho việc xây dựng giao diện người dùng (UI), đặc biệt là các ứng dụng trang đơn (SPA) với khả năng mở rộng và hiệu suất cao. Sức mạnh của React nằm ở mô hình component-based (dựa trên thành phần) và cách nó quản lý DOM ảo (Virtual DOM). Báo cáo này sẽ đi sâu vào hai khái niệm nền tảng và cốt lõi nhất của React: **JSX** và **Components** (Class Component và Function Component), giúp người đọc có cái nhìn toàn diện để bắt đầu với thư viện này.

**PHẦN 1: JSX - JavaScript XML**

**1.1. JSX là gì?**

JSX là một cú pháp mở rộng của JavaScript, cho phép lập trình viên viết mã HTML ngay bên trong code JavaScript. Về bản chất, nó cung cấp một cách trực quan và dễ đọc để mô tả giao diện người dùng (UI) sẽ được hiển thị.

Ví dụ cơ bản:

jsx

const element = <h1>Xin chào, thế giới!</h1>;

Dòng code trên không phải là chuỗi hay HTML thuần túy. Nó được gọi là JSX.

**1.2. Tại sao React sử dụng JSX?**

* **Trực quan hơn:** JSX cho phép nhúng cấu trúc UI (trông giống HTML) vào logic (JavaScript), giúp code dễ hình dung và bảo trì hơn so với việc sử dụng các hàm thuần JavaScript như React.createElement().
* **Năng suất cao hơn:** Cú pháp ngắn gọn và quen thuộc với HTML giúp developer viết code nhanh hơn.
* **Tận dụng sức mạnh của JavaScript:** Ngay bên trong JSX, bạn có thể sử dụng toàn bộ sức mạnh của JavaScript (như biến, biểu thức, hàm) để tạo ra UI động.

**1.3. Cú pháp JSX cơ bản**

**a. Nhúng biểu thức JavaScript vào JSX**  
Bạn có thể nhúng bất kỳ biểu thức JavaScript hợp lệ nào vào JSX bằng cách đặt nó trong cặp dấu ngoặc nhọn {}.

jsx

const name = 'Tuấn Anh';

const element = <h1>Xin chào, {name}!</h1>; *// Hiển thị: Xin chào, Tuấn Anh!*

const user = { firstName: 'Nguyễn', lastName: 'Văn A' };

const greeting = <p>Xin chào, {user.firstName} {user.lastName}!</p>;

const number = 10;

const mathElement = <p>{number} \* 2 = {number \* 2}</p>; *// Hiển thị: 10 \* 2 = 20*

**b. Chỉ định thuộc tính với JSX**

* Sử dụng dấu ngoặc kép "" cho giá trị chuỗi.
* Sử dụng dấu ngoặc nhọn {} cho giá trị biểu thức JavaScript.
* Thuộc tính class trong HTML trở thành className trong JSX (vì class là từ khóa trong JavaScript).
* Thuộc tính for trong HTML trở thành htmlFor.

jsx

*// Giá trị chuỗi*

const element1 = <div className="container">Nội dung</div>;

*// Giá trị biểu thức*

const avatarUrl = 'https://example.com/avatar.png';

const element2 = <img src={avatarUrl} alt="Avatar" />;

*// Sử dụng className thay vì class*

const element3 = <p className="text-red-500">Đoạn văn có màu đỏ</p>;

**c. JSX là một biểu thức**  
Sau khi biên dịch, các biểu thức JSX trở thành các lời gọi hàm JavaScript thông thường (React.createElement) và được đánh giá thành các đối tượng JavaScript thuần. Điều này có nghĩa là bạn có thể sử dụng JSX bên trong các câu lệnh if, for, gán cho biến, chấp nhận làm đối số của hàm, và trả về từ hàm.

jsx

function getGreeting(user) {

if (user) {

return <h1>Xin chào, {user}!</h1>;

}

return <h1>Xin chào, Người lạ.</h1>;

}

**d. JSX Children**  
Một thẻ JSX có thể chứa các thẻ con (children), giống như HTML.

jsx

const element = (

<div className="my-container">

<h1>Tiêu đề</h1>

<p>Đây là một đoạn văn bản.</p>

<ul>

<li>Mục 1</li>

<li>Mục 2</li>

</ul>

</div>

);

*Lưu ý: Khi viết JSX trên nhiều dòng, nên đặt toàn bộ biểu thức trong cặp ngoặc tròn () để tránh lỗi "Automatic Semicolon Insertion" của JavaScript.*

**1.4. Các quy tắc quan trọng**

* **Chỉ trả về một phần tử gốc:** Một component phải trả về một phần tử JSX duy nhất. Để bao nhiều phần tử, bạn phải bọc chúng trong một thẻ cha (ví dụ: <div>, <>) hoặc sử dụng React.Fragment (<>...</>).
* **Đóng tất cả các thẻ:** Mọi thẻ trong JSX đều phải được đóng, kể cả thẻ tự đóng (self-closing tags). Ví dụ: <img />, <br />, <input />.

