BÁO CÁO REACTJS

MỤC LỤC

1. Giới thiệu về ReactJS
2. JSX (JavaScript XML)
3. Components
4. Props và State
5. Virtual DOM
6. React Lifecycle
7. Form và Xử lý Sự kiện trong ReactJS
8. Kết luận và Xu hướng phát triển

**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 tổng hợp các khái niệm nền tảng và cốt lõi nhất của React.

**1. GIỚI THIỆU VỀ REACTJS**

**1.1. Lịch sử và xuất xứ**

ReactJS là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook (nay là Meta) vào năm 2011. Ban đầu, React được tạo ra để giải quyết các vấn đề về hiệu suất và bảo trì trong các ứng dụng web phức tạp của Facebook. Nó chính thức được công bố mã nguồn mở tại hội nghị JSConf US vào tháng 5/2013.

**1.2. Định nghĩa**

ReactJS là một thư viện JavaScript dùng để xây dựng giao diện người dùng (UI), đặc biệt tập trung vào việc xây dựng các ứng dụng single-page (SPA). React sử dụng component-based architecture, cho phép phát triển các thành phần giao diện độc lập, tái sử dụng và dễ bảo trì.

**1.3. Ưu điểm**

* **Hiệu suất cao**: Nhờ cơ chế Virtual DOM
* **Tái sử dụng component**: Giảm thiểu code trùng lặp
* **Dễ học và sử dụng**: Cú pháp đơn giản, tài liệu phong phú
* **Cộng đồng lớn**: Nhiều thư viện và công cụ hỗ trợ
* **Hỗ trợ mobile**: React Native cho phát triển ứng dụng di động

**1.4. Kiến trúc tổng quan**

React hoạt động dựa trên mô hình declarative programming, nơi developer mô tả UI nên trông như thế nào ở các trạng thái khác nhau, và React sẽ tự động cập nhật DOM khi dữ liệu thay đổi.

**2. JSX (JAVASCRIPT XML)**

**2.1. Định nghĩa**

JSX là một cú pháp mở rộng của JavaScript, cho phép viết mã HTML-like trong JavaScript. Nó không bắt buộc trong React nhưng được khuyến khích sử dụng vì tính rõ ràng và dễ hiểu.

**2.2. 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.
* **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.

**2.3. Cú pháp cơ bản**

* **Nhúng biểu thức JavaScript vào JSX:** Sử dụng cặp dấu ngoặc nhọn {}.

jsx

const name = 'Tuấn Anh';

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

* **Chỉ định thuộc tính với JSX:** Sử dụng className thay cho class, htmlFor thay cho for.

jsx

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

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

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

* **JSX là một biểu thức:** 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>;

}

* **JSX Children:** Một thẻ JSX có thể chứa các thẻ con (children).

jsx

const element = (

<div className="my-container">

<h1>Tiêu đề</h1>

<p>Đây là một đoạn văn bản.</p>

</div>

);

**2.4. 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 />.

**2.5. Biên dịch JSX**

JSX được biên dịch thành các lệnh React.createElement():

jsx

*// JSX*

const element = <h1 className="greeting">Hello, world!</h1>;

*// Sau khi biên dịch*

const element = React.createElement(

'h1',

{className: 'greeting'},

'Hello, world!'

);

**3. COMPONENTS**

**3.1. 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.

**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.

**3.2. Phân loại Component**

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

**3.3. Function Components (Component hàm)**

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;

**3.4. Class Components (Component lớp)**

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;

**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 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 tại các thời điểm cụ thể.

**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>

<button onClick={this.increment}>

Bấm vào tôi

</button>

</div>

);

}

}

export default Counter;

**3.5. Component Composition**

Các component có thể lồng vào nhau:

jsx

function App() {

return (

<div>

<Welcome name="Alice" />

<Welcome name="Bob" />

<Welcome name="Charlie" />

</div>

);

}

**3.6. Ưu điểm của component-based architecture**

* **Tái sử dụng**: Một component có thể dùng nhiều nơi
* **Bảo trì**: Dễ dàng sửa đổi và kiểm thử từng phần
* **Phân chia công việc**: Nhiều developer có thể làm việc song song
* **Tổ chức code**: Code được cấu trúc rõ ràng, dễ hiểu

**4. PROPS VÀ STATE**

**4.1. Props (Properties)**

**4.1.1. Định nghĩa**

Props là dữ liệu được truyền từ component cha xuống component con. Props là read-only và không thể thay đổi bởi component nhận chúng.

