**Day01: Các Khái Niệm Cơ Bản Của ReactJS**

## 1. Giới Thiệu Về ReactJS

ReactJS, thường được gọi tắt là React, là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook (nay là Meta) vào năm 2013. React được thiết kế để xây dựng giao diện người dùng (UI) cho các ứng dụng web đơn trang (Single Page Applications - SPA) một cách hiệu quả và dễ dàng mở rộng. Thay vì là một framework đầy đủ như Angular hay Vue.js, React tập trung vào việc xử lý phần view layer, cho phép các lập trình viên xây dựng các thành phần UI có thể tái sử dụng.

### **Lịch Sử Phát Triển**

React được giới thiệu lần đầu tiên tại hội nghị JSConf US năm 2013 bởi Jordan Walke, một kỹ sư phần mềm tại Facebook. Ban đầu, nó được sử dụng nội bộ để giải quyết vấn đề hiệu suất trong việc cập nhật giao diện trên Facebook và Instagram. Đến năm 2015, React Native được ra mắt, mở rộng React sang phát triển ứng dụng di động. Hiện nay, React là một trong những thư viện phổ biến nhất, với hơn 10 triệu lượt tải hàng tuần trên npm (Node Package Manager).

### **Ưu Điểm Của React**

* **Hiệu Suất Cao**: Nhờ cơ chế Virtual DOM, React giảm thiểu việc thao tác trực tiếp với DOM thực, giúp ứng dụng nhanh hơn.
* **Tái Sử Dụng Code**: Các component cho phép chia nhỏ UI thành các phần độc lập, dễ quản lý và tái sử dụng.
* **Cộng Đồng Lớn**: Hỗ trợ từ Meta, cùng với hàng ngàn thư viện bên thứ ba như Redux, React Router.
* **Dễ Học**: Nếu bạn đã biết JavaScript, việc học React khá đơn giản, tập trung vào các khái niệm cốt lõi.
* **Tích Hợp Linh Hoạt**: Có thể kết hợp với các công nghệ khác như Node.js, Express cho backend.

### **Nhược Điểm**

* **Không Phải Framework Đầy Đủ**: Cần kết hợp với các thư viện khác để xử lý state management, routing, v.v.
* **Học Đường Cong Ban Đầu**: Các khái niệm như hooks (từ React 16.8) có thể phức tạp với người mới.
* **Cập Nhật Thường Xuyên**: Phiên bản mới có thể thay đổi cách làm việc, đòi hỏi cập nhật kiến thức.

React sử dụng mô hình declarative programming, nghĩa là bạn mô tả UI nên trông như thế nào dựa trên state, và React sẽ tự động cập nhật khi state thay đổi. Điều này khác biệt với imperative programming trong vanilla JavaScript, nơi bạn phải chỉ rõ từng bước thao tác DOM.

### **Cài Đặt React**

Để bắt đầu, bạn có thể sử dụng Create React App (CRA):

bash

npx create-react-app my-app

cd my-app

npm start

Điều này sẽ tạo một dự án cơ bản với Babel (chuyển đổi JSX) và Webpack (bundle code).

React đã phát triển qua nhiều phiên bản, với React 18 (2022) giới thiệu concurrent rendering để cải thiện hiệu suất.

## 2. Virtual DOM

Virtual DOM (Document Object Model ảo) là một trong những tính năng cốt lõi làm nên sức mạnh của React. Nó là một bản sao nhẹ của DOM thực tế, được lưu trữ trong bộ nhớ, giúp React tối ưu hóa việc cập nhật giao diện mà không cần thao tác trực tiếp với DOM browser – một quá trình thường chậm và tốn kém.

### **Khái Niệm Cơ Bản**

DOM thực là một cấu trúc cây đại diện cho HTML của trang web. Mỗi thay đổi (như thêm/xóa element) yêu cầu browser reflow và repaint, dẫn đến hiệu suất kém nếu cập nhật thường xuyên. Virtual DOM giải quyết bằng cách:

* Tạo một bản sao ảo của DOM.
* Khi state thay đổi, React tạo một Virtual DOM mới.
* So sánh (diffing) Virtual DOM mới với cũ để tìm sự khác biệt (reconciliation).
* Chỉ cập nhật những phần thay đổi lên DOM thực (batching updates).

Quá trình này được gọi là "Reconciliation" và được thực hiện bởi thuật toán React Fiber (từ React 16).