**PHẦN 2: COMPONENTS - THÀNH PHẦN**

**2.1. Component là gì?**

Components là các khối xây dựng độc lập, có thể tái sử dụng để xây dựng giao diện người dùng. Mỗi component về cơ bản là một hàm hoặc lớp JavaScript trả về (hoặc render) các phần tử React (JSX) để mô tả những gì sẽ xuất hiện trên màn hình.

Ví dụ: Bạn có thể tách một trang web thành các component như Header, Sidebar, ArticleList, UserCard, Footer.

**Ưu điểm:**

* **Tái sử dụng:** Một component có thể được sử dụng nhiều lần ở nhiều nơi với các dữ liệu (props) khác nhau.
* **Tách biệt trách nhiệm (Separation of Concerns):** Mỗi component quản lý logic và giao diện riêng của nó, giúp code dễ hiểu, dễ bảo trì và kiểm thử hơn.
* **Tổ chức code tốt hơn:** Ứng dụng lớn được chia nhỏ thành các phần độc lập, dễ quản lý.

**2.2. Phân loại Component**

React có hai loại component chính: **Class Components** và **Function Components**.

**PHẦN 3: CLASS COMPONENTS**

**3.1. Định nghĩa**

Class Component là cách truyền thống để tạo component trong React trước khi Hooks ra đời. Nó là một lớp ES6 (JavaScript class) kế thừa từ React.Component.

jsx

import React, { Component } from 'react';

class Welcome extends Component {

render() {

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

}

}

export default Welcome;

**3.2. Đặc điểm và tính năng**

* **Phương thức**render()**:** Bắt buộc phải có. Phương thức này trả về JSX để hiển thị.
* **State (Trạng thái):** Class Component có thể quản lý trạng thái nội bộ (dữ liệu có thể thay đổi) thông qua đối tượng this.state và phương thức this.setState().
* **Props (Thuộc tính):** Nhận dữ liệu từ component cha thông qua this.props.
* **Vòng đời (Lifecycle Methods):** Có thể sử dụng các phương thức vòng đời như componentDidMount, componentDidUpdate, componentWillUnmount để thực hiện các tác vụ (gọi API, subscribe, xử lý sự kiện) tại các thời điểm cụ thể.

**3.3. Ví dụ minh họa đầy đủ**

jsx

import React, { Component } from 'react';

class Counter extends Component {

*// 1. Khởi tạo State trong constructor*

constructor(props) {

super(props);

this.state = {

count: 0

};

*// Ràng buộc (bind) ngữ cảnh 'this' cho phương thức*

this.increment = this.increment.bind(this);

}

*// 2. Phương thức để cập nhật State*

increment() {

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

}

*// 3. Phương thức vòng đời: chạy sau khi component được render lần đầu*

componentDidMount() {

console.log('Component đã được mount!');

}

*// 4. Phương thức render bắt buộc*

render() {

return (

<div>

<p>Bạn đã bấm {this.state.count} lần</p>

{*/\* Gọi phương thức khi click, sử dụng this.increment đã được bind \*/*}

<button onClick={this.increment}>

Bấm vào tôi

</button>

</div>

);

}

}

export default Counter;

**PHẦN 4: FUNCTION COMPONENTS**

**4.1. Định nghĩa**

Function Component đơn giản là một hàm JavaScript (có thể là function thông thường hoặc arrow function) nhận props làm đối số và trả về JSX.

jsx

*// Arrow Function (Phổ biến)*

const Welcome = (props) => {

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

};

*// Function Declaration*

function Welcome(props) {

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

}

export default Welcome;

**4.2. Lịch sử và sự phát triển**

Trước đây, Function Component được gọi là "Stateless Components" vì chúng không thể quản lý state hay sử dụng các phương thức vòng đời. Chúng chỉ dùng để hiển thị UI dựa trên props nhận được.

Tuy nhiên, kể từ khi **React Hooks** được giới thiệu trong phiên bản 16.8 (2019), Function Component đã có đầy đủ sức mạnh của Class Component. Chúng có thể sử dụng state, side effects, context, và mọi tính năng khác thông qua các Hook.