**4.1.2. Truyền và nhận props**

jsx

*// Truyền props*

<UserProfile name="John" age={25} isActive={true} />

*// Nhận props trong function component*

function UserProfile(props) {

return (

<div>

<p>Name: {props.name}</p>

<p>Age: {props.age}</p>

<p>Active: {props.isActive ? 'Yes' : 'No'}</p>

</div>

);

}

*// Nhận props trong class component*

class UserProfile extends React.Component {

render() {

return (

<div>

<p>Name: {this.props.name}</p>

<p>Age: {this.props.age}</p>

</div>

);

}

}

**4.1.3. Default Props**

jsx

function Greeting(props) {

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

}

Greeting.defaultProps = {

name: 'Guest'

};

**4.1.4. PropTypes (kiểm tra kiểu dữ liệu)**

jsx

import PropTypes from 'prop-types';

function UserProfile(props) {

*// ...*

}

UserProfile.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number,

isActive: PropTypes.bool

};

**4.2. State**

**4.2.1. Định nghĩa**

State là dữ liệu nội bộ của component, có thể thay đổi theo thời gian. Khi state thay đổi, component sẽ re-render.

**4.2.2. State trong function component (với Hook)**

jsx

import { 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>

);

}

**4.2.3. State trong class component**

jsx

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = {

count: 0

};

}

render() {

return (

<div>

<p>You clicked {this.state.count} times</p>

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

Click me

</button>

</div>

);

}

}

**4.2.4. So sánh Props và State**

| **Tiêu chí** | **Props** | **State** |
| --- | --- | --- |
| **Truyền dữ liệu** | Từ cha → con | Nội bộ component |
| **Thay đổi** | Read-only | Có thể thay đổi |
| **Ảnh hưởng** | Component con re-render | Component hiện tại re-render |
| **Khởi tạo** | Từ component cha | Trong component |

**4.3. Quy tắc khi làm việc với State**

* **Không thay đổi state trực tiếp**: Luôn dùng setState() hoặc hàm setter từ useState()
* **State updates có thể bất đồng bộ**: React có thể gộp nhiều setState() lại
* **State updates được merge**: Trong class component, setState() merge object

**5. VIRTUAL DOM**

**5.1. Khái niệm**

Virtual DOM (DOM ảo) là một khái niệm then chốt trong React. Đây là một bản sao JavaScript của DOM thật, được lưu trữ trong bộ nhớ. Khi state của ứng dụng thay đổi, React tạo một Virtual DOM mới và so sánh với phiên bản trước đó để xác định những thay đổi cần thiết.

**5.2. Cơ chế hoạt động**

Quy trình làm việc của Virtual DOM:

1. **Khi state thay đổi**: React tạo một Virtual DOM mới
2. **So sánh (Diffing)**: So sánh Virtual DOM mới với phiên bản cũ
3. **Xác định thay đổi**: Tìm ra chính xác những phần tử cần cập nhật
4. **Cập nhật DOM thật**: Chỉ cập nhật những phần thực sự thay đổi

**5.3. Ưu điểm**

* **Hiệu suất**: Giảm thiểu thao tác trực tiếp với DOM thật
* **Tối ưu hóa**: Tự động nhóm các thay đổi để cập nhật một lần
* **Trừu tượng hóa**: Developer không cần quan tâm đến chi tiết cập nhật DOM

**5.4. Ví dụ minh họa**

javascript

*// Trước khi thay đổi*

const oldVirtualDOM = {

type: 'div',

props: { className: 'container', children: 'Hello World' }

};

*// Sau khi thay đổi*

const newVirtualDOM = {

type: 'div',

props: { className: 'container', children: 'Hello React' }

};

*// React sẽ chỉ cập nhật text content từ 'Hello World' thành 'Hello React'*

**6. REACT LIFECYCLE**

**6.1. Khái niệm**

Vòng đời component là các giai đoạn mà một component trải qua từ khi được tạo, cập nhật đến khi bị hủy. Hiểu vòng đời giúp kiểm soát hành vi component ở các thời điểm khác nhau.

