Mục Lục

[ĐỀ CƯƠNG BÁO CÁO 3](#_Toc20044)

[NGHIÊN CỨU TỔNG QUAN VỀ REACTJS 3](#_Toc9051)

[Mở đầu 3](#_Toc26584)

[Chương 1. Tổng quan về ReactJS 3](#_Toc17173)

[1 .Giới thiệu ReactJS 3](#_Toc7217)

[Chương 2. Virtual DOM 5](#_Toc24980)

[Là gì? 5](#_Toc11188)

[1. Render (Tái kết xuất): 5](#_Toc6446)

[2. Diffing (So sánh): 6](#_Toc23838)

[3. Reconciliation (Đối chiếu/Cập nhật): 6](#_Toc23651)

[Chương 3. JSX (JavaScript XML) 6](#_Toc27846)

[1. Viết mã: 6](#_Toc699)

[2. Chuyển đổi: 6](#_Toc22129)

[3. Tạo thành phần UI: 6](#_Toc6590)

[Chương 4. Components 9](#_Toc22270)

[Chương 5. Props và State 11](#_Toc635)

[1. React.useState<number>(0): 13](#_Toc11325)

[2. Hiển thị trạng thái: 13](#_Toc26746)

[3. Cập nhật trạng thái: 13](#_Toc3243)

[Chương 6. React Lifecycle 14](#_Toc17678)

[1. Khởi tạo (Initialization/Mounting): 14](#_Toc24047)

[2. Cập nhật (Updating): 14](#_Toc31328)

[3. Gỡ bỏ (Unmounting): 15](#_Toc10305)

[1. Mounting (Khởi tạo) 15](#_Toc11180)

[2. Updating (Cập nhật) 15](#_Toc30257)

[3. Unmounting (Hủy) 15](#_Toc25732)

[Chương 7. Ứng dụng thực tiễn của ReactJS 18](#_Toc2906)

[7.1 Các công ty lớn sử dụng ReactJS 18](#_Toc28439)

[7.2 Ứng dụng trong phát triển 19](#_Toc29198)

[Kết luận 20](#_Toc31186)

[BÁO CÁO 21](#_Toc7932)

[NGHIÊN CỨU CÁC THÀNH PHẦN CƠ BẢN TRONG REACTJS 21](#_Toc28366)

[Mở đầu 21](#_Toc9047)

[Chương 1. Tổng quan về ReactJS 21](#_Toc13290)

[Chương 2. JSX (JavaScript XML) 21](#_Toc26906)

[Chương 3. Components 21](#_Toc132)

[3.1. Function Component 21](#_Toc4678)

[3.2. Class Component 22](#_Toc7726)

[3.3. So sánh Function và Class Component 22](#_Toc7005)

[Chương 4. Thực hành tạo Components 22](#_Toc21456)

[Chương 5. Kết luận 23](#_Toc8802)

[Tài liệu tham khảo 23](#_Toc30488)

[ĐỀ CƯƠNG BÁO CÁO 24](#_Toc32548)

[EVENT FORM TRONG REACTJS 24](#_Toc15910)

[Chương 1. Tổng quan về Event trong ReactJS 24](#_Toc32418)

[1. Hệ thống SyntheticEvent: 24](#_Toc31698)

[2. Xử lý sự kiện: 24](#_Toc25289)

[3.Cú pháp sự kiện trong React 24](#_Toc774)

[4. So sánh với HTML truyền thống 25](#_Toc4309)

[Cú pháp 25](#_Toc4608)

[2. Cách xử lý sự kiện 25](#_Toc27686)

[3. Quản lý trạng thái và khả năng tái sử dụng 27](#_Toc27763)

[4. Khả năng bảo trì và mở rộng 27](#_Toc2113)

[5. Hiệu suất 27](#_Toc17376)

[6. Ví dụ minh họa 28](#_Toc29783)

[Kết luận 28](#_Toc18343)

[Chương 2. Các sự kiện phổ biến 28](#_Toc13195)

[1. Sự kiện chuột 28](#_Toc17046)

[2. Sự kiện bàn phím 30](#_Toc20674)

[3. Sự kiện form 31](#_Toc6661)

[4. Sự kiện khác 32](#_Toc12759)

[So sánh tổng quan 33](#_Toc25281)

[Lưu ý quan trọng 33](#_Toc11605)

[Chương 3. Form trong React 33](#_Toc8783)

[1. Khái niệm về Form 33](#_Toc16175)

[2. Controlled Component trong React 34](#_Toc18258)

[So sánh Form trong HTML truyền thống và Controlled Component trong React 35](#_Toc23567)

[Liên kết với các sự kiện Form 36](#_Toc31257)

[Ưu điểm và nhược điểm của Controlled Component 36](#_Toc8233)

[Uncontrolled Component (so sánh ngắn gọn) 37](#_Toc11934)

[Chương 4 Các Ví dụ bài làm 37](#_Toc15810)

[1. Giới thệu Event Handling trong ReactJS 37](#_Toc27518)

[2. Ví dụ 1: Function Component – Event không tham số 37](#_Toc28868)

[3. Ví dụ 2: Class Component – Event có và không có tham số 38](#_Toc5554)

[4. Ví dụ 3: Form cơ bản – Controlled Component 38](#_Toc10263)

[5. Ví dụ 4: Form tổng hợp nhiều trường dữ liệu 39](#_Toc20241)

[6. Kết luận 41](#_Toc23188)

[Kết luận 41](#_Toc9702)

[Báo cáo 42](#_Toc27858)

[REACTJS FORM TRONG REACTJS 42](#_Toc1558)

[Nguồn gốc: 42](#_Toc30239)

[Phát triển thêm: 42](#_Toc8373)

[Chương 2. JSX Khái niệm, cú pháp. 43](#_Toc26199)

[1. Khái niệm 43](#_Toc25843)

[2. Cú pháp 43](#_Toc23430)

[3. Ví dụ minh họa 43](#_Toc15819)

[Chương 3. Component Class component. Function component. 43](#_Toc11629)

[1. Class Component (Component Dạng Lớp) 43](#_Toc8673)

[2. Function Component (Component Dạng Hàm) 44](#_Toc9832)

[3. Ví dụ minh họa chung 45](#_Toc12128)

[Chương 4. Props và State Khái niệm, sự khác biệt. 45](#_Toc23619)

[1. Khái niệm và Sự khác biệt 45](#_Toc7143)

[2. Truyền dữ liệu qua Props 46](#_Toc28931)

[3. Quản lý dữ liệu với State 46](#_Toc16581)

[Chương 5. Event Handling 47](#_Toc689)

[Chương 6. Render List và Key 50](#_Toc3593)

[Chương 7. ReactJS Form 52](#_Toc2716)

[1. Component được kiểm soát (Controlled Component) 52](#_Toc17212)

[2. Component không kiểm soát (Uncontrolled Component) 52](#_Toc12223)

[3. Quản lý nhiều Input với State (Controlled Components) 53](#_Toc701)

[Kết luận 55](#_Toc27937)

**ĐỀ CƯƠNG BÁO CÁO**

**NGHIÊN CỨU TỔNG QUAN VỀ REACTJS**

**Mở đầu**

ReactJS là thư viện JavaScript mã nguồn mở của Meta, giúp xây dựng giao diện người dùng (UI) hiệu quả bằng cách sử dụng các thành phần (components) có thể tái sử dụng và phương pháp Virtual DOM để tối ưu hiệu suất. Các thành phần cốt lõi của React bao gồm JSX, Virtual DOM, Props & State và Lifecycle, mang lại lợi ích như hiệu suất cao, khả năng tái sử dụng mã, hỗ trợ đa nền tảng (cả web và mobile) và khả năng tương thích tốt với SEO. React hoạt động bằng cách sử dụng Virtual DOM để cập nhật UI một cách hiệu quả, chỉ áp dụng những thay đổi cần thiết lên DOM thật của trình duyệt.

**Chương 1. Tổng quan về ReactJS**

**1 .Giới thiệu ReactJS**

Lịch sử ra đời

Phát triển bởi Facebook:

ReactJS được phát triển bởi kỹ sư Jordan Walke tại Facebook (nay là Meta) và ban đầu được sử dụng nội bộ từ năm 2011 để giải quyết các vấn đề về hiệu suất và khả năng mở rộng của giao diện người dùng trên Facebook và Instagram.

Công bố mã nguồn mở:

Facebook đã công bố phát hành ReactJS dưới dạng mã nguồn mở vào năm 2013, mở đường cho sự phát triển mạnh mẽ trong cộng đồng.

Tính chất: Thư viện (Library)

Chỉ tập trung vào UI:

ReactJS là một thư viện JavaScript, có nghĩa là nó chỉ tập trung vào việc xây dựng các phần giao diện người dùng (view layer) của ứng dụng, khác với một framework (như Angular) có phạm vi rộng hơn, bao gồm cả các quy tắc và cấu trúc cho toàn bộ ứng dụng.

Kiến trúc dựa trên Component:

React cho phép chia giao diện thành các thành phần (components) độc lập, có thể tái sử dụng, giúp đơn giản hóa và tổ chức code hiệu quả.

Ứng dụng chính

Xây dựng giao diện người dùng (UI):

Ứng dụng chính của ReactJS là xây dựng các giao diện người dùng động và tương tác cho trang web.

Hỗ trợ ứng dụng di động:

Thông qua React Native, ReactJS cũng được sử dụng để phát triển các ứng dụng di động đa nền tảng với hiệu suất cao.

1. Ưu điểm của ReactJS

Hiệu năng cao nhờ Virtual DOM:

React sử dụng Virtual DOM, một bản sao ảo của DOM thật. Khi có thay đổi, React sẽ so sánh Virtual DOM với trạng thái trước đó và chỉ cập nhật những phần cần thiết lên DOM thật, giúp tăng tốc độ và hiệu suất ứng dụng.

Khả năng tái sử dụng component:

React cho phép chia nhỏ giao diện người dùng (UI) thành các thành phần độc lập, có thể tái sử dụng ở nhiều nơi trong cùng một dự án hoặc nhiều dự án khác nhau. Điều này giúp giảm thiểu việc viết lại mã, tiết kiệm thời gian và công sức phát triển, cũng như dễ dàng bảo trì và mở rộng ứng dụng.