**4.3. React Hooks - "Siêu năng lực" cho Function Component**

Hooks là các hàm đặc biệt cho phép bạn "móc" vào các tính năng của React (như state và lifecycle) từ within một Function Component.

**a.**useState**Hook:**  
Cho phép thêm state vào Function Component.

jsx

import React, { useState } from 'react';

function Counter() {

*// Khai báo một biến state mới, gọi là "count"*

*// useState trả về một cặp [currentState, functionToUpdateState]*

const [count, setCount] = useState(0); *// Giá trị khởi tạo là 0*

return (

<div>

<p>Bạn đã bấm {count} lần</p>

{*/\* Cập nhật state bằng hàm setCount \*/*}

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

Bấm vào tôi

</button>

</div>

);

}

export default Counter;

**b.**useEffect**Hook:**  
Cho phép thực hiện "side effects" (các tác vụ phụ) trong Function Component, thay thế cho componentDidMount, componentDidUpdate, và componentWillUnmount.

jsx

import React, { useState, useEffect } from 'react';

function UserProfile({ userId }) {

const [user, setUser] = useState(null);

*// Effect này chạy sau mỗi lần render, nhưng chỉ gọi API nếu userId thay đổi*

useEffect(() => {

*// Hàm để fetch dữ liệu user*

const fetchUser = async () => {

const response = await fetch(`/api/users/${userId}`);

const userData = await response.json();

setUser(userData);

};

fetchUser();

*// Hàm cleanup (optional): chạy trước khi component unmount hoặc trước effect chạy lại*

return () => {

*// Hủy các subscription hoặc requests đang chờ ở đây*

};

}, [userId]); *// Mảng phụ thuộc: Effect chỉ chạy lại nếu userId thay đổi*

if (!user) {

return <div>Đang tải...</div>;

}

return (

<div>

<h1>{user.name}</h1>

<p>{user.email}</p>

</div>

);

}

**PHẦN 5: SO SÁNH CLASS COMPONENT VÀ FUNCTION COMPONENT**

| Tiêu chí | Class Component | Function Component với Hooks |
| --- | --- | --- |
| **Cú pháp** | Dài dòng hơn, cần class, extends, render() | Ngắn gọn, rõ ràng, chỉ là một hàm. |
| **State** | this.state và this.setState() | useState Hook |
| **Lifecycle Methods** | Các phương thức riêng biệt (componentDidMount, ...) | useEffect Hook thay thế cho tất cả. |
| this**keyword** | Cần sử dụng và phải bind đúng ngữ cảnh, dễ gây lỗi. | **Không cần**, tránh hoàn toàn sự phức tạp của this. |
| **Hiệu suất & Tối ưu** | Có thể hơi nặng hơn do lớp là đối tượng. | Nhẹ hơn một chút. |
| **Khả năng tái sử dụng** | Sử dụng HOCs (Higher-Order Components) - phức tạp. | Sử dụng Custom Hooks - **linh hoạt và dễ dàng hơn nhiều**. |
| **Cộng đồng & Xu hướng** | Vẫn được hỗ trợ nhưng **không còn là xu hướng chính**. | **Là cách viết chính thống và được khuyến khích** hiện nay. |

**KẾT LUẬN VÀ XU HƯỚNG**

JSX và Components là hai trụ cột không thể thiếu khi học và làm việc với React. JSX cung cấp một cú pháp mạnh mẽ và trực quan để mô tả UI, trong khi Components cho phép chúng ta chia nhỏ ứng dụng phức tạp thành các phần độc lập, dễ quản lý và tái sử dụng.

Sự phát triển từ Class Component sang Function Component với Hooks đánh dấu một bước tiến lớn trong cách tiếp cận của React. Function Component với Hooks không chỉ đơn giản hóa cú pháp mà còn làm code dễ đọc, dễ bảo trì và dễ kiểm thử hơn. Chúng giải quyết được nhiều vấn đề phức tạp của Class Component (như this, chia sẻ logic giữa các component).

**Xu hướng hiện nay và tương lai:** Cộng đồng React và chính đội ngũ phát triển React (React team) đều **khuyến khích sử dụng Function Component cùng với Hooks** cho các dự án mới. Các tính năng mới của React cũng sẽ tập trung vào mô hình Function Component và Hooks. Mặc dù Class Component vẫn được hỗ trợ đầy đủ và sẽ không bị gỡ bỏ, nhưng việc thành thạo Function Component và các Hooks cơ bản (useState, useEffect, useContext) là kỹ năng bắt buộc đối với bất kỳ nhà phát triển React hiện đại nào.