**6.2. Vòng đời trong class component**

**6.2.1. Mounting (Gắn vào DOM)**

1. **constructor()**: Khởi tạo state và bind methods
2. **static getDerivedStateFromProps()**: Cập nhật state dựa trên props
3. **render()**: Trả về JSX để render
4. **componentDidMount()**: Được gọi sau khi component được render lần đầu

**6.2.2. Updating (Cập nhật)**

1. **static getDerivedStateFromProps()**: Cập nhật state khi props thay đổi
2. **shouldComponentUpdate()**: Quyết định component có nên re-render không
3. **render()**: Render lại component
4. **getSnapshotBeforeUpdate()**: Lấy thông tin từ DOM trước khi cập nhật
5. **componentDidUpdate()**: Được gọi sau khi update hoàn tất

**6.2.3. Unmounting (Gỡ khỏi DOM)**

1. **componentWillUnmount()**: Dọn dẹp trước khi component bị hủy

**6.2.4. Error Handling (Xử lý lỗi)**

1. **static getDerivedStateFromError()**: Cập nhật state khi có lỗi
2. **componentDidCatch()**: Xử lý lỗi

**6.3. Vòng đời trong function component (với Hooks)**

**6.3.1. useEffect Hook**

jsx

import { useEffect, useState } from 'react';

function Example() {

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

*// Tương đương componentDidMount và componentDidUpdate*

useEffect(() => {

document.title = `You clicked ${count} times`;

});

*// Tương đương componentDidMount*

useEffect(() => {

console.log('Component mounted');

}, []);

*// Tương đương componentDidUpdate với điều kiện*

useEffect(() => {

console.log('Count changed:', count);

}, [count]); *// Chỉ chạy khi count thay đổi*

*// Tương đương componentWillUnmount*

useEffect(() => {

return () => {

console.log('Component will unmount');

};

}, []);

return (

<div>

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

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

Click me

</button>

</div>

);

}

**6.4. So sánh lifecycle methods và hooks**

| **Class Component** | **Function Component với Hooks** |
| --- | --- |
| componentDidMount | useEffect với dependency array rỗng |
| componentDidUpdate | useEffect với dependencies cụ thể |
| componentWillUnmount | Return cleanup function từ useEffect |
| shouldComponentUpdate | React.memo hoặc useMemo |

**7. FORM VÀ XỬ LÝ SỰ KIỆN TRONG REACTJS**

**7.1. Giới thiệu**

Form là thành phần không thể thiếu trong các ứng dụng web hiện đại, cho phép người dùng nhập và gửi dữ liệu lên server. Báo cáo này trình bày chi tiết quá trình thiết kế, xây dựng và triển khai một component EventForm trong ứng dụng ReactJS.

**Mục tiêu:**

* Tạo một form nhập liệu để thu thập thông tin về một sự kiện (event) với các trường thông tin cơ bản.
* Áp dụng các khái niệm cốt lõi của ReactJS như state, props, và event handling.
* Triển khai cơ chế xác thực dữ liệu (validation) để đảm bảo tính toàn vẹn của dữ liệu đầu vào.
* Xử lý sự kiện gửi form một cách hiệu quả.

**Phạm vi:**  
Báo cáo tập trung vào việc xây dựng component form phía client-side sử dụng React Hook (useState). Các vấn đề như kết nối đến API backend, routing, hoặc state management phức tạp (Redux, Context API) sẽ không được đề cập chi tiết.

**7.2. Tổng quan về Form trong ReactJS**

Form trong ReactJS có thể được quản lý theo hai cách chính: Controlled Components và Uncontrolled Components.