Cộng đồng phát triển mạnh mẽ:

Là một thư viện JavaScript phổ biến, React có một cộng đồng lập trình viên lớn và năng động. Điều này đồng nghĩa với việc có nhiều tài liệu, công cụ, thư viện hỗ trợ, giúp việc học tập và giải quyết vấn đề trong quá trình phát triển trở nên dễ dàng hơn.

Hỗ trợ tốt cho SPA (Single Page Application):

React rất phù hợp để xây dựng các ứng dụng một trang (SPA). Với cấu trúc component và khả năng cập nhật DOM hiệu quả, React giúp tạo ra trải nghiệm người dùng mượt mà, phản hồi nhanh chóng, đặc biệt quan trọng cho các ứng dụng web phức tạp.

Hỗ trợ phát triển đa nền tảng:

Với React Native, kiến thức về React có thể được áp dụng để phát triển ứng dụng di động cho cả iOS và Android mà không cần viết lại nhiều mã, mang lại sự linh hoạt và hiệu quả cao.

1. So sánh với các framework khác

ReactJS vs Angular

React:

Là một thư viện JavaScript tập trung vào việc tạo các thành phần UI có thể tái sử dụng. Nó cho phép bạn tự do lựa chọn kiến trúc và các thư viện bổ sung, rất phù hợp cho các ứng dụng có giao diện tương tác phức tạp và cần tính linh hoạt cao.

Angular:

Là một framework (khuôn khổ) đầy đủ tính năng, cung cấp một giải pháp có cấu trúc cho các ứng dụng quy mô lớn và doanh nghiệp. Angular sử dụng TypeScript và bao gồm nhiều tính năng tích hợp sẵn cho việc phát triển ứng dụng web.

Khi nào chọn:

Chọn Angular cho các dự án doanh nghiệp phức tạp, đòi hỏi cấu trúc và quy mô lớn. Chọn React cho các ứng dụng cần tính linh hoạt, giao diện năng động và muốn tự do lựa chọn công cụ.

ReactJS vs VueJS

React:

Tập trung vào việc xây dựng các thành phần UI và sử dụng JSX để viết mã. React có một cộng đồng lớn và hệ sinh thái phong phú, cung cấp nhiều tài nguyên và giải pháp cho nhà phát triển.

Vue:

Là một framework dễ học, cung cấp các công cụ front-end toàn diện và sử dụng cú pháp mẫu HTML đơn giản. Vue cung cấp một hệ sinh thái được hỗ trợ chính thức, làm cho việc quản lý trạng thái và định tuyến trở nên dễ dàng.

Khi nào chọn:

Chọn Vue khi bạn cần một giải pháp dễ học, thiết lập nhanh chóng và một framework linh hoạt. Chọn React khi bạn cần một cộng đồng lớn, hệ sinh thái đa dạng và làm việc với JavaScript/JSX.

**Chương 2. Virtual DOM**

Khái niệm DOM (Mô hình Đối tượng Tài liệu)

Là gì?

DOM là một giao diện lập trình ứng dụng (API) định nghĩa cách các phần tử trong tài liệu HTML hoặc XML được biểu diễn như các đối tượng.

Cấu trúc:

Tài liệu được biểu diễn dưới dạng một cấu trúc cây (DOM Tree), trong đó mỗi nút (node) đại diện cho một phần tử, thuộc tính, hoặc văn bản trong tài liệu.

Cách hoạt động:

DOM cho phép các ngôn ngữ lập trình như JavaScript truy cập và thao tác với các đối tượng này để thay đổi nội dung, thuộc tính, kiểu dáng và sự kiện trên trang web một cách động.

Nhược điểm của DOM truyền thống

Hiệu suất kém khi thao tác DOM trực tiếp:

Khi bạn thực hiện bất kỳ thay đổi nào đối với các phần tử trong cây DOM thực tế (real DOM) bằng JavaScript, trình duyệt cần phải vẽ lại (re-render) toàn bộ các phần tử đó.

Tốn kém tài nguyên:

Việc vẽ lại này tiêu tốn nhiều tài nguyên và ảnh hưởng tiêu cực đến hiệu suất của ứng dụng web, đặc biệt là đối với các ứng dụng có nhiều tương tác hoặc cập nhật thường xuyên.

Tốc độ cập nhật chậm:

Do quá trình re-render trực tiếp này, DOM truyền thống không phù hợp cho các ứng dụng web phức tạp và yêu cầu hiệu năng cao, nơi mà tốc độ cập nhật giao diện là yếu tố quan trọng.

Cơ chế hoạt động chi tiết:

1. Render (Tái kết xuất):

Khi có bất kỳ thay đổi nào về trạng thái trong ứng dụng React, một cây Virtual DOM mới sẽ được tạo ra để phản ánh trạng thái đó.

2. Diffing (So sánh):

React sẽ so sánh cây Virtual DOM mới này với cây Virtual DOM cũ bằng cách sử dụng một thuật toán gọi là "diffing" để xác định chính xác những phần tử nào đã thay đổi.

3. Reconciliation (Đối chiếu/Cập nhật):

Sau khi tìm ra sự khác biệt, React sẽ chỉ cập nhật những phần tử bị thay đổi đó lên DOM thật (real DOM) trên trình duyệt, thay vì vẽ lại toàn bộ trang.

Lợi ích chính của Virtual DOM:

Tối ưu hiệu suất:

Việc chỉ cập nhật những thay đổi nhỏ trên DOM thật giúp giảm thiểu chi phí thao tác và giảm thời gian tái kết xuất giao diện, làm cho ứng dụng phản hồi nhanh hơn.

Tối ưu số lần thao tác DOM:

React sẽ gộp nhóm nhiều cập nhật (Batch Updates) và áp dụng chúng cùng lúc, tránh việc tạo ra quá nhiều bản cập nhật DOM thực tế riêng lẻ.

Cải thiện trải nghiệm nhà phát triển:

Nhà phát triển có thể tập trung vào việc xây dựng giao diện mà không cần can thiệp trực tiếp vào DOM thật, một quá trình thường phức tạp và tốn kém

Ưu điểm của Virtual DOM

Virtual DOM giúp cải thiện hiệu suất của ứng dụng React bằng cách giảm thiểu số lần cập nhật trực tiếp vào DOM thực tế. Thay vì vẽ lại toàn bộ trang web mỗi khi có sự thay đổi, React chỉ cập nhật những phần tử cần thiết, điều này giúp giảm tải cho trình duyệt và tăng tốc độ của ứng dụng.

**Chương 3. JSX (JavaScript XML)**

Cách thức hoạt động:

1. Viết mã:

Bạn viết mã JSX tương tự như HTML, ví dụ: const element = <h1>Hello, React!</h1>;.

2. Chuyển đổi:

Trình duyệt không hiểu trực tiếp JSX, nên cần các công cụ như Babel để chuyển đổi JSX thành các lệnh gọi hàm JavaScript thuần.

3. Tạo thành phần UI:

Sau khi chuyển đổi, các phần tử JSX trở thành các lệnh gọi hàm JavaScript thông thường (ví dụ: React.createElement), sau đó được sử dụng để tạo và hiển thị giao diện người dùng.

Lợi ích của JSX:

Dễ đọc và hiểu:

Cú pháp giống HTML giúp mã nguồn dễ đọc hơn, giống như việc viết các mẫu giao diện.

Trực quan hóa UI:

Giúp các nhà phát triển hình dung cấu trúc giao diện người dùng một cách rõ ràng trong chính mã JavaScript.

Tích hợp biểu thức JavaScript:

Cho phép bạn nhúng các biến, hàm và biểu thức JavaScript khác vào bên trong cú pháp giống HTML bằng dấu ngoặc nhọn {}.

Giúp tìm lỗi hiệu quả:

Các lỗi trong cú pháp JSX có thể được phát hiện trong quá trình biên dịch, giúp dễ dàng tìm và sửa lỗi hơn.

Những lợi ích chính của JSX:

Gọn gàng và dễ đọc:

Cú pháp giống HTML làm cho mã nguồn trực quan, dễ đọc, dễ hiểu và dễ quản lý hơn so với việc viết mã JavaScript thuần để tạo các phần tử UI.

Giúp giảm độ phức tạp khi phát triển giao diện người dùng, làm cho code JavaScript và HTML gần gũi và ngắn gọn hơn.

Tích hợp logic JavaScript dễ dàng:

Cho phép bạn chèn các biểu thức JavaScript, biến, điều kiện và vòng lặp trực tiếp vào mã JSX, giúp quản lý dữ liệu động và tương tác người dùng một cách liền mạch.

Quản lý Component hiệu quả:

JSX là một phần quan trọng trong kiến trúc dựa trên component của React.

Mỗi component có thể trả về JSX, mô tả giao diện riêng của nó, giúp việc chia nhỏ các cấu trúc UI phức tạp thành các component độc lập và dễ tái sử dụng hơn.

Phát hiện lỗi sớm:

Trong quá trình biên dịch, các lỗi cú pháp trong JSX, như quên đóng thẻ, sẽ được hiển thị ngay lập tức, giúp bạn phát hiện và sửa lỗi nhanh chóng thay vì gặp phải chúng ở thời điểm chạy.

Tối ưu hiệu suất:

JSX được biên dịch thành các lệnh JavaScript thông qua các trình biên dịch như Babel.

Khi kết hợp với Virtual DOM của React, quá trình này giúp cập nhật giao diện nhanh chóng và hiệu quả hơn, tránh thao tác trực tiếp lên DOM thông thường.

Ví dụ cơ bản

Đây là cách bạn khai báo một thẻ HTML và nhúng một biến JavaScript vào nó:

Mã

const name = "Dân"; // Khai báo biến JavaScript

const element = <h1>Xin chào, {name}!</h1>; // Sử dụng biến trong JSX

Trong ví dụ này, name được thay thế bằng giá trị "Dân" khi phần tử được hiển thị.

Nhúng biểu thức JavaScript phức tạp hơn

Bạn có thể sử dụng các biểu thức JavaScript như điều kiện, phép toán, hoặc gọi hàm bên trong dấu ngoặc nhọn:

Mã

const isLoggedIn = true;

const greeting = <h1>{isLoggedIn ? 'Chào mừng bạn!' : 'Vui lòng đăng nhập'}</h1>;

Trong ví dụ này, một ternary operator được sử dụng để hiển thị các lời chào khác nhau tùy thuộc vào giá trị của biến isLoggedIn.