### **Cách Hoạt Động**

1. **Render Ban Đầu**: React xây dựng Virtual DOM từ JSX và render thành DOM thực.
2. **Cập Nhật**: Khi props/state thay đổi, phương thức render() được gọi lại, tạo Virtual DOM mới.
3. **Diffing**: React sử dụng heuristic algorithm để so sánh:
   * So sánh type của element (nếu khác, thay thế toàn bộ subtree).
   * So sánh attributes (props).
   * Duyệt đệ quy các child elements.
   * Sử dụng key prop để tối ưu hóa list rendering.
4. **Commit**: Áp dụng thay đổi tối thiểu lên DOM thực.

Ví dụ mã:

jsx

import React, { useState } from 'react';

function Counter() {

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

return (

<div>

<p>You clicked {count} times</p>

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

Click me

</button>

</div>

);

}

Khi click, state count thay đổi, Virtual DOM cập nhật chỉ phần text {count}, không render lại toàn bộ component.

### **Ưu Điểm**

* **Hiệu Suất**: Giảm thiểu DOM manipulations, đặc biệt hữu ích cho ứng dụng lớn.
* **Cross-Platform**: Áp dụng cho React Native (Virtual DOM cho native views).
* **Debug Dễ Dàng**: Công cụ như React DevTools hiển thị Virtual DOM tree.

### **Nhược Điểm**

* **Bộ Nhớ**: Virtual DOM chiếm bộ nhớ, nhưng thường không đáng kể.
* **Overhead**: Diffing có thể chậm nếu tree lớn mà không tối ưu (sử dụng shouldComponentUpdate hoặc React.memo).

So sánh với DOM thực: Trong vanilla JS, bạn dùng document.getElementById('id').innerHTML = newValue;, nhưng React làm điều này hiệu quả hơn.

## 3. JSX

JSX (JavaScript XML) là một cú pháp mở rộng của JavaScript, cho phép viết code giống HTML ngay trong JS. Nó không phải là HTML mà là syntactic sugar, được Babel chuyển đổi thành các lời gọi React.createElement().

### **Khái Niệm Cơ Bản**

JSX giúp viết UI declarative, kết hợp logic JS với markup. Ví dụ:

jsx

const element = <h1>Hello, world!</h1>*;*

Babel chuyển thành:

js

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

### **Quy Tắc JSX**

* **Tag Phải Đóng**: <img /> hoặc <div></div>.
* **Attribute**: Sử dụng camelCase (e.g., className thay vì class, onClick thay vì onclick).
* **JS Expression**: Sử dụng {} để nhúng JS: <p>{2 + 2}</p>.
* **Style**: Là object: style={{ color: 'red' }}.
* **Children**: Có thể là text, element, hoặc array.
* **Fragment**: Sử dụng <></> hoặc <React.Fragment> để nhóm elements mà không thêm node.

Ví dụ phức tạp:

jsx

function Greeting({ name }) {

return (

<div className="greeting">

<h1>Hello, {name}!</h1>

{name === 'Admin' ? <p>Admin panel</p> : <p>User dashboard</p>}

</div>

);

}

### Ưu Điểm

* **Đọc Dễ**: Code giống HTML, dễ hình dung UI.
* **Tích Hợp JS**: Dễ dàng sử dụng variables, functions, loops (sử dụng map() cho lists).
* **An Toàn**: JSX tự động escape để tránh XSS attacks.

### **Nhược Điểm**

* **Cần Transpiler**: Phải dùng Babel.
* **Học Ban Đầu**: Trông lạ nếu quen vanilla JS.

JSX là nền tảng cho việc viết components, làm React trở nên trực quan.

## 4. Components

Components là các khối xây dựng cơ bản của React, đại diện cho một phần UI độc lập và tái sử dụng. Chúng có thể là function hoặc class, nhận input (props) và trả về JSX.

### **Loại Components**

1. **Function Components**: Đơn giản, stateless (trước hooks). Từ React 16.8, có thể dùng hooks cho state.

jsx

function Welcome(props) {

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

}

1. **Class Components**: Sử dụng ES6 class, hỗ trợ state và lifecycle methods.

jsx

class Welcome extends React.Component {

render() {

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

}

}

1. **Higher-Order Components (HOC)**: Function nhận component và trả về component mới, dùng cho code reuse (e.g., withAuth).
2. **Render Props**: Pattern chia sẻ code qua prop là function render.

### **Composition**

Components có thể nest:

jsx

function App() {

return (

<div>

<Header />

<Content />

<Footer />

</div>

);

}

### **Best Practices**

* **Single Responsibility**: Mỗi component chỉ làm một việc.
* **Pure Functions**: Tránh side effects trong render.
* **Key Prop**: Cho lists để tối ưu diffing: <li key={id}>...</li>.
* **Controlled vs Uncontrolled**: Controlled components quản lý state qua React.

Components làm React modular, dễ test (sử dụng Jest/Enzyme).