**7.2.1. Controlled Components**

Là kỹ thuật phổ biến nhất. Trong kỹ thuật này, giá trị của các phần tử form (input, textarea, select) được điều khiển bởi state của React. Mỗi lần người dùng nhập liệu, một sự kiện (onChange) được kích hoạt để cập nhật state, và state này sau đó lại được dùng để render lại giá trị cho phần tử form.

* **Ưu điểm:** Dữ liệu form được quản lý tập trung, dễ dàng để validate hoặc xử lý trước khi gửi đi.
* **Nhược điểm:** Có thể gây ra hiệu suất kém nếu form quá phức tạp và state update quá thường xuyên (tuy nhiên, trong hầu hết trường hợp, điều này là không đáng kể).

**7.2.2. Uncontrolled Components**

Dữ liệu form được xử lý trực tiếp bởi chính DOM. React không quản lý state của input. Để lấy giá trị, ta sử dụng useRef để truy cập trực tiếp đến phần tử DOM.

* **Ưu điểm:** Hiệu suất tốt hơn một chút vì tránh được nhiều lần re-render.
* **Nhược điểm:** Khó khăn hơn trong việc validate theo thời gian thực và quản lý dữ liệu tập trung.

**7.2.3. Xử lý sự kiện (Event Handling)**

Các sự kiện trong React được đặt tên theo camelCase (ví dụ: onChange, onSubmit) thay vì chữ thường trong HTML (onchange, onsubmit). Hàm xử lý sự kiện được truyền dưới dạng JSX expression {handleSubmit} thay vì một string "handleSubmit()".

**7.3. Thiết kế và Cài đặt Event Form**

**7.3.1. Phân tích Yêu cầu**

Giả sử form sự kiện cần thu thập các thông tin sau:

* Tên sự kiện (eventName) - Bắt buộc.
* Ngày diễn ra (eventDate) - Bắt buộc, phải là ngày ở tương lai.
* Mô tả (eventDescription) - Không bắt buộc.
* Thể loại (eventCategory) - Bắt buộc, chọn từ danh sách.

**7.3.2. Thiết kế State**

Sử dụng React Hook useState để tạo một state object formData chứa tất cả giá trị của form, và một state object errors để lưu trữ thông báo lỗi cho từng trường.

jsx

const [formData, setFormData] = useState({

eventName: '',

eventDate: '',

eventDescription: '',

eventCategory: '',

});

const [errors, setErrors] = useState({});

**7.3.3. Cấu trúc Component**

Component EventForm sẽ là một functional component. Nó sẽ return một JSX structure chứa thẻ <form> và các trường input bên trong.

**7.3.4. Code Implementation**

jsx

*// EventForm.jsx*

import React, { useState } from 'react';

const EventForm = () => {

*// Khai báo state*

const [formData, setFormData] = useState({

eventName: '',

eventDate: '',

eventDescription: '',

eventCategory: '',

});

const [errors, setErrors] = useState({});

*// Hàm xử lý thay đổi chung cho tất cả các input*

const handleInputChange = (e) => {

const { name, value } = e.target;

setFormData({

...formData,

[name]: value,

});

*// Xóa lỗi của trường vừa được nhập (nếu có)*

if (errors[name]) {

setErrors({

...errors,

[name]: '',

});

}

};

*// Hàm xử lý validate form*

const validateForm = () => {

let isValid = true;

let newErrors = {};

*// Validate eventName*

if (!formData.eventName.trim()) {

newErrors.eventName = 'Tên sự kiện là bắt buộc.';

isValid = false;

}

*// Validate eventDate*

if (!formData.eventDate) {

newErrors.eventDate = 'Ngày diễn ra là bắt buộc.';

isValid = false;

} else if (new Date(formData.eventDate) < new Date()) {

newErrors.eventDate = 'Ngày diễn ra phải ở trong tương lai.';

isValid = false;

}

*// Validate eventCategory*

if (!formData.eventCategory) {

newErrors.eventCategory = 'Vui lòng chọn một thể loại.';

isValid = false;

}

setErrors(newErrors);

return isValid;

};

*// Hàm xử lý gửi form*

const handleSubmit = (e) => {

e.preventDefault(); *// Ngăn chặn hành vi gửi form mặc định*

if (validateForm()) {

*// Form hợp lệ, xử lý dữ liệu ở đây (ví dụ: gọi API)*

console.log('Form data submitted:', formData);

alert('Đăng ký sự kiện thành công!');

*// Reset form (tùy chọn)*

*// setFormData({ eventName: '', eventDate: '', ... });*

} else {

console.log('Form có lỗi', errors);

}

};

return (

<form onSubmit={handleSubmit} style={{ maxWidth: '500px', margin: '20px auto' }}>

<h2>Đăng ký Sự kiện Mới</h2>

{*/\* Trường Tên sự kiện \*/*}

<div style={{ marginBottom: '15px' }}>

<label htmlFor="eventName">Tên sự kiện:</label>

<input

type="text"

id="eventName"

name="eventName"

value={formData.eventName}

onChange={handleInputChange}

style={{ width: '100%', padding: '8px', fontSize: '12px', fontFamily: 'Times New Roman' }}

/>

{errors.eventName && <span style={{ color: 'red', fontSize: '10px' }}>{errors.eventName}</span>}

</div>

{*/\* Trường Ngày diễn ra \*/*}

<div style={{ marginBottom: '15px' }}>

<label htmlFor="eventDate">Ngày diễn ra:</label>

<input

type="date"

id="eventDate"

name="eventDate"

value={formData.eventDate}

onChange={handleInputChange}

style={{ width: '100%', padding: '8px', fontSize: '12px', fontFamily: 'Times New Roman' }}

/>

{errors.eventDate && <span style={{ color: 'red', fontSize: '10px' }}>{errors.eventDate}</span>}

</div>

{*/\* Trường Mô tả \*/*}

<div style={{ marginBottom: '15px' }}>

<label htmlFor="eventDescription">Mô tả:</label>

<textarea

id="eventDescription"

name="eventDescription"

value={formData.eventDescription}

onChange={handleInputChange}

style={{ width: '100%', padding: '8px', fontSize: '12px', fontFamily: 'Times New Roman', height: '80px' }}

/>

</div>

{*/\* Trường Thể loại \*/*}

<div style={{ marginBottom: '15px' }}>

<label htmlFor="eventCategory">Thể loại:</label>

<select

id="eventCategory"

name="eventCategory"

value={formData.eventCategory}

onChange={handleInputChange}

style={{ width: '100%', padding: '8px', fontSize: '12px', fontFamily: 'Times New Roman' }}

>

<option value="">-- Chọn thể loại --</option>

<option value="conference">Hội nghị</option>

<option value="webinar">Hội thảo trực tuyến</option>

<option value="workshop">Workshop</option>

<option value="seminar">Seminar</option>

</select>

{errors.eventCategory && <span style={{ color: 'red', fontSize: '10px' }}>{errors.eventCategory}</span>}

</div>

{*/\* Nút Gửi \*/*}

<button type="submit" style={{ padding: '10px 20px', fontSize: '12px', fontFamily: 'Times New Roman' }}>

Đăng ký Sự kiện

</button>

</form>

);

};

export default EventForm;

**7.4. Xác thực Dữ liệu (Validation)**

**7.4.1. Validation cơ bản**

Hàm validateForm được triển khai để kiểm tra tính hợp lệ của dữ liệu:

* **Kiểm tra rỗng:** Sử dụng !formData.fieldName.trim() cho string và !formData.fieldName cho các trường khác.
* **Kiểm tra logic nghiệp vụ:** So sánh ngày nhập vào với ngày hiện tại để đảm bảo ngày diễn ra là trong tương lai.

**7.4.2. Hiển thị thông báo lỗi**

* State errors lưu trữ thông báo lỗi cho từng trường.
* Khi validate thất bại, thông báo lỗi tương ứng được thiết lập vào state errors.
* Trong JSX, điều kiện render {errors.fieldName && <span>...</span>} được sử dụng để hiển thị thông báo lỗi ngay dưới trường input đó, giúp người dùng nhận biết và sửa lỗi kịp thời.

**7.5. Xử lý Gửi Form (Form Submission)**

* Sự kiện onSubmit của thẻ <form> được gắn với hàm handleSubmit.
* Hàm handleSubmit đầu tiên gọi e.preventDefault() để ngăn trình duyệt reload trang.
* Sau đó, hàm gọi validateForm() để kiểm tra tính hợp lệ của toàn bộ dữ liệu.
* Nếu form hợp lệ (isValid là true), dữ liệu trong formData sẽ được xử lý. Trong ví dụ này, chúng ta chỉ log ra console và hiện alert. Trong thực tế, tại đây bạn sẽ gọi một API để gửi dữ liệu về server.

**7.6. Kết luận và Hướng phát triển**

Component EventForm đã được xây dựng thành công bằng ReactJS, sử dụng Controlled Components để quản lý state của form. Form bao gồm các chức năng cơ bản: nhập liệu, xác thực theo thời gian thực, hiển thị lỗi và xử lý sự kiện gửi form.

**Hướng phát triển trong tương lai:**

1. **Sử dụng Thư viện Form chuyên nghiệp:** Đối với form phức tạp, có thể cân nhắc sử dụng các thư viện như Formik hoặc React Hook Form để giảm boilerplate code và tối ưu hiệu suất.
2. **API Integration:** Kết nối với backend thực tế thông qua các HTTP client như Axios hoặc Fetch API để lưu trữ và truy xuất dữ liệu.
3. **Loading State & Feedback:** Thêm trạng thái loading và thông báo kết quả (thành công/thất bại) cho người dùng sau khi gửi form.
4. **UI/UX Improvement:** Cải thiện giao diện người dùng với CSS framework (như Material-UI, Ant Design) hoặc custom CSS để form trực quan và thân thiện hơn.
5. **Kiểm thử (Testing):** Viết unit test cho component và hàm validation sử dụng Jest và React Testing Library.

**8. KẾT LUẬN VÀ XU HƯỚNG PHÁT TRIỂN**

ReactJS đã cách mạng hóa cách phát triển ứng dụng web với kiến trúc component-based và cơ chế Virtual DOM. Các khái niệm cơ bản như JSX, Components, Props, State và Lifecycle tạo nền tảng vững chắc cho việc xây dựng ứng dụng React hiệu quả.

**8.1. Tóm tắt điểm mạnh**

* **Hiệu suất**: Virtual DOM tối ưu hóa render
* **Tái sử dụng**: Component architecture cho phép tái sử dụng code
* **Dễ bảo trì**: Code được tổ chức rõ ràng, dễ hiểu
* **Cộng đồng**: Ecosystem phong phú với nhiều thư viện hỗ trợ

**8.2. Xu hướng phát triển**

* **Function components với Hooks**: Xu hướng chính thay thế class components
* **Server Components**: React Server Components cho render phía server
* **Concurrent Features**: Cải thiện khả năng responsive của ứng dụng

**8.3. 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. |

**8.4. Khuyến nghị**

**9. Router**

**9.1. Khái niệm React Router**

Trong ReactJS, ứng dụng thường được xây dựng dưới dạng **Single Page Application (SPA)**. Điều này có nghĩa là toàn bộ ứng dụng chỉ có một tệp HTML duy nhất, và việc điều hướng giữa các “trang” thực chất chỉ là thay đổi giao diện (component) hiển thị mà không cần tải lại toàn bộ trang.

Để xử lý điều hướng trong SPA, **React Router** ra đời như một thư viện tiêu chuẩn và phổ biến nhất. Nó cung cấp cơ chế quản lý URL, định tuyến và hiển thị các component phù hợp tương ứng với đường dẫn mà người dùng truy cập.

**9.2. Lợi ích của React Router**

* **Quản lý tuyến đường (routes) dễ dàng**: Cho phép định nghĩa nhiều đường dẫn và ánh xạ chúng tới các component.
* **Hiệu năng cao**: Không cần tải lại toàn bộ trang khi chuyển trang, chỉ cập nhật component cần thiết.
* **Hỗ trợ tham số trên URL**: Có thể truyền dữ liệu động qua URL (ví dụ: /product/:id).
* **Điều hướng (Navigation)**: Cung cấp các thẻ <Link> hoặc hook như useNavigate để điều hướng giữa các trang.
* **Nested Routes**: Hỗ trợ định nghĩa tuyến lồng nhau, giúp tổ chức cấu trúc giao diện phức tạp.

**9.3. Các thành phần chính trong React Router**

1. **BrowserRouter**:
   * Là thành phần bao bọc ứng dụng, giúp React Router quản lý lịch sử duyệt web bằng HTML5 history API.
   * Ví dụ:
   * import { BrowserRouter } from "react-router-dom";
   * function App() {
   * return (
   * <BrowserRouter>
   * <MyRoutes />
   * </BrowserRouter>
   * );
   * }
2. **Routes và Route**:
   * Routes chứa các Route con, mỗi Route sẽ ánh xạ một URL với một component.
   * Ví dụ:
   * import { Routes, Route } from "react-router-dom";
   * import Home from "./pages/Home";
   * import About from "./pages/About";
   * function MyRoutes() {
   * return (
   * <Routes>
   * <Route path="/" element={<Home />} />
   * <Route path="/about" element={<About />} />
   * </Routes>
   * );
   * }
3. **Link và NavLink**:
   * Dùng để tạo liên kết điều hướng mà không tải lại toàn bộ trang.
   * NavLink cho phép thêm style đặc biệt khi link đang được chọn.
   * Ví dụ:
   * import { Link, NavLink } from "react-router-dom";
   * <Link to="/about">Giới thiệu</Link>
   * <NavLink to="/about" className="active-link">Giới thiệu</NavLink>
4. **useNavigate**:
   * Là một hook cho phép điều hướng bằng code (chương trình), thay vì click vào link.
   * Ví dụ:
   * import { useNavigate } from "react-router-dom";
   * function Login() {
   * const navigate = useNavigate();
   * const handleLogin = () => {
   * // Sau khi đăng nhập thành công
   * navigate("/dashboard");
   * };
   * return <button onClick={handleLogin}>Đăng nhập</button>;
   * }
5. **useParams**:
   * Hook dùng để lấy tham số động từ URL.
   * Ví dụ: /product/:id → lấy id từ URL.
   * import { useParams } from "react-router-dom";
   * function ProductDetail() {
   * const { id } = useParams();
   * return <h2>Chi tiết sản phẩm: {id}</h2>;
   * }
6. **Nested Routes**:
   * Cho phép tạo các route lồng nhau để tái sử dụng layout.
   * Ví dụ:
   * <Routes>
   * <Route path="/dashboard" element={<Dashboard />}>
   * <Route path="profile" element={<Profile />} />
   * <Route path="settings" element={<Settings />} />
   * </Route>
   * </Routes>

**9.4. So sánh React Router với điều hướng truyền thống**

| **Tiêu chí** | **Điều hướng truyền thống (Multi-Page)** | **React Router (SPA)** |
| --- | --- | --- |
| Tải lại trang | Mỗi lần đổi URL phải tải lại toàn bộ trang | Không tải lại, chỉ thay đổi component |
| Tốc độ | Chậm hơn do phải tải nhiều lần | Nhanh, chỉ render phần thay đổi |
| Quản lý trạng thái | Reset lại sau mỗi lần tải | Giữ nguyên trạng thái ứng dụng |
| Trải nghiệm người dùng | Ít mượt mà hơn | Trải nghiệm giống ứng dụng desktop |

Đối với người mới học React, nên:

1. Bắt đầu với function components và hooks
2. Hiểu rõ sự khác biệt giữa props và state
3. Thực hành với các dự án nhỏ trước
4. Tận dụng React Dev Tools để debug

ReactJS tiếp tục là một trong những thư viện frontend phổ biến nhất và là kỹ năng quan trọng cho các developer web hiện đại.

**Tài liệu tham khảo:**

* React Official Documentation: <https://reactjs.org/>
* React Beta Documentation: <https://beta.reactjs.org/>
* "Learning React" by Alex Banks and Eve Porcello
* "React Up and Running" by Stoyan Stefanov