# Các thành phần cơ bản của ReactJs

Mục Lục:

[**Các thành phần cơ bản của ReactJs** 1](#_Toc208478397)

[1. GIỚI THIỆU VỀ REACTJS 1](#_Toc208478398)

[2. JSX (JAVASCRIPT XML) 2](#_Toc208478399)

[3. COMPONENTS TRONG REACTJS 4](#_Toc208478400)

[4. CLASS COMPONENTS 4](#_Toc208478401)

[5. FUNCTION COMPONENTS 8](#_Toc208478402)

[6. SO SÁNH CLASS COMPONENTS VÀ FUNCTION COMPONENTS 12](#_Toc208478403)

[7. KẾT LUẬN 13](#_Toc208478404)

[Tài liệu tham khảo: 14](#_Toc208478405)

[**EVENT – FORM – LIST&KEY** 14](#_Toc208478406)

[1. GIỚI THIỆU 14](#_Toc208478407)

[2. CƠ SỞ LÝ THUYẾT 14](#_Toc208478408)

[2.1. Event trong ReactJS 14](#_Toc208478409)

[2.2. Form trong ReactJS 15](#_Toc208478410)

[2.3. Render có điều kiện 15](#_Toc208478411)

[2.4. List và Key 15](#_Toc208478412)

[3. THIẾT KẾ VÀ MINH HỌA 15](#_Toc208478413)

[3.1. Ví dụ về Event 15](#_Toc208478414)

[3.2. Ví dụ về Form 16](#_Toc208478415)

[3.3. Ví dụ về Render có điều kiện 17](#_Toc208478416)

[3.4. Ví dụ về List & Key 17](#_Toc208478417)

[4. ỨNG DỤNG THỰC TẾ 17](#_Toc208478418)

[5. ĐÁNH GIÁ ƯU – NHƯỢC ĐIỂM 17](#_Toc208478419)

[6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 18](#_Toc208478420)

[7. TÀI LIỆU THAM KHẢO 18](#_Toc208478421)

### **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

# 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