## 5. Props Và State

Props và State là hai khái niệm quản lý dữ liệu trong React.

### **Props (Properties)**

* Input cho component, immutable (không thay đổi bởi component nhận).
* Truyền từ parent sang child: <Child name="John" />.
* Default props: Child.defaultProps = { name: 'Guest' };.
* PropTypes: Để validate (dùng PropTypes library hoặc TypeScript).

Ví dụ:

jsx

function Child(props) {

return <p>{props.name}</p>;

}

### **State**

* Dữ liệu nội bộ của component, mutable (có thể thay đổi).
* Khởi tạo trong constructor (class) hoặc useState (function).
* Cập nhật bằng setState() (class) hoặc setter từ hook.

Ví dụ class:

jsx

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

inc*rement = () => {*

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

};

render() {

return <button onClick={this.inc*rement}>{this.state.count}</button>;*

}

}

Ví dụ hook:

jsx

import { useState } from 'react'*;*

function Counter() {

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

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

}

### **So Sánh Props Và State**

| **Đặc Điểm** | **Props** | **State** |
| --- | --- | --- |
| Mutable? | Không | Có |
| Truyền Từ | Parent | Nội bộ component |
| Sử Dụng | Dữ liệu tĩnh, config | Dữ liệu động, user input |
| Cập Nhật | Parent re-render | setState/useState |

* **Lifting State Up**: Di chuyển state lên parent để chia sẻ giữa children.
* **useReducer**: Cho state phức tạp, tương tự Redux.

Quản lý props/state tốt giúp tránh bugs như stale state.

## 6. React Lifecycle

Lifecycle methods là các hàm đặc biệt trong class components được gọi tại các giai đoạn khác nhau của component's life. Với function components, sử dụng useEffect hook để simulate.

### **Giai Đoạn Lifecycle**

1. **Mounting** (Tạo):
   * constructor(): Khởi tạo state/props.
   * getDerivedStateFromProps(): Cập nhật state từ props (rare).
   * render(): Trả về JSX.
   * componentDidMount(): Sau khi render, dùng cho API calls, event listeners.
2. **Updating** (Cập Nhật):
   * getDerivedStateFromProps().
   * shouldComponentUpdate(): Quyết định có re-render không (tối ưu).
   * render().
   * getSnapshotBeforeUpdate(): Lấy snapshot trước update (rare).
   * componentDidUpdate(): Sau update, xử lý side effects.
3. **Unmounting** (Hủy):
   * componentWillUnmount(): Cleanup (remove listeners, timers).

### **Với Hooks**

useEffect thay thế nhiều methods:

jsx

useEffect(() => {

*// componentDidMount + componentDidUpdate*

return () => {

*// componentWillUnmount*

};

}, [dependencies]);

Ví dụ full lifecycle class:

jsx

class Timer extends React.Component {

constructor(props) {

super(props);

this.state = { seconds: 0 };

}

componentDidMount() {

this.interval = setInterval(() => this.setState({ seconds: this.state.seconds + 1 }), 1000);

}

componentWillUnmount() {

clearInterval(this.interval);

}

render() {

return <div>Seconds: {this.state.seconds}</div>;

}

}

Lifecycle giúp quản lý side effects như fetch data, animations.

## 7. Kết Luận

ReactJS với các khái niệm cốt lõi như Virtual DOM, JSX, Components, Props/State, và Lifecycle cung cấp nền tảng vững chắc để xây dựng ứng dụng web hiện đại. Bằng cách nắm vững chúng, bạn có thể tạo UI responsive, hiệu suất cao. Để đi sâu hơn, học hooks, context, và Redux. React tiếp tục phát triển, với focus trên server-side rendering (Next.js) và concurrent mode.

## 8. Tài Liệu Tham Khảo

* Official React Docs: reactjs.org
* MDN Web Docs for DOM
* Books: "Learning React" by Alex Banks & Eve Porcello
* Online: freeCodeCamp React Tutorial, Codecademy

# **Day02: Các thành phần cơ bản của ReactJs**

### **1. GIỚI THIỆU VỀ REACTJS**

ReactJS là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook, ra mắt năm 2013. Nó được thiết kế để xây dựng giao diện người dùng (UI) cho các ứng dụng web, đặc biệt là các ứng dụng single-page (SPA). ReactJS nổi bật với các tính năng:

* **Component-based**: Ứng dụng được xây dựng từ các component độc lập, có thể tái sử dụng
* **Virtual DOM**: Cơ chế tối ưu hóa hiệu năng bằng cách so sánh và cập nhật chỉ những phần thay đổi
* **Unidirectional Data Flow**: Luồng dữ liệu một chiều giúp ứng dụng dễ dàng debug và bảo trì
* **JSX**: Cú pháp mở rộng cho phép viết HTML trong JavaScript

### **2. JSX (JAVASCRIPT XML)**

#### **2.1. Khái niệm và đặc điểm**

JSX là một cú pháp mở rộng của JavaScript, cho phép nhúng mã HTML vào trong JavaScript. Nó không phải là chuẩn ECMAScript nhưng được transpile thành JavaScript thông thường bằng các công cụ như Babel.