Sử dụng thuộc tính JavaScript (Attribute)

Thay vì dùng class như trong HTML, JSX sử dụng className để gán các lớp CSS vào các phần tử:

Mã

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

JSX biên dịch thành các lệnh gọi hàm JavaScript, ví dụ như React.createElement('h1', {className: 'greeting'}, 'Hello, world!').

Render danh sách

Bạn có thể sử dụng phương thức map() của mảng để render một danh sách các phần tử:

Mã

const listItems = ['Sách', 'Vở', 'Bút'];

const myList = (

<ul>

{listItems.map((item, index) => (

<li key={index}>{item}</li>

))}

</ul>

);

Ở đây, mỗi phần tử trong mảng listItems được chuyển thành một thẻ <li>.

**Chương 4. Components**

Tái sử dụng và dễ bảo trì

Tái sử dụng:

Bạn có thể định nghĩa một component một lần và sử dụng lại nó ở nhiều nơi trong ứng dụng. Ví dụ, một component hiển thị nút bấm có thể được sử dụng lại trên nhiều trang mà không cần viết lại mã.

Dễ bảo trì:

Việc chia nhỏ UI thành các component riêng biệt giúp mã nguồn trở nên module hóa, dễ hiểu, dễ dàng sửa lỗi và cập nhật mà không ảnh hưởng đến các phần khác của ứng dụng.

Cấu trúc và cách hoạt động

Khối xây dựng cơ bản:

Mọi ứng dụng React được xây dựng từ nhiều component.

Trả về React elements:

Component trả về các React elements (thường dưới dạng JSX) để hiển thị ra giao diện người dùng.

Hai loại chính:

Functional Components: Là các hàm JavaScript nhận vào các props (thuộc tính) và trả về UI. Chúng là dạng component được khuyên dùng hiện nay, đặc biệt là khi có sự hỗ trợ của Hooks.

Class Components: Là các lớp JavaScript có khả năng quản lý trạng thái (state) và sử dụng các phương thức vòng đời.

Cấu trúc cây:

Các component thường được tổ chức theo cấu trúc dạng cây, tương tự như cấu trúc của Virtual DOM.

Trong React, có hai loại Component chính là Function Component và Class Component. Function Component là các hàm JavaScript đơn giản nhận props và trả về JSX, không có state hay lifecycle methods. Ngược lại, Class Component là các class kế thừa từ React.Component, có thể quản lý state và sử dụng các lifecycle methods để xử lý logic phức tạp.

1. Function Component

Định nghĩa: Là một hàm JavaScript nhận một đối số là props (nếu cần) và trả về một phần tử React (thường là JSX).

Đặc điểm:

Đơn giản, dễ viết hơn.

Không có state hay lifecycle methods riêng.

Thích hợp cho các component chỉ hiển thị giao diện, nhận dữ liệu từ props.

Ví dụ:

Mã

function Greeting(props) {

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

}

2. Class Component

Định nghĩa: Là một class trong JavaScript kế thừa từ React.Component.

Đặc điểm:

Có thể sử dụng state để quản lý dữ liệu nội tại của component.

Có các lifecycle methods (ví dụ: componentDidMount, componentDidUpdate) để quản lý vòng đời của component.

Sử dụng this.props để truy cập props và this.state để truy cập state.

Ví dụ:

Mã

import React from 'react';

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

render() {

return <div>Count: {this.state.count}</div>;

}

}

Ví dụ minh họa Component.

Component là các thành phần độc lập, có thể tái sử dụng, lắp ráp với nhau để tạo thành một hệ thống lớn hơn; ví dụ minh họa bao gồm các nút bấm, thanh trượt trong giao diện ứng dụng (giao diện người dùng), bu lông, bánh răng trong máy móc (kỹ thuật cơ khí), hoặc nguyên tử, phân tử trong vật chất (khoa học). Trong lập trình web và di động, các component có thể là thanh điều hướng, thẻ sản phẩm, hoặc biểu đồ, mỗi component chứa cả giao diện (HTML, CSS) và logic xử lý (JavaScript, TypeScript).

Ví dụ trong lập trình:

Giao diện người dùng (UI): Một trang web hoặc ứng dụng di động có thể được chia thành các component như:

Thanh tìm kiếm (Search Box): Một component có chức năng nhận input và tìm kiếm.

Danh sách sản phẩm: Một component lớn hơn, được tạo thành từ nhiều component con nhỏ hơn, mỗi component con là một "thẻ sản phẩm".

Menu thả xuống (Dropdown Menu): Một component tương tác với người dùng, hiển thị các tùy chọn khi được nhấp vào.

Thành phần UI ảo: Component trong các framework như React hoặc Angular, được chuyển thành các phần tử HTML gốc như UIView trên iOS hoặc TextView trên Android, giúp xử lý dữ liệu và giao diện độc lập.

Ví dụ trong các lĩnh vực khác:

Kỹ thuật:

Một chiếc xe hơi là một hệ thống phức tạp được tạo thành từ các component cơ khí như động cơ, bánh xe, hệ thống lái.

Khoa học:

Nguyên tử là component cơ bản nhất của vật chất, và chúng kết hợp lại để tạo thành phân tử, sau đó là các cấu trúc lớn hơn như tế bào.

Kỹ thuật cơ khí:

Các ổ trục, đai ốc, và bu lông là những component có thể lắp ráp để tạo thành máy móc.

**Chương 5. Props và State**

Props là viết tắt của "Properties", được sử dụng để truyền dữ liệu một chiều từ component cha xuống component con, và dữ liệu này là bất biến (không thể bị thay đổi bởi component con). Bạn có thể dùng Props để truyền các thuộc tính hoặc cấu hình cho component con, giúp chúng linh hoạt và có thể tái sử dụng.

Ví dụ minh họa Props:

Trong ví dụ sau, component App (cha) truyền giá trị cho thuộc tính name xuống component Greeting (con).

TypeScript

// Định nghĩa kiểu dữ liệu cho props mà component Greeting sẽ nhận

interface GreetingProps {

name: string;

}

// Component Greeting (con) nhận props 'name' từ component cha

const Greeting: React.FC<GreetingProps> = ({ name }) => {

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

};

// Component App (cha)

const App: React.FC = () => {

return (

<div>

{/\* Truyền props 'name' với giá trị "Hello" \*/}

<Greeting name="Hello" />

{/\* Truyền props 'name' với giá trị "200Lab" \*/}

<Greeting name="200Lab" />

</div>

);

};

export default App;

Giải thích ví dụ:

GreetingProps:

Khai báo kiểu dữ liệu cho các props mà component Greeting nhận, ở đây là một thuộc tính name có kiểu string.

Greeting Component:

Component này nhận thuộc tính name qua props và hiển thị lời chào tương ứng.

App Component:

Component App sử dụng component Greeting hai lần, mỗi lần truyền một giá trị name khác nhau ("Hello" và "200Lab"). Điều này cho thấy tính tái sử dụng của component và cách truyền dữ liệu từ cha xuống con bằng props.

State trong React là một đối tượng chứa dữ liệu nội bộ của component, có thể thay đổi được (mutable) theo thời gian và có chức năng tự động cập nhật lại giao diện người dùng (UI) khi dữ liệu đó thay đổi. State giúp quản lý các dữ liệu động trong ứng dụng, chẳng hạn như dữ liệu người dùng nhập, dữ liệu từ API, hoặc bất kỳ trạng thái nào thay đổi trong quá trình sử dụng ứng dụng.

Ví dụ minh họa State trong React

Dưới đây là ví dụ về một component Counter sử dụng useState hook để quản lý trạng thái bộ đếm:

JavaScript

import React from 'react';

const Counter: React.FC = () => {

// Khai báo state 'count' với giá trị ban đầu là 0

// setCount là hàm để cập nhật state 'count'

const [count, setCount] = React.useState<number>(0);

return (

<div>

<p>Bạn đã nhấp {count} lần</p>

{/\* Khi nút được nhấp, gọi setCount để cập nhật giá trị count \*/}

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

Nhấp vào tôi

</button>

</div>

);

};

export default Counter;

Giải thích ví dụ:

1. React.useState<number>(0):

Chúng ta sử dụng hook useState để tạo một biến state mới tên là count và một hàm để cập nhật nó là setCount.

Giá trị ban đầu của count được khởi tạo là 0.

2. Hiển thị trạng thái:

count được hiển thị trong thẻ <p> để người dùng biết giá trị hiện tại của bộ đếm.

3. Cập nhật trạng thái:

Khi người dùng nhấp vào nút "Nhấp vào tôi", hàm onClick sẽ được gọi.

setCount(count + 1) sẽ cập nhật giá trị của count bằng cách tăng nó lên 1.

React sẽ tự động nhận thấy sự thay đổi này và render lại component Counter, hiển thị giá trị count mới trên giao diện.

+-------------------+-------------------------------------+-------------------------------------+

| Đặc điểm | Props (Properties) | State |

+-------------------+-------------------------------------+-------------------------------------+

| Định nghĩa | Dữ liệu truyền từ cha xuống con. | Dữ liệu nội bộ, do component quản lý|

+-------------------+-------------------------------------+-------------------------------------+

| Khả năng thay đổi | Bất biến, không thể thay đổi. | Có thể thay đổi bằng setState. |

+-------------------+-------------------------------------+-------------------------------------+

| Truyền/truy cập | Từ bên ngoài vào như tham số hàm. | Tạo và quản lý trong component. |

+-------------------+-------------------------------------+-------------------------------------+

| Mục đích | Truyền dữ liệu, cấu hình từ cha. | Quản lý dữ liệu động nội bộ. |

+-------------------+-------------------------------------+-------------------------------------+

| Quy tắc | Luồng dữ liệu một chiều, từ trên. | Riêng tư, chỉ component truy cập. |

+-------------------+-------------------------------------+-------------------------------------+

| Ví dụ | Dữ liệu người dùng từ danh sách. | Trạng thái nút: "mở" hoặc "đóng". |

+-------------------+-------------------------------------+-------------------------------------+

**Chương 6. React Lifecycle**

Các giai đoạn chính trong vòng đời component:

1. Khởi tạo (Initialization/Mounting):

Đây là giai đoạn khi component được tạo ra và "gắn kết" vào DOM.