**Đặc điểm chính:**

* Cho phép viết mã HTML-like trong JavaScript
* Cung cấp cú pháp trực quan, dễ đọc và bảo trì
* Giúp phát hiện lỗi sớm trong quá trình biên dịch

#### **2.2. Cú pháp cơ bản**

jsx

// Biểu thức JSX cơ bản

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

// JSX với nhiều phần tử (phải có một thẻ bao bọc)

const element = (

<div>

<h1>Hello</h1>

<p>Welcome to React</p>

</div>

);

// JSX với component

const Greeting = () => <h1>Hello, React!</h1>;

#### **2.3. Biểu thức JavaScript trong JSX**

JSX cho phép nhúng biểu thức JavaScript bằng cặp dấu ngoặc nhọn {}:

jsx

const name = "John Doe";

const element = <h1>Hello, {name}!</h1>;

// Sử dụng biểu thức

const price = 100;

const discount = 0.1;

const finalPrice = <p>Final price: ${price \* (1 - discount)}</p>;

// Gọi hàm JavaScript

const formatName = (user) => `${user.firstName} ${user.lastName}`;

const user = { firstName: "John", lastName: "Doe" };

const element = <h1>Hello, {formatName(user)}!</h1>;

#### **2.4. Thuộc tính và styling**

jsx

// Thuộc tính HTML

const element = <div className="container" id="main-content"></div>;

// Sự kiện

const handleClick = () => console.log("Clicked!");

const button = <button onClick={handleClick}>Click me</button>;

// Inline styling (sử dụng object JavaScript)

const styles = {

color: "blue",

backgroundColor: "lightgray",

padding: "10px"

};

const element = <div style={styles}>Styled element</div>;

// Styling trực tiếp

const element = <div style={{ color: "red", fontSize: "16px" }}>Text</div>;

#### **2.5. JSX và bảo mật**

JSX tự động escape các giá trị để ngăn chặn XSS (Cross-Site Scripting):

jsx

const userInput = "<script>alert('XSS')</script>";

const element = <div>{userInput}</div>; // An toàn, hiển thị như text thông

thường

### **3. COMPONENTS TRONG REACTJS**

#### **3.1. Khái niệm về Components**

Components là các khối xây dựng cơ bản của ứng dụng React. Mỗi component là một phần độc lập, có thể tái sử dụng, quản lý trạng thái và giao diện riêng.

**Đặc điểm:**

* Tách biệt concerns
* Có thể kết hợp (composable)
* Có thể tái sử dụng (reusable)
* Dễ dàng test và bảo trì

#### **3.2. Phân loại Components**

Có hai loại component chính:

1. **Class Components**: Sử dụng ES6 classes
2. **Function Components**: Sử dụng JavaScript functions

### **4. CLASS COMPONENTS**

#### **4.1. Định nghĩa và cú pháp**

Class components là các ES6 classes kế thừa từ React.Component:

jsx

import React, { Component } from 'react';

class Welcome extends Component {

render() {

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

}

}

// Sử dụng component

<Welcome name="John" />

#### **4.2. State và Lifecycle**

**State**: Là dữ liệu nội bộ của component, có thể thay đổi theo thời gian.

jsx

class Counter extends Component {

constructor(props) {

super(props);

this.state = {

count: 0

};

}

incrementCount = () => {

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

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.incrementCount}>Increment</button>

</div>

);

}

}

**Lifecycle Methods**: Các phương thức được gọi tại các thời điểm khác nhau trong vòng đời component.

jsx

class Example extends Component {

constructor(props) {

super(props);

console.log('Constructor');

}

componentDidMount() {

console.log('Component mounted');

// Gọi API, thiết lập subscriptions

}

componentDidUpdate(prevProps, prevState) {

console.log('Component updated');

// Phản ứng với props/state changes

}

componentWillUnmount() {

console.log('Component will unmount');

// Dọn dẹp subscriptions, timers

}

render() {

console.log('Render');

return <div>Example Component</div>;

}

}

#### **4.3. Phương thức render()**

Phương thức render() là bắt buộc, trả về JSX để mô tả giao diện:

jsx

class UserProfile extends Component {

render() {

const { user } = this.props;

return (

<div className="user-profile">

<img src={user.avatar} alt={user.name} />

<h2>{user.name}</h2>

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

{user.isOnline && <span className="online-badge">Online</span>}

</div>

);

}

}

#### **4.4. Xử lý sự kiện**

jsx

class EventExample extends Component {

handleClick = (event) => {

event.preventDefault();

console.log('Button clicked!');

}

handleInputChange = (event) => {

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

}

render() {

return (

<div>

<input

type="text"

onChange={this.handleInputChange}

value={this.state.value}

/>

<button onClick={this.handleClick}>Click me</button>

</div>

);

}

}

### **5. FUNCTION COMPONENTS**

#### **5.1. Định nghĩa và cú pháp**

Function components là các JavaScript functions trả về JSX:

jsx

// Function component cơ bản

function Welcome(props) {

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

}

// Arrow function component

const Welcome = (props) => {

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

};

// Implicit return

const Welcome = (props) => <h1>Hello, {props.name}</h1>;

#### **5.2. React Hooks**

Hooks là các hàm đặc biệt cho phép function components sử dụng state và lifecycle features.

#### **5.3. useState Hook**

Cho phép function components có state:

jsx

import React, { useState } from 'react';

function Counter() {

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

const [user, setUser] = useState({ name: '', age: 0 });

return (

<div>

<p>Count: {count}</p>

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

<input

value={user.name}

onChange={(e) => setUser({...user, name: e.target.value})}

/>

</div>

);

}

#### **5.4. useEffect Hook**

Thay thế lifecycle methods trong class components:

jsx

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

function UserProfile({ userId }) {

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

const [loading, setLoading] = useState(true);

// Tương đương componentDidMount và componentDidUpdate

useEffect(() => {

console.log('Effect running');

const fetchUser = async () => {

setLoading(true);

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

const userData = await response.json();

setUser(userData);

setLoading(false);

};

fetchUser();

}, [userId]); // Chỉ chạy lại khi userId thay đổi

// Tương đương componentWillUnmount

useEffect(() => {

const timer = setInterval(() => {

console.log('Timer tick');

}, 1000);

return () => {

clearInterval(timer); // Cleanup function

};

}, []); // Chỉ chạy một lần khi mount

if (loading) return <div>Loading...</div>;

return (

<div>

<h2>{user.name}</h2>

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

</div>

);

}

#### **5.5. Các Hooks khác**

**useContext**: Truy cập context mà không cần nesting

jsx

const ThemeContext = React.createContext('light');

function ThemedButton() {

const theme = useContext(ThemeContext);

return <button className={theme}>Themed Button</button>;

}

**useReducer**: Quản lý state phức tạp

jsx

const initialState = { count: 0 };

function reducer(state, action) {

switch (action.type) {

case 'increment':

return { count: state.count + 1 };

case 'decrement':

return { count: state.count - 1 };

default:

throw new Error();

}

}

function Counter() {

const [state, dispatch] = useReducer(reducer, initialState);

return (

<div>

Count: {state.count}

<button onClick={() => dispatch({ type: 'increment' })}>+</button>

<button onClick={() => dispatch({ type: 'decrement' })}>-</button>

</div>

);

}

**Custom Hooks**: Tạo hooks tùy chỉnh

jsx

function useLocalStorage(key, initialValue) {

const [storedValue, setStoredValue] = useState(() => {

try {

const item = window.localStorage.getItem(key);

return item ? JSON.parse(item) : initialValue;

} catch (error) {

return initialValue;

}

});

const setValue = (value) => {

try {

setStoredValue(value);

window.localStorage.setItem(key, JSON.stringify(value));

} catch (error) {

console.error(error);

}

};

return [storedValue, setValue];

}

// Sử dụng custom hook

function App() {

const [name, setName] = useLocalStorage('name', 'John Doe');

return (

<input

value={name}

onChange={(e) => setName(e.target.value)}

/>

);

}

### **6. SO SÁNH CLASS COMPONENTS VÀ FUNCTION COMPONENTS**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Class Components** | **Function Components** |
| **Cú pháp** | ES6 class | JavaScript function |
| **State** | this.state và this.setState() | useState Hook |
| **Lifecycle** | Lifecycle methods | useEffect Hook |
| **Performance** | Nặng hơn do class instance | Nhẹ hơn, không cần instance |
| **Code organization** | Logic phân tán theo lifecycle | Logic nhóm theo concern với Hooks |
| **Learning curve** | Cần hiểu this, bind, lifecycle | Đơn giản hơn, ít concept |
| **Reusability** | HOCs, Render Props | Custom Hooks |
| **Testing** | Phức tạp hơn | Dễ test hơn |
| **Future** | Legacy, vẫn được hỗ trợ | Xu hướng chính thức |

**Ưu điểm của Function Components với Hooks:**

* Code ngắn gọn và dễ đọc hơn
* Tránh được vấn đề với this
* Dễ dàng tái sử dụng logic với Custom Hooks
* Tổ chức code tốt hơn theo từng concern
* Hiệu năng tốt hơn

**Trường hợp sử dụng Class Components:**

* Legacy code base
* Error boundaries (hiện tại chỉ có trong class components)
* Components phức tạp với nhiều lifecycle logic

### **7. KẾT LUẬN**

ReactJS đã cách mạng hóa cách phát triển ứng dụng web front-end với mô hình component-based. JSX cung cấp cú pháp trực quan để xây dựng giao diện, trong khi components (cả class và function) cho phép tạo ra các khối xây dựng độc lập, có thể tái sử dụng.

**Xu hướng phát triển:**

* Function components với Hooks đang trở thành tiêu chuẩn mới
* Custom Hooks cho phép tái sử dụng logic hiệu quả
* React tiếp tục phát triển với các tính năng mới như Concurrent Mode, Server Componentsđ

**Khuyến nghị:**

* Ưu tiên sử dụng function components với Hooks cho dự án mới
* Học và nắm vững các Hooks cơ bản (useState, useEffect) và nâng cao
* Tận dụng Custom Hooks để tái sử dụng logic và giữ components gọn gàng

Việc hiểu rõ các thành phần cơ bản này là nền tảng quan trọng để xây dựng ứng dụng React hiệu quả, dễ bảo trì và mở rộng.

## Tài liệu tham khảo:

* React Official Documentation
* "Learning React" by Alex Banks and Eve Porcello
* "React Patterns" by Michael Chan
* React GitHub Repository và RFCs

# **Day04-05: EVENT – FORM – LIST&KEY**

## 1. GIỚI THIỆU

Trong lập trình giao diện người dùng hiện đại, **ReactJS** là một thư viện JavaScript phổ biến giúp phát triển ứng dụng web động. Ba khái niệm quan trọng cần nắm vững để xây dựng ứng dụng React là: **Event, Form, List & Key**.

* **Event** cho phép ứng dụng phản hồi tương tác của người dùng.
* **Form** cung cấp giao diện nhập liệu.
* **Render có điều kiện** cho phép hiển thị linh hoạt dựa trên trạng thái.
* **List & Key** giúp quản lý và hiển thị danh sách dữ liệu hiệu quả.

Đây là những thành phần nền tảng, xuất hiện trong hầu hết các ứng dụng React thực tế, từ form đăng nhập đơn giản cho đến hệ thống quản lý dữ liệu phức tạp.

## 2. CƠ SỞ LÝ THUYẾT

### 2.1. Event trong ReactJS

* Trong HTML truyền thống, event thường được viết trực tiếp trong thẻ.
* Trong JSX (React), event được viết dưới dạng **camelCase** và gán cho một function.
* Ví dụ:
  + onClick={handleClick} → hàm handleClick sẽ được gọi khi click.
  + Phân biệt onClick={fncName} và onClick={fncName()}.

Các loại event phổ biến:

* **Không có tham số** → gọi trực tiếp function.
* **Có tham số** → dùng arrow function hoặc bind.
* **Sử dụng props/state** để thay đổi giao diện động.
* **Event trên thẻ <a>** → sử dụng e.preventDefault() để ngăn hành vi mặc định.

### 2.2. Form trong ReactJS

Form trong React chia làm 2 loại:

* **Controlled Component**
  + Dữ liệu trong form được quản lý bởi **state**.
  + Mỗi lần thay đổi input sẽ kích hoạt setState.
  + Dữ liệu luôn đồng bộ giữa giao diện và state.
* **Uncontrolled Component**
  + Dữ liệu được quản lý trực tiếp bởi DOM.
  + Sử dụng **ref** để truy cập giá trị input.
  + Thường dùng cho các input đặc biệt (file, media).

### 2.3. Render có điều kiện

* Cho phép hiển thị component tùy theo state.
* Có thể sử dụng:
  + Câu lệnh if thông thường.
  + Toán tử ba ngôi ? :.
  + Biểu thức logic &&.
* Giúp giao diện linh hoạt, tránh hiển thị dư thừa.

### 2.4. List và Key

* Khi render danh sách từ mảng, React yêu cầu **key** cho mỗi phần tử.
* **Key** giúp React xác định phần tử nào thay đổi, thêm hoặc xóa.
* Key cần **duy nhất** trong danh sách.
* Có thể dùng **index** làm key, nhưng không khuyến khích khi dữ liệu thay đổi thường xuyên.

## 3. THIẾT KẾ VÀ MINH HỌA

### 3.1. Ví dụ về Event

function Welcome() {

function handleClick() {

alert("Xin chào!");

}

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

}

### 3.2. Ví dụ về Form

* **Controlled Component**

function LoginForm() {

const [username, setUsername] = useState("");

const handleChange = (e) => setUsername(e.target.value);

return (

<form>

<input type="text" value={username} onChange={handleChange} />

<p>Xin chào, {username}</p>

</form>

);

}

* **Uncontrolled Component**

function FileUpload() {

const fileInput = useRef(null);

const handleSubmit = (e) => {

e.preventDefault();

alert(`File: ${fileInput.current.files[0].name}`);

};

return (

<form onSubmit={handleSubmit}>

<input type="file" ref={fileInput} />

<button type="submit">Upload</button>

</form>

);

}

### 3.3. Ví dụ về Render có điều kiện

function UserGreeting({ isLoggedIn }) {

return (

<div>

{isLoggedIn ? <h1>Chào mừng bạn quay lại!</h1> : <h1>Vui lòng đăng nhập</h1>}

</div>

);

}

### 3.4. Ví dụ về List & Key

function NumberList({ numbers }) {

return (

<ul>

{numbers.map((num) => (

<li key={num}>{num}</li>

))}

</ul>

);

}

## 4. ỨNG DỤNG THỰC TẾ

* **Event**: nút bấm like/share, điều khiển game, điều hướng menu.
* **Form**: đăng nhập, đăng ký, giỏ hàng thương mại điện tử.
* **Render có điều kiện**: hiển thị thông báo khi chưa đăng nhập, render dữ liệu động.
* **List & Key**: hiển thị danh sách sản phẩm, bảng dữ liệu, tin nhắn trong chat app.

## 5. ĐÁNH GIÁ ƯU – NHƯỢC ĐIỂM

* **Event**
  + Ưu điểm: dễ quản lý, gọn gàng.
  + Nhược điểm: khi nhiều event lồng nhau dễ gây khó bảo trì.
* **Form**
  + Ưu điểm: Controlled component cho phép quản lý chặt chẽ dữ liệu.
  + Nhược điểm: Uncontrolled component khó kiểm soát logic.
* **Render có điều kiện**
  + Ưu điểm: giao diện linh hoạt, thân thiện người dùng.
  + Nhược điểm: nếu quá nhiều điều kiện có thể gây rối.
* **List & Key**
  + Ưu điểm: tối ưu hiệu suất, dễ quản lý.
  + Nhược điểm: nếu key không duy nhất → bug khó phát hiện.

## 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

**Event – Form – Render có điều kiện – List & Key** là nền tảng không thể thiếu khi lập trình với ReactJS. Việc nắm vững giúp:

* Xây dựng ứng dụng dễ mở rộng, dễ bảo trì.
* Tăng hiệu suất xử lý dữ liệu và trải nghiệm người dùng.

**Hướng phát triển**:

* Kết hợp với **Context API**, **Redux** để quản lý state toàn cục.
* Tối ưu hiệu suất render bằng kỹ thuật **memoization**.
* Ứng dụng trong các dự án thực tế như thương mại điện tử, mạng xã hội, hệ thống quản trị.

## 7. TÀI LIỆU THAM KHẢO

1. ReactJS Official Documentation – https://react.dev
2. W3Schools React Tutorial – https://www.w3schools.com/react
3. Slide giảng dạy: Session 03 - Event-Form-ListKey
4. GeeksforGeeks – ReactJS Guide

**Day07-08: React Hook**

## 1. React Hook là gì?

**Hook** là các hàm đặc biệt trong React cho phép bạn “móc nối” (hook into) vào các tính năng của React như state, lifecycle, context, v.v. mà không cần viết Class Component.

Đặc điểm chính:

* Chỉ dùng trong **Function Component**.
* Cho phép quản lý state, side effects, context, ref, memoization, v.v.
* Làm code gọn hơn, dễ tái sử dụng logic.

Ví dụ so sánh:

**Trước (Class Component):**

class Counter extends React.Component {

state = { count: 0 };

render() {

return (

<div>

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

<button onClick={() => this.setState({ count: this.state.count + 1 })}>

Nhấn tôi

</button>

</div>

);

}

}

**Sau (Function Component + Hook):**

import { useState } from "react";

function Counter() {

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

return (

<div>

<p>Bạn đã nhấn {count} lần</p>

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

</div>

);

}

## 2. Các loại Hook cơ bản trong React

### 3.1. useState

* Dùng để khai báo và quản lý **state** trong Function Component.
* Cú pháp:

const [state, setState] = useState(initialValue);

* Trong đó state là giá trị hiện tại, setState là hàm để cập nhật giá trị.

### 3.2. useEffect

* Dùng để quản lý **side effect**: gọi API, thao tác DOM, timer, subscription...
* Cú pháp:

useEffect(() => {

// code thực thi

return () => {

// cleanup (nếu cần)

};

}, [dependencies]);

* dependencies: mảng phụ thuộc. Nếu rỗng ([]) thì chỉ chạy 1 lần sau render.

### 3.3. useContext

* Cho phép sử dụng **Context API** để truyền dữ liệu qua nhiều component mà không cần props.
* Giúp quản lý theme, ngôn ngữ, thông tin người dùng.

### 3.4. useRef

* Tạo ra một object chứa giá trị có thể thay đổi mà không làm component render lại.
* Thường dùng để truy cập trực tiếp DOM hoặc lưu trữ giá trị tạm.

### 3.5. useMemo

* Ghi nhớ (memoize) giá trị tính toán phức tạp để tránh tính toán lại không cần thiết.
* Tăng hiệu năng khi có các phép tính nặng hoặc danh sách lớn.

### 3.6. useCallback

* Ghi nhớ (memoize) một hàm callback để tránh tạo lại ở mỗi lần render.
* Hữu ích khi truyền callback xuống component con để ngăn re-render không cần thiết.

### 3.7. useReducer

* Một dạng quản lý state phức tạp, thay thế useState khi logic cập nhật state rắc rối.
* Gần giống Redux thu nhỏ bên trong component.

## 3. Nguyên tắc sử dụng Hook

* Chỉ gọi Hook ở **cấp cao nhất** của Function Component, không được gọi trong vòng lặp, điều kiện hay nested function.
* Chỉ gọi Hook trong **Function Component** hoặc **Custom Hook**, không dùng trong Class.
* Phải tuân theo quy tắc tên: **Custom Hook** luôn bắt đầu bằng use.

## 4. Tạo Custom Hook

Ngoài Hook có sẵn, ta có thể tự viết **Custom Hook** để tái sử dụng logic.

Ví dụ: Custom Hook đếm thời gian online:

import { useState, useEffect } from "react";

function useTimer() {

const [seconds, setSeconds] = useState(0);

useEffect(() => {

const timer = setInterval(() => setSeconds(s => s + 1), 1000);

return () => clearInterval(timer);

}, []);

return seconds;

}

// Sử dụng:

function OnlineTime() {

const seconds = useTimer();

return <p>Bạn đã online {seconds} giây</p>;

}

## 5. Ưu điểm của Hook

* Code ngắn gọn, dễ đọc.
* Tái sử dụng logic giữa nhiều component qua Custom Hook.
* Không cần dùng Class, giảm phức tạp (constructor, this...).
* Quản lý state và side effect hiệu quả hơn.

## 6. Nhược điểm của Hook

* Quy tắc sử dụng nghiêm ngặt (dễ mắc lỗi nếu không quen).
* Khó debug hơn trong một số trường hợp (closure, dependency array).
* Người mới học có thể thấy trừu tượng.

## 7. So sánh Hook và Class Component

| **Tiêu chí** | **Class Component** | **Function Component + Hook** |
| --- | --- | --- |
| Quản lý state | Có | Có (useState, useReducer) |
| Lifecycle | Có (componentDidMount...) | Có (useEffect) |
| Độ dài code | Dài hơn | Ngắn gọn hơn |
| Tái sử dụng logic | Khó | Dễ (Custom Hook) |
| Hiệu năng | Trung bình | Tối ưu hơn (memo, useCallback, useMemo) |

## 8. Ứng dụng thực tế

* **useState, useEffect**: Form, Counter, API call.
* **useContext**: Quản lý theme (dark/light), đa ngôn ngữ.
* **useRef**: Truy cập input, video, canvas.
* **useReducer**: Quản lý giỏ hàng trong website thương mại điện tử.
* **Custom Hook**: Đo kích thước màn hình, quản lý authentication.

## 9. Kết luận

React Hook đã thay đổi cách viết code trong React, giúp Function Component trở nên mạnh mẽ, thay thế dần Class Component. Với Hook, developer có thể viết code ngắn gọn, dễ đọc, tái sử dụng logic, và quản lý state hiệu quả hơn.

Tuy còn một số hạn chế, nhưng Hook đã trở thành **chuẩn mực trong phát triển ứng dụng React hiện đại**. Nắm vững Hook là kỹ năng bắt buộc cho bất kỳ lập trình viên Frontend nào.