Các phương thức như constructor (hoặc tương đương) được gọi để khai báo state ban đầu và bind các hàm của component.

Sau đó, phương thức render được gọi để hiển thị component ra ngoài trình duyệt.

Cuối cùng, componentDidMount (ở React) được gọi một lần duy nhất, là nơi lý tưởng để thực hiện các tác vụ như gọi API để lấy dữ liệu, hoặc thiết lập các subscription.

2. Cập nhật (Updating):

Giai đoạn này xảy ra khi có sự thay đổi về state hoặc props của component.

Các phương thức như shouldComponentUpdate kiểm tra xem component có cần được render lại hay không.

Sau đó, hàm render sẽ được gọi lại để cập nhật giao diện.

Phương thức componentDidUpdate được gọi sau khi component đã được render lại, cho phép thực hiện các hiệu ứng phụ như gửi yêu cầu mạng hoặc cập nhật state dựa trên dữ liệu mới.

3. Gỡ bỏ (Unmounting):

Đây là giai đoạn cuối cùng, khi component bị loại bỏ khỏi DOM.

Phương thức componentWillUnmount được gọi để thực hiện các tác vụ dọn dẹp, như hủy bỏ các timer, các request mạng chưa hoàn thành, hoặc ngắt kết nối subscription để tránh rò rỉ bộ nhớ.

Các giai đoạn:

Ba giai đoạn Mounting (khởi tạo), Updating (cập nhật), và Unmounting (hủy) là các giai đoạn chính trong vòng đời của một thành phần (component) trong React. Giai đoạn Mounting là khi thành phần được tạo ra và chèn vào DOM. Giai đoạn Updating là khi thành phần được cập nhật do thay đổi về state hoặc props. Giai đoạn Unmounting là khi thành phần bị loại bỏ khỏi DOM.

1. Mounting (Khởi tạo)

Mục đích: Tạo và chèn một instance của component vào DOM.

Diễn biến: Các phương thức được gọi trong giai đoạn này bao gồm constructor() để khởi tạo state và liên kết các hàm xử lý sự kiện.

Ví dụ: Khi ứng dụng bắt đầu chạy, các thành phần ban đầu được mount để hiển thị trên giao diện người dùng.

2. Updating (Cập nhật)

Mục đích: Cập nhật lại component khi state hoặc props của nó thay đổi.

Diễn biến: React sẽ chạy các phương thức vòng đời để cập nhật lại giao diện dựa trên các thay đổi mới.

Ví dụ: Khi người dùng nhập liệu vào một ô tìm kiếm, state của component đó thay đổi, dẫn đến việc component được cập nhật để phản ánh dữ liệu mới.

3. Unmounting (Hủy)

Mục đích: Xóa một instance của component khỏi DOM.

Diễn biến: Phương thức như componentWillUnmount() sẽ được gọi để thực hiện dọn dẹp như vô hiệu hóa các bộ hẹn giờ, hủy các yêu cầu mạng hoặc xóa các đăng ký.

Ví dụ: Khi người dùng đóng một modal hoặc chuyển sang một trang khác, các thành phần liên quan đến modal đó sẽ được unmount.

Các phương thức lifecycle trong Class Component:

Trong React Class Component, componentDidMount được gọi sau khi component được render lên DOM, là nơi lý tưởng để thực hiện các thao tác phụ như gọi API hay thiết lập đăng ký. componentDidUpdate được gọi sau khi component đã cập nhật và render lại, thường dùng để xử lý các side effect dựa trên sự thay đổi của props hoặc state. componentWillUnmount (hiện nay là componentWillUnmount trong các phiên bản mới hơn) là phương thức cuối cùng được gọi ngay trước khi component bị xóa khỏi DOM, dùng để dọn dẹp các tài nguyên như timeout, đăng ký.

1. componentDidMount

Mục đích:

Được thực thi sau khi component được gắn kết (render) vào DOM.

Sử dụng:

Thực hiện các yêu cầu mạng (AJAX/API calls) để lấy dữ liệu.

Thiết lập các đăng ký (subscriptions) như từ kho Redux.

Gọi this.setState() để cập nhật trạng thái ban đầu.

Khởi động các hàm với độ trễ như setTimeout hoặc setInterval.

2. componentDidUpdate

Mục đích:

Được gọi ngay sau khi component đã được cập nhật và render lại, có thể do thay đổi props hoặc state.

Sử dụng:

Thực hiện các hành động phụ thuộc vào sự thay đổi của props hay state.

Điều kiện tiên quyết để gọi componentDidUpdate là kiểm tra sự thay đổi của props hoặc state để tránh vòng lặp vô hạn.

3. componentWillUnmount

Mục đích:

Được gọi ngay trước khi component bị gỡ bỏ (unmount) khỏi DOM.

Sử dụng:

Dọn dẹp các tài nguyên đã được tạo trong componentDidMount, ví dụ như hủy các đăng ký, timeout.

Ngăn chặn memory leak bằng cách giải phóng bộ nhớ hoặc tài nguyên không cần thiết.

Hooks trong Function Component (thay thế Lifecycle)

là các React Hooks cho phép bạn quản lý state và thực hiện tác vụ phụ (side effects) trong Function Component, thay thế cho các phương thức vòng đời của Class Component. useState dùng để khai báo và cập nhật biến trạng thái (state), trong khi useEffect cho phép bạn chạy các đoạn mã sau mỗi lần render, tương tự như componentDidMount, componentDidUpdate, và componentWillUnmount.

1. useState Hook

Mục đích: Quản lý trạng thái (state) bên trong một Functional Component.

Cách sử dụng:

Bạn gọi useState với giá trị khởi tạo ban đầu cho state.

Nó trả về một mảng có hai phần tử:

Biến state hiện tại (giá trị state).

Hàm để cập nhật biến state đó.

Ví dụ:

JavaScript

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0); // Khởi tạo state count với giá trị ban đầu là 0

return (

<div>

<p>Số lần click: {count}</p>

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

Tăng

</button>

</div>

);

}

Trong ví dụ trên, setCount sẽ cập nhật giá trị của count và làm component re-render để hiển thị số mới.

2. useEffect Hook

Mục đích: Thực hiện các tác vụ phụ như lấy dữ liệu, thao tác DOM, hoặc đăng ký các subscription sau khi component render xong.

Cách sử dụng:

Bạn truyền một hàm (callback function) vào useEffect. Hàm này sẽ chạy sau khi render.

useEffect có thể nhận một mảng dependencies (mảng các giá trị).

Ví dụ:

JavaScript

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

function ExampleComponent() {

const [data, setData] = useState(null);

useEffect(() => {

// Thực hiện việc lấy dữ liệu sau mỗi lần render

fetch('https://api.example.com/data')

.then(response => response.json())

.then(data => setData(data));

// Hàm cleanup: chạy khi component unmount hoặc trước lần render tiếp theo

return () => {

console.log('Component unmounted or re-rendered');

};

}, []); // Mảng rỗng [] nghĩa là effect chỉ chạy một lần sau mount và cleanup chạy khi unmount.

return (

<div>

{data ? <p>Dữ liệu: {JSON.stringify(data)}</p> : <p>Đang tải...</p>}

</div>

);

}

Nếu không có mảng dependency, effect sẽ chạy sau mỗi lần render.

Với mảng rỗng [], effect chỉ chạy một lần sau khi component được mount (tương tự componentDidMount).

Hàm trả về từ useEffect sẽ được gọi là hàm cleanup, tương tự như componentWillUnmount.

**Chương 7. Ứng dụng thực tiễn của ReactJS**

7.1 Các công ty lớn sử dụng ReactJS

ReactJS đã được áp dụng rộng rãi bởi nhiều tập đoàn công nghệ hàng đầu thế giới nhờ hiệu năng cao, khả năng mở rộng và cộng đồng phát triển mạnh mẽ:

Facebook: Chính là “cha đẻ” của ReactJS, Facebook sử dụng React để phát triển giao diện người dùng cho mạng xã hội Facebook và ứng dụng Messenger.

Instagram: Toàn bộ ứng dụng web Instagram được xây dựng trên ReactJS, với nhiều tính năng phức tạp như tìm kiếm, bản đồ, bộ lọc ảnh, bình luận theo thời gian thực.

Netflix: Ứng dụng ReactJS để tối ưu hóa tốc độ khởi tạo trang và nâng cao trải nghiệm người dùng trên TV và Web.

Airbnb: Sử dụng ReactJS trong hệ thống tìm kiếm, đặt phòng, và quản lý tài khoản để đảm bảo tính linh hoạt và giao diện thân thiện.

Uber Eats: Phần giao diện người dùng của Uber Eats cũng áp dụng React để xử lý các thao tác thời gian thực.

Việc các công ty lớn tin tưởng và sử dụng ReactJS đã chứng minh tính ổn định, bền vững và sức mạnh của thư viện này trong phát triển web hiện đại.

7.2 Ứng dụng trong phát triển

SPA (Single Page Application)

ReactJS hỗ trợ xây dựng các ứng dụng web một trang, giúp người dùng thao tác mượt mà mà không phải tải lại toàn bộ trang.

SPA với React tối ưu trải nghiệm người dùng nhờ cơ chế Virtual DOM và Routing thông minh.

PWA (Progressive Web Application)

Kết hợp ReactJS với Service Workers và các công cụ như Workbox, developer có thể tạo ra ứng dụng web hoạt động như ứng dụng di động, chạy offline, có thể cài đặt trên màn hình chính.

Ví dụ: Twitter Lite là một PWA được xây dựng với công nghệ React.

Mobile App với React Native

React Native là “người anh em” của ReactJS, cho phép xây dựng ứng dụng di động native (Android, iOS) sử dụng cùng tư duy Component của React.

Lợi ích: Tái sử dụng code giữa web và mobile, rút ngắn thời gian phát triển, tiết kiệm chi phí.

Ví dụ: Ứng dụng di động của Facebook, Instagram, Skype… đều phát triển một phần với React Native.

**Kết luận**

Qua quá trình nghiên cứu, có thể thấy rằng ReactJS là một trong những thư viện JavaScript mạnh mẽ và phổ biến nhất trong phát triển giao diện web hiện đại.

Ưu điểm nổi bật:

Tốc độ và hiệu năng cao nhờ Virtual DOM.

Tái sử dụng component, dễ bảo trì, dễ mở rộng.

Cộng đồng rộng lớn, tài liệu phong phú.

Phù hợp cho cả Web, SPA, PWA, và Mobile App.

Nhược điểm:

ReactJS chỉ là thư viện giao diện, cần tích hợp với thư viện khác (Redux, React Router…) để xây dựng ứng dụng hoàn chỉnh.

Tốc độ thay đổi phiên bản nhanh, gây khó khăn cho người mới học.

Đường cong học tập (learning curve) ban đầu có thể cao với lập trình viên chưa quen ES6+.

👉 Khẳng định: Với tính linh hoạt, hiệu suất, khả năng mở rộng và sự hỗ trợ mạnh mẽ từ cộng đồng, ReactJS xứng đáng là công cụ hàng đầu cho các lập trình viên khi xây dựng ứng dụng web và di động hiện đại.

**BÁO CÁO**

**NGHIÊN CỨU CÁC THÀNH PHẦN CƠ BẢN TRONG REACTJS**

## Mở đầu

ReactJS là một thư viện JavaScript mạnh mẽ, được sử dụng rộng rãi trong phát triển giao diện người dùng (UI). Các thành phần cốt lõi của React bao gồm JSX và Components. Đây chính là nền tảng giúp React trở thành một công cụ hiệu quả, linh hoạt và dễ dàng bảo trì. Báo cáo này tập trung vào việc nghiên cứu hai thành phần cơ bản đó để hiểu rõ hơn cách ReactJS hoạt động và ứng dụng trong thực tiễn.

## Chương 1. Tổng quan về ReactJS

ReactJS được phát triển bởi Facebook (nay là Meta) vào năm 2011 và được công bố mã nguồn mở vào năm 2013. React không phải là một framework mà là một thư viện, tập trung vào việc xây dựng giao diện người dùng. Điểm mạnh của React nằm ở khả năng tái sử dụng component và tốc độ cập nhật giao diện nhờ Virtual DOM.

Trong kiến trúc của React, giao diện người dùng được chia thành nhiều component nhỏ. Mỗi component là một khối độc lập, dễ bảo trì, dễ tái sử dụng và có thể lồng ghép với nhau để tạo thành ứng dụng lớn.

## Chương 2. JSX (JavaScript XML)

JSX là một cú pháp mở rộng cho JavaScript, cho phép viết mã trông giống HTML ngay trong code JavaScript. Trình duyệt không hiểu JSX, do đó cần Babel biên dịch JSX thành các lệnh gọi hàm JavaScript thuần (React.createElement).

Ví dụ minh họa JSX:

const element = <h1>Xin chào, React!</h1>;  
// Babel biên dịch thành:  
const element = React.createElement("h1", null, "Xin chào, React!");

Ưu điểm của JSX:  
- Dễ đọc, dễ hiểu, trực quan.  
- Cho phép nhúng biểu thức JavaScript trong dấu ngoặc nhọn {}.  
- Giúp phát hiện lỗi sớm trong quá trình biên dịch.

## Chương 3. Components

Components là khối xây dựng cơ bản của ứng dụng React. Mỗi component có thể coi như một hàm độc lập, có thể tái sử dụng ở nhiều nơi. Có hai loại chính là Function Component và Class Component.

### **3.1. Function Component**

Function Component chỉ là một hàm JavaScript nhận props và trả về JSX. Đây là cách viết component đơn giản và hiện đại, thường kết hợp với React Hooks.

Ví dụ:

function Welcome(props) {  
 return <h1>Xin chào, {props.name}!</h1>;  
}

Ví dụ với useState:

import { useState } from 'react';  
  
function Counter() {  
 const [count, setCount] = useState(0);  
 return (  
 <div>  
 <p>Bạn đã nhấp {count} lần</p>  
 <button onClick={() => setCount(count + 1)}>Nhấp</button>  
 </div>  
 );  
}

### 3.2. Class Component

Class Component là cách khai báo component bằng class trong JavaScript, kế thừa từ React.Component. Class Component có thể quản lý state và sử dụng các phương thức vòng đời (lifecycle).

Ví dụ:

import React, { Component } from 'react';  
  
class Counter extends Component {  
 constructor(props) {  
 super(props);  
 this.state = { count: 0 };  
 }  
  
 render() {  
 return (  
 <div>  
 <p>Bạn đã nhấp {this.state.count} lần</p>  
 <button onClick={() => this.setState({ count: this.state.count + 1 })}>Nhấp</button>  
 </div>  
 );  
 }  
}

### 3.3. So sánh Function và Class Component

Function Component:  
- Ngắn gọn, dễ viết.  
- Quản lý state bằng Hooks (useState, useEffect).  
  
Class Component:  
- Cú pháp dài hơn.  
- Quản lý state bằng this.state, setState.  
- Có các phương thức vòng đời như componentDidMount, componentDidUpdate.

## Chương 4. Thực hành tạo Components

1. Tạo dự án React bằng lệnh: npx create-react-app my-app  
2. Mở thư mục dự án trong Visual Studio Code.  
3. Sửa file App.js để sử dụng các component mới.  
4. Tạo một component Greeting và một component Counter.  
5. Chạy ứng dụng bằng npm start .

## Chương 5. Kết luận

JSX và Components là hai thành phần cơ bản nhất trong ReactJS. JSX giúp việc viết giao diện dễ dàng, trực quan. Components giúp tổ chức ứng dụng thành các khối độc lập, tái sử dụng và dễ bảo trì. Trong đó, Function Component kết hợp với Hooks là xu hướng phát triển hiện nay.

## Tài liệu tham khảo

1. ReactJS Official Documentation: https://react.dev  
2. W3Schools React Tutorial: https://www.w3schools.com/react/  
3. FreeCodeCamp: https://www.freecodecamp.org  
4. Medium Blogs về ReactJS

**ĐỀ CƯƠNG BÁO CÁO**

**EVENT FORM TRONG REACTJS**

**Chương 1. Tổng quan về Event trong ReactJS**

Trong React, "event" (sự kiện) là các hành động mà người dùng thực hiện trên giao diện người dùng (UI), như click chuột, nhập liệu vào form, hoặc gửi form. React xử lý các sự kiện này thông qua hệ thống SyntheticEvent, một lớp "vỏ bọc" đa trình duyệt để đảm bảo hoạt động nhất quán trên các trình duyệt khác nhau. Khi một sự kiện xảy ra, một hàm xử lý sự kiện được gọi để phản hồi lại hành động đó, cho phép bạn xây dựng các ứng dụng web tương tác.

1. Hệ thống SyntheticEvent:

React sử dụng SyntheticEvent, một trình bao bọc xung quanh sự kiện gốc của trình duyệt, cung cấp một API thống nhất cho các sự kiện này bất kể trình duyệt nào.

2. Xử lý sự kiện:

Bạn định nghĩa các hàm xử lý (event handlers) để chạy khi một sự kiện xảy ra.

Đặt tên quy ước: Các thuộc tính sự kiện dùng tên theo kiểu camelCase (ví dụ: onClick, onChange).

Truyền hàm xử lý: Hàm xử lý được truyền dưới dạng một prop (thuộc tính) cho component.

Ví dụ về các sự kiện phổ biến

onClick: Kích hoạt khi một phần tử được nhấp.

onChange: Kích hoạt khi giá trị của một trường nhập thay đổi.

onSubmit: Kích hoạt khi một form được gửi đi.

onKeyDown: Kích hoạt khi một phím được nhấn xuống.

Xử lý hành động mặc định

Trong nhiều trường hợp, các thành phần có hành vi mặc định đi kèm với một sự kiện (ví dụ: gửi form sẽ làm tải lại trang).

Bạn có thể sử dụng event.preventDefault() bên trong hàm xử lý để ngăn chặn hành vi mặc định này và thay thế bằng logic tùy chỉnh của riêng bạn.

3.Cú pháp sự kiện trong React

Trong React, cú pháp xử lý sự kiện sử dụng tên sự kiện viết hoa dạng camelCase (ví dụ: onClick, onChange) và truyền một hàm callback vào prop của sự kiện thay vì một chuỗi.

Ví dụ:

Mã

<button onClick={handleClick}>

Nhấn vào đây

</button>

Trong ví dụ trên:

onClick là tên sự kiện, được viết theo dạng camelCase.

handleClick là một hàm callback được truyền vào prop onClick để thực thi khi sự kiện xảy ra.

1. So sánh với HTML truyền thống

### ****Cú pháp****

* **HTML truyền thống**:

html

<button onclick="handleClick()">Click</button>

* + Sử dụng thuộc tính onclick (viết thường) để gắn một hàm JavaScript (handleClick()) trực tiếp vào phần tử HTML.
  + Tên hàm được gọi dưới dạng chuỗi, và thường được định nghĩa trong một tệp JavaScript riêng hoặc trong thẻ <script>.
* **React**:

jsx

<button onClick={handleClick}>Click</button>

* + Sử dụng cú pháp JSX, một phần mở rộng của JavaScript, trông giống HTML nhưng được biên dịch thành JavaScript.
  + Thuộc tính sự kiện được viết theo kiểu **camelCase** (onClick thay vì onclick).
  + Giá trị của onClick là một **tham chiếu đến hàm** (handleClick) chứ không phải chuỗi hay gọi hàm trực tiếp (handleClick()).

### 2. ****Cách xử lý sự kiện****

* **HTML truyền thống**:
  + Sự kiện được xử lý trực tiếp trong DOM, thông qua thuộc tính onclick được gắn vào phần tử.
  + Hàm handleClick phải được định nghĩa trong phạm vi toàn cục (global scope) hoặc được truy cập qua chuỗi mã JavaScript.
  + Dễ dẫn đến vấn đề về phạm vi (scope) hoặc xung đột tên hàm nếu không quản lý tốt.
  + Hiệu suất có thể kém hơn trong các ứng dụng lớn do sự kiện được gắn trực tiếp vào DOM.
* **React**:
  + React sử dụng **Synthetic Events** (sự kiện tổng hợp), một lớp bao bọc quanh các sự kiện DOM gốc, giúp chuẩn hóa hành vi sự kiện giữa các trình duyệt.
  + Hàm handleClick được truyền dưới dạng tham chiếu, cho phép React quản lý sự kiện hiệu quả hơn thông qua **Virtual DOM**.
  + React kiểm soát việc gắn và gỡ sự kiện (event delegation), giảm số lượng trình xử lý sự kiện trực tiếp trên DOM, cải thiện hiệu suất.

### 3. ****Quản lý trạng thái và khả năng tái sử dụng****

* **HTML truyền thống**:
  + Không có cơ chế quản lý trạng thái tích hợp. Nếu muốn cập nhật giao diện sau khi click, bạn cần thao tác DOM thủ công (ví dụ: document.getElementById).
  + Khó tái sử dụng logic hoặc giao diện vì HTML tĩnh không hỗ trợ các thành phần (components).
* **React**:
  + React khuyến khích sử dụng **components** và quản lý trạng thái thông qua useState hoặc các cơ chế khác.
  + Hàm handleClick thường được định nghĩa trong phạm vi của component, dễ dàng truy cập trạng thái (state) hoặc props, giúp logic xử lý sự kiện linh hoạt và tái sử dụng.
  + Khi trạng thái thay đổi, React tự động cập nhật giao diện thông qua cơ chế **re-rendering**.

### 4. ****Khả năng bảo trì và mở rộng****

* **HTML truyền thống**:
  + Phù hợp cho các trang web tĩnh hoặc ứng dụng nhỏ, nơi logic đơn giản.
  + Khi ứng dụng phức tạp, việc duy trì mã JavaScript và HTML riêng lẻ trở nên khó khăn, dễ dẫn đến mã "spaghetti".
* **React**:
  + Phù hợp cho các ứng dụng giao diện người dùng phức tạp, nơi cần quản lý trạng thái và tái sử dụng component.
  + JSX cho phép kết hợp giao diện và logic trong cùng một tệp, giúp mã dễ đọc và bảo trì hơn.
  + Hỗ trợ các công cụ như hooks (useState, useEffect) để xử lý logic phức tạp.

### 5. ****Hiệu suất****

* **HTML truyền thống**:
  + Mỗi phần tử có thể có một trình xử lý sự kiện riêng, dẫn đến tiêu tốn tài nguyên khi có nhiều phần tử.
  + Thao tác DOM trực tiếp (như element.innerHTML) thường chậm trong ứng dụng lớn.
* **React**:
  + Sử dụng Virtual DOM để giảm thiểu thao tác trực tiếp trên DOM thật, chỉ cập nhật những phần thay đổi.
  + Event delegation giúp giảm số lượng trình xử lý sự kiện, cải thiện hiệu suất trong các ứng dụng lớn.

### 6. ****Ví dụ minh họa****

* **HTML truyền thống**:

html

<button onclick="handleClick()">Click</button><script> function handleClick() { alert("Button clicked!"); }</script>

* **React**:

jsx

function Button() { const handleClick = () => { alert("Button clicked!"); }; return <button onClick={handleClick}>Click</button>;}

### Kết luận

* **HTML truyền thống** phù hợp cho các trang web đơn giản, tĩnh, hoặc khi không cần quản lý trạng thái phức tạp. Tuy nhiên, nó thiếu linh hoạt và khó bảo trì trong các ứng dụng lớn.
* **React** cung cấp cách tiếp cận hiện đại, mạnh mẽ hơn, với JSX, Virtual DOM, và cơ chế quản lý trạng thái, phù hợp cho các ứng dụng giao diện người dùng động và phức tạp.

Nếu bạn cần giải thích chi tiết hơn về bất kỳ phần nào, hãy cho tôi biết!

**Chương 2. Các sự kiện phổ biến**

### 1. ****Sự kiện chuột****

* **onClick**:
  + **Mô tả**: Kích hoạt khi người dùng nhấp chuột (thường là chuột trái) vào một phần tử.
  + **HTML truyền thống**:

html

<button onclick="handleClick()">Click</button><script> function handleClick() { console.log("Clicked!"); }</script>

* + **React**:

jsx

<button onClick={() => console.log("Clicked!")}>Click</button>

* + **Khác biệt**: HTML dùng chuỗi gọi hàm, React dùng tham chiếu hàm và camelCase (onClick).
* **onDoubleClick**:
  + **Mô tả**: Kích hoạt khi người dùng nhấp đúp chuột vào phần tử.
  + **HTML**:

html

<div ondblclick="handleDblClick()">Double Click</div>

* + **React**:

jsx

<div onDoubleClick={() => console.log("Double Clicked!")}>Double Click</div>

* + **Ghi chú**: Trong HTML, thuộc tính là ondblclick (viết thường), còn React dùng onDoubleClick.
* **onMouseEnter**:
  + **Mô tả**: Kích hoạt khi con trỏ chuột di chuyển vào phần tử.
  + **HTML**:

html

<div onmouseenter="handleEnter()">Hover me</div>

* + **React**:

jsx

<div onMouseEnter={() => console.log("Mouse entered!")}>Hover me</div>

* + **Khác biệt**: HTML dùng onmouseenter, React dùng onMouseEnter.
* **onMouseLeave**:
  + **Mô tả**: Kích hoạt khi con trỏ chuột rời khỏi phần tử.
  + **HTML**:

html

<div onmouseleave="handleLeave()">Leave me</div>

* + **React**:

jsx

<div onMouseLeave={() => console.log("Mouse left!")}>Leave me</div>

### 2. ****Sự kiện bàn phím****

* **onKeyDown**:
  + **Mô tả**: Kích hoạt khi một phím được nhấn xuống.
  + **HTML**:

html

<input onkeydown="handleKeyDown()" />

* + **React**:

jsx

<input onKeyDown={(e) => console.log(e.key)} />

* + **Ghi chú**: React cung cấp đối tượng sự kiện (e) để truy cập thông tin phím (như e.key).
* **onKeyPress**:
  + **Mô tả**: Kích hoạt khi một phím được nhấn và tạo ra một ký tự (lưu ý: đã bị deprecated trong nhiều trình duyệt).
  + **HTML**:

html

<input onkeypress="handleKeyPress()" />

* + **React**:

jsx

<input onKeyPress={(e) => console.log("Key pressed!")} />

* + **Khuyến nghị**: Trong React, nên dùng onKeyDown thay vì onKeyPress vì onKeyPress không còn được hỗ trợ tốt.
* **onKeyUp**:
  + **Mô tả**: Kích hoạt khi một phím được thả ra.
  + **HTML**:

html

<input onkeyup="handleKeyUp()" />

* + **React**:

jsx

<input onKeyUp={(e) => console.log("Key released!")} />

### 3. ****Sự kiện form****

* **onChange**:
  + **Mô tả**: Kích hoạt khi giá trị của phần tử form (như input, select) thay đổi.
  + **HTML**:

html

<input onchange="handleChange()" />

* + **React**:

jsx

<input onChange={(e) => console.log(e.target.value)} />

* + **Khác biệt**: Trong React, onChange kích hoạt ngay khi giá trị thay đổi, không cần đợi mất focus như HTML ở một số trường hợp.
* **onSubmit**:
  + **Mô tả**: Kích hoạt khi form được gửi (submit).
  + **HTML**:

html

<form onsubmit="handleSubmit()"> <button type="submit">Submit</button></form>

* + **React**:

jsx

<form onSubmit={(e) => { e.preventDefault(); console.log("Form submitted!"); }}> <button type="submit">Submit</button></form>

* + **Ghi chú**: Trong React, thường cần gọi e.preventDefault() để ngăn hành vi mặc định của form (như làm mới trang).
* **onFocus**:
  + **Mô tả**: Kích hoạt khi phần tử nhận được focus.
  + **HTML**:

html

<input onfocus="handleFocus()" />

* + **React**:

jsx

<input onFocus={() => console.log("Focused!")} />

* **onBlur**:
  + **Mô tả**: Kích hoạt khi phần tử mất focus.
  + **HTML**:

html

<input onblur="handleBlur()" />

* + **React**:

jsx

<input onBlur={() => console.log("Blurred!")} />

### 4. ****Sự kiện khác****

* **onLoad**:
  + **Mô tả**: Kích hoạt khi một phần tử (như hình ảnh hoặc trang) được tải xong.
  + **HTML**:

html

<img src="image.jpg" onload="handleLoad()" />

* + **React**:

jsx

<img src="image.jpg" onLoad={() => console.log("Image loaded!")} />

* **onError**:
  + **Mô tả**: Kích hoạt khi có lỗi xảy ra (ví dụ: hình ảnh không tải được).
  + **HTML**:

html

<img src="invalid.jpg" onerror="handleError()" />

* + **React**:

jsx

<img src="invalid.jpg" onError={() => console.log("Image load failed!")} />

* **onScroll**:
  + **Mô tả**: Kích hoạt khi người dùng cuộn nội dung trong phần tử có thanh cuộn.
  + **HTML**:

html

<div style="height: 100px; overflow: auto;" onscroll="handleScroll()">...</div>

* + **React**:

jsx

<div style={{ height: '100px', overflow: 'auto' }} onScroll={() => console.log("Scrolled!")}>...</div>

### So sánh tổng quan

| **Tiêu chí** | **HTML truyền thống** | **React** |
| --- | --- | --- |
| **Cú pháp** | Thuộc tính viết thường (onclick, onchange) | CamelCase (onClick, onChange) |
| **Giá trị sự kiện** | Chuỗi gọi hàm ("handleClick()") | Tham chiếu hàm (handleClick) |
| **Quản lý sự kiện** | Gắn trực tiếp vào DOM, dễ gây tốn tài nguyên | Synthetic Events, tối ưu qua Virtual DOM |
| **Hiệu suất** | Kém hơn trong ứng dụng lớn | Tốt hơn nhờ event delegation và Virtual DOM |
| **Khả năng tái sử dụng** | Khó tái sử dụng, logic tách biệt | Dễ dàng với components và hooks |
| **Đối tượng sự kiện** | Truy cập qua event trong hàm | Truy cập qua tham số (e) trong hàm |

### Lưu ý quan trọng

* **HTML truyền thống**: Phù hợp cho các trang web tĩnh hoặc ứng dụng nhỏ, nhưng khó mở rộng và bảo trì khi logic phức tạp.
* **React**: Sử dụng **Synthetic Events**, cung cấp tính nhất quán giữa các trình duyệt và hiệu suất tốt hơn. Tuy nhiên, cần học JSX và cách quản lý trạng thái.
* Trong React, luôn sử dụng tham chiếu hàm (onClick={handleClick}) thay vì gọi hàm trực tiếp (onClick={handleClick()}) để tránh gọi hàm ngay khi component render.
* Một số sự kiện như onKeyPress đã bị deprecated, nên ưu tiên onKeyDown trong các ứng dụng hiện đại.

**Chương 3. Form trong React**

### 1. ****Khái niệm về Form****

* **Định nghĩa**: Trong cả HTML và React, **form** là một phần tử giao diện dùng để thu thập dữ liệu từ người dùng. Các thành phần phổ biến của form bao gồm:
  + <input>: Nhập văn bản, số, email, mật khẩu, v.v.
  + <checkbox>: Hộp kiểm cho lựa chọn bật/tắt.
  + <select>: Danh sách thả xuống để chọn một hoặc nhiều giá trị.
  + <textarea>: Vùng nhập văn bản nhiều dòng.
  + Các thành phần khác như <radio>, <file>, v.v.
* **Mục đích**: Thu thập dữ liệu người dùng (ví dụ: thông tin đăng nhập, biểu mẫu đăng ký) và gửi dữ liệu đó (thường qua onSubmit) để xử lý (gửi lên server, lưu trữ, v.v.).
* **HTML truyền thống**:
  + Form sử dụng thuộc tính action để chỉ định URL nhận dữ liệu và method (thường là GET hoặc POST).

Ví dụ:

html

<form action="/submit" method="POST"> <input type="text" name="username" /> <button type="submit">Submit</button></form>

* + Dữ liệu được gửi trực tiếp qua HTTP request khi form được submit, không có trạng thái (state) quản lý giá trị đầu vào.
* **React**:
  + Form thường được quản lý thông qua **state** (trạng thái) trong React, sử dụng **Controlled Components** hoặc **Uncontrolled Components**.
  + React không sử dụng action hay method trực tiếp mà xử lý logic trong hàm JavaScript (thường qua onSubmit).

### 2. ****Controlled Component trong React****

* **Định nghĩa**:
  + Một **Controlled Component** là phần tử form (như <input>, <select>, <textarea>) có giá trị được điều khiển bởi **state** của React.
  + Giá trị của phần tử được liên kết với một biến state, và mọi thay đổi (do người dùng nhập) được xử lý thông qua sự kiện onChange để cập nhật state.
  + Điều này đảm bảo UI luôn đồng bộ với state, theo triết lý "single source of truth" (nguồn dữ liệu duy nhất) của React.
* **Cách hoạt động**:
  + **State** lưu trữ giá trị của phần tử form.
  + **onChange** được gắn vào phần tử để cập nhật state mỗi khi người dùng tương tác (nhập, chọn, v.v.).
  + Khi state thay đổi, React tự động re-render để hiển thị giá trị mới trên UI.
* **Ví dụ Controlled Component**:

jsx

import { useState } from 'react';function MyForm() { const [inputValue, setInputValue] = useState(''); const handleChange = (event) => { setInputValue(event.target.value); }; const handleSubmit = (event) => { event.preventDefault(); console.log('Submitted:', inputValue); }; return ( <form onSubmit={handleSubmit}> <input type="text" value={inputValue} onChange={handleChange} /> <button type="submit">Submit</button> </form> );}

* + **Giải thích**:
    - inputValue là state lưu giá trị của <input>.
    - Thuộc tính value={inputValue} liên kết giá trị của input với state.
    - onChange={handleChange} cập nhật inputValue mỗi khi người dùng nhập.
    - onSubmit={handleSubmit} xử lý khi form được gửi, với event.preventDefault() để ngăn làm mới trang.

### So sánh Form trong HTML truyền thống và Controlled Component trong React

| **Tiêu chí** | **HTML truyền thống** | **Controlled Component trong React** |
| --- | --- | --- |
| **Quản lý giá trị** | Giá trị nằm trong DOM, truy cập qua name hoặc id. | Giá trị được lưu trong state của React. |
| **Cập nhật UI** | Cập nhật thủ công qua DOM (element.value). | Tự động re-render khi state thay đổi. |
| **Sự kiện onChange** | Chỉ kích hoạt khi mất focus (ở một số trường hợp). | Kích hoạt ngay khi giá trị thay đổi. |
| **Sự kiện onSubmit** | Gửi dữ liệu qua action và method. | Xử lý logic trong hàm JavaScript, cần e.preventDefault(). |
| **Tính đồng bộ** | Không có cơ chế đồng bộ giá trị với logic. | Giá trị luôn đồng bộ với state. |
| **Tính linh hoạt** | Phù hợp cho form đơn giản, tĩnh. | Linh hoạt, dễ xử lý logic phức tạp (validation, điều kiện). |
| **Hiệu suất** | Nhanh cho ứng dụng nhỏ, nhưng khó quản lý khi phức tạp. | Tốt hơn cho ứng dụng lớn nhờ Virtual DOM. |

### Liên kết với các sự kiện Form

Các sự kiện form (onChange, onSubmit, onFocus, onBlur) đóng vai trò quan trọng trong **Controlled Component**:

* **onChange**:
  + Bắt buộc trong Controlled Component để cập nhật state mỗi khi người dùng thay đổi giá trị.
  + Ví dụ: <input onChange={(e) => setInputValue(e.target.value)} />.
* **onSubmit**:
  + Xử lý logic khi form được gửi, thường đi kèm e.preventDefault() để ngăn hành vi mặc định (làm mới trang).
  + Ví dụ: Gửi dữ liệu đến server hoặc hiển thị thông báo.
* **onFocus**:
  + Dùng để xử lý khi người dùng focus vào input, ví dụ: hiển thị gợi ý hoặc thay đổi giao diện.
  + Ví dụ: <input onFocus={() => console.log('Input focused!')} />.
* **onBlur**:
  + Dùng để xử lý khi input mất focus, thường để kiểm tra dữ liệu (validation).
  + Ví dụ: <input onBlur={() => console.log('Input blurred!')} />.

### Ưu điểm và nhược điểm của Controlled Component

* **Ưu điểm**:
  + Dễ dàng kiểm soát và xác thực dữ liệu (validation) trong state.
  + Dễ tích hợp với logic phức tạp (ví dụ: cập nhật giao diện theo giá trị nhập).
  + Đảm bảo UI và dữ liệu luôn đồng bộ.
* **Nhược điểm**:
  + Cần viết thêm code để quản lý state và sự kiện onChange, đặc biệt với form lớn có nhiều trường.
  + Có thể gây giảm hiệu suất trong form rất lớn nếu không tối ưu.

### Uncontrolled Component (so sánh ngắn gọn)

* Trong React, ngoài **Controlled Component**, còn có **Uncontrolled Component**, nơi giá trị của phần tử form không được quản lý bởi state mà được truy xuất qua **ref**.
* Ví dụ Uncontrolled Component:

jsx

import { useRef } from 'react';function UncontrolledForm() { const inputRef = useRef(); const handleSubmit = (event) => { event.preventDefault(); console.log('Submitted:', inputRef.current.value); }; return ( <form onSubmit={handleSubmit}> <input type="text" ref={inputRef} /> <button type="submit">Submit</button> </form> );}

* **Khác biệt chính**:
  + Controlled: Giá trị được lưu trong state, đồng bộ với UI.
  + Uncontrolled: Giá trị nằm trong DOM, truy xuất qua ref, ít code hơn nhưng khó kiểm soát logic phức tạp.

**Chương 4 Các Ví dụ bài làm**

**1. Giới thệu Event Handling trong ReactJS**

Trong ReactJS, event (sự kiện) là các hành động mà người dùng thực hiện trên giao diện, ví dụ: click chuột, nhập dữ liệu, submit form. React xử lý sự kiện tương tự như HTML, nhưng tuân theo quy tắc camelCase và thường gắn trực tiếp với component thay vì DOM thuần.

**2. Ví dụ 1: Function Component – Event không tham số**

Mã nguồn:

import React from 'react'  
  
export default function LxtEventComp() {  
  
 const lxt\_Func1 = () => {  
 alert("Xin chào")  
 }  
 const lxt\_Func2 = (param) => {  
 alert(param)  
 }  
 return (  
 <div>  
 <h1>Event - Component Function</h1>  
 <button onClick={lxt\_Func1}>Click Here</button>  
 <button onClick={() => lxt\_Func2('Lê Xuân Toàn')}>Click Here Param</button>  
 </div>  
 )  
}

→ Trong ví dụ trên, `lxt\_Func1` được gọi khi người dùng click vào nút đầu tiên. Nút thứ hai truyền tham số 'Lê Xuân Toàn' thông qua hàm arrow function.

**3. Ví dụ 2: Class Component – Event có và không có tham số**

Mã nguồn:

import React, { Component } from 'react';  
  
class LxtEventCompClass extends Component {  
  
 lxt\_Func1 = ()=>{  
 alert("Hàm không có tham số")  
 }  
  
 lxt\_Func2 = (param)=>{  
 alert("Hàm có tham số:\n"+ param)  
 }  
  
 render() {  
 return (  
 <div>  
 <h2>Event in class component</h2>  
 <button onClick={this.lxt\_Func1}>Click</button>  
 <button onClick={()=>this.lxt\_Func2("Lê Xuân Toàn")}>Click Param</button>  
 </div>  
 );  
 }  
}  
  
export default LxtEventCompClass;

→ Class component định nghĩa các hàm xử lý sự kiện là thuộc tính của class. Khi gọi hàm có tham số, cần dùng arrow function để truyền giá trị.

**4. Ví dụ 3: Form cơ bản – Controlled Component**

Mã nguồn:

import React, { Component } from 'react';  
  
class LxtEventForm1 extends Component {  
 constructor(props) {  
 super(props);  
 this.state = {  
 studentName: "Toan"  
 }  
 }  
  
 handleChange = (e) => {  
 this.setState({  
 studentName: e.target.value  
 })  
 }  
 handleUpdate = (e) => {  
 e.preventDefault();  
 this.setState({  
 studentName: 'Toan'  
 })  
 }  
  
 handleSubmit = (event) => {  
 event.preventDefault();  
 alert(this.state.studentName)  
 }  
 render() {  
 return (  
 <div className='alert alert-danger'>  
 <h2>Event Form</h2>  
 <form>  
 <label htmlFor='studentName'>Student Name:</label>  
 <input name='studentName' id='studentName'  
 value={this.state.studentName}  
 onChange={this.handleChange}  
 />  
 <button onClick={this.handleUpdate}>Change value</button>  
 <button onClick={this.handleSubmit}>Submit</button>  
 </form>  
 </div>  
 );  
 }  
}  
  
export default LxtEventForm1;

→ Form này quản lý giá trị input thông qua state (`studentName`). Khi người dùng nhập liệu, hàm `handleChange` sẽ cập nhật state. Submit form sẽ hiển thị giá trị trong alert.

**5. Ví dụ 4: Form tổng hợp nhiều trường dữ liệu**

Mã nguồn:

import React, { Component } from 'react';  
  
class LxtEventForm2 extends Component {  
 constructor(props) {  
 super(props);  
 this.state = {  
 studentName: '',  
 age: 0,  
 course: 'ReactJS'  
 }  
 }  
  
 handleChange = (event) => {  
 let name = event.target.name;  
 let val = event.target.value;  
  
 this.setState({  
 [name]: val  
 })  
 }  
  
 handleSubmit = (e) => {  
 e.preventDefault();  
 alert("Name:" + this.state.studentName + "\n Age:" + this.state.age)  
 }  
 render() {  
 return (  
 <div className='alert alert-info'>  
 <h1>Event Form - Tổng hợp</h1>  
 <form onSubmit={this.handleSubmit}>  
 <div>  
 <label htmlFor='studentName'>Student Name:</label>  
 <input name='studentName'  
 value={this.state.studentName}  
 onChange={this.handleChange}  
 />  
 </div>  
 <div>  
 <label htmlFor='age'>Student Age:</label>  
 <input name='age' value={this.state.age} onChange={this.handleChange} />  
 </div>  
 <div>  
 <label htmlFor='course'>Course:</label>  
 <select name='course' value={this.state.course} onChange={this.handleChange}>  
 <option value={'HTML5'}>HTML5</option>  
 <option value={'ReactJS'}>ReactJS</option>  
 <option value={'PHP'}>PHP</option>  
 </select>  
 </div>  
 <button type='submit'>Submit</button>  
 </form>  
 </div>  
 );  
 }  
}  
  
export default LxtEventForm2;

→ Form tổng hợp nhiều trường dữ liệu. Các input được liên kết trực tiếp với state qua thuộc tính `name`. Điều này cho phép cập nhật state động mà không cần viết nhiều hàm handle riêng cho từng input.

**6. Kết luận**

Qua các ví dụ trên, ta thấy ReactJS cung cấp cách xử lý sự kiện gọn gàng, linh hoạt hơn so với JavaScript thuần. Event kết hợp với Form giúp React trở thành công cụ mạnh mẽ trong việc phát triển ứng dụng web tương tác, nơi dữ liệu luôn được đồng bộ với giao diện thông qua state.

**Kết luận**

* **Form** trong HTML truyền thống và React đều dùng để thu thập dữ liệu, nhưng React cung cấp cách tiếp cận hiện đại hơn với **Controlled Component**, nơi giá trị được quản lý bởi state, đảm bảo đồng bộ và dễ xử lý logic phức tạp.
* **Controlled Component** sử dụng onChange để cập nhật state, kết hợp với các sự kiện như onSubmit, onFocus, onBlur để tạo trải nghiệm người dùng linh hoạt.
* So với HTML truyền thống, Controlled Component trong React mạnh mẽ hơn, nhưng đòi hỏi thêm code để quản lý state. Trong trường hợp form đơn giản, **Uncontrolled Component** có thể là lựa chọn gọn nhẹ hơn.

**Báo cáo**

**REACTJS FORM TRONG REACTJS**

**Chương 1. Giới thiệu tổng quan về ReactJS Lịch sử, sự phát triển.**  
 ReactJS là thư viện JavaScript mã nguồn mở do Facebook phát triển, ra mắt năm 2013, để xây dựng giao diện người dùng (UI) hiệu quả và nhanh chóng bằng cách sử dụng các thành phần UI có thể tái sử dụng. Lý do nên sử dụng ReactJS là nhờ hiệu suất cao nhờ DOM ảo, khả năng tái sử dụng mã nguồn qua các component, một cộng đồng lớn mạnh hỗ trợ tài liệu và giải pháp, cùng với việc phát triển ứng dụng di động đa nền tảng bằng React Native.

Lịch sử và Sự phát triển

**Nguồn gốc:**

React được Jordan Walke tạo ra tại Facebook vào năm 2011 để giải quyết sự phức tạp của các mô hình MVC truyền thống trên web.

Sử dụng nội bộ:

Ban đầu, React được dùng trong các sản phẩm của Facebook như Facebook News Feed (2011) và Instagram (2012).

Phát hành công khai:

Vào năm 2013, Facebook đã công bố ReactJS ra cộng đồng phát triển web dưới dạng mã nguồn mở.

**Phát triển thêm:**

Năm 2015, Facebook giới thiệu React Native, cho phép xây dựng ứng dụng di động cho iOS và Android chỉ với cùng một codebase.

Phát triển liên tục:

Facebook (nay là Meta) vẫn tiếp tục đầu tư và nâng cấp ReactJS để đáp ứng nhu cầu của cộng đồng phát triển hiện đại.

Tại sao nên sử dụng ReactJS?

Hiệu suất cao:

React sử dụng DOM ảo (Virtual DOM) để cập nhật UI, giúp tăng tốc độ render và hiệu suất ứng dụng so với DOM thực thông thường.

Tái sử dụng các thành phần (Component):

UI được chia thành các thành phần độc lập, có thể tái sử dụng, giúp giảm thời gian phát triển và dễ dàng bảo trì.

Cộng đồng lớn mạnh:

React có một cộng đồng nhà phát triển rất lớn, cung cấp nhiều tài nguyên, tài liệu và sự hỗ trợ, giúp giải quyết vấn đề và cập nhật kiến thức.

Phát triển ứng dụng di động:

Với React Native, bạn có thể xây dựng ứng dụng di động cho nhiều nền tảng (iOS và Android) chỉ với một codebase, tận dụng cùng kiến thức React đã học.

Cú pháp JSX:

Sử dụng JSX, một cú pháp cho phép nhúng code HTML trực tiếp vào JavaScript, giúp việc viết component trở nên trực quan và dễ hiểu hơn.

Thân thiện với SEO:

React được tối ưu hóa để cải thiện khả năng hiển thị trang web với các công cụ tìm kiếm, giúp tăng tỷ lệ tiếp cận.

**Chương 2. JSX Khái niệm, cú pháp.**

**1. Khái niệm**

JSX: là một cách viết cú pháp cho phép nhúng mã giống HTML vào trong mã JavaScript.

Nó giúp tạo ra các thành phần React một cách trực quan, dễ đọc và dễ hiểu hơn, thay vì sử dụng các hàm React.createElement phức tạp.

JSX không phải là HTML thực sự, mà là một cú pháp mở rộng JavaScript được các trình duyệt hiểu.

**2. Cú pháp**

Giống HTML:

JSX sử dụng các thẻ tương tự như HTML để tạo các phần tử React.

Biểu thức JavaScript:

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

Thuộc tính:

Tên thuộc tính trong JSX tuân theo quy tắc đặt tên của JavaScript, thường sử dụng cách viết camelCase. Ví dụ: className thay cho class, htmlFor thay cho for.

**3. Ví dụ minh họa**

Ví dụ sau cho thấy cách JSX sử dụng để hiển thị một lời chào:

Mã

// Cách viết JSX

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

// Cách React biên dịch JSX thành JavaScript

const Greeting = React.createElement("h1", null, "Xin chào thế giới!");

Trong ví dụ này, mã JSX <h1>Xin chào thế giới!</h1> được biên dịch thành một lời gọi hàm React.createElement để tạo ra một phần tử H1.

**Chương 3. Component Class component. Function component.**

**1. Class Component (Component Dạng Lớp)**

Định nghĩa: Đây là cách tạo component bằng cách định nghĩa một lớp (class) kế thừa từ một lớp cơ sở (ví dụ: Component trong React).

Đặc điểm:

Sử dụng cú pháp ES6 classes.

Có thể quản lý trạng thái (state) và vòng đời (lifecycle) thông qua this.state và các phương thức của lớp.

Thuận tiện khi cần sử dụng các tính năng nâng cao hoặc khi làm việc với các thư viện cũ.

Ví dụ minh họa (React):

Mã

import React from 'react';

class WelcomeMessage extends React.Component {

render() {

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

}

}

export default WelcomeMessage;

Trong ví dụ này, WelcomeMessage là một Class Component hiển thị lời chào.

**2. Function Component (Component Dạng Hàm)**

Định nghĩa: Đây là cách tạo component bằng cách định nghĩa một hàm JavaScript thông thường.

Đặc điểm:

Đơn giản, ngắn gọn và dễ đọc hơn.

Trong React, Function Component có thể quản lý trạng thái và vòng đời bằng cách sử dụng React Hooks (ví dụ: useState, useEffect).

Đây là cách được khuyến khích sử dụng trong React hiện đại.

Ví dụ minh họa (React):

Mã

import React from 'react';

function WelcomeMessage(props) {

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

}

export default WelcomeMessage;

Trong ví dụ này, WelcomeMessage là một Function Component cũng hiển thị lời chào, nhưng với cú pháp hàm đơn giản hơn.

**3. Ví dụ minh họa chung**

Các ví dụ minh họa thường sẽ đi sâu vào:

Truyền dữ liệu (Props):

Cách component nhận dữ liệu từ component cha.

Quản lý trạng thái (State):

Cách component tự quản lý dữ liệu nội bộ và cập nhật giao diện khi dữ liệu thay đổi, đặc biệt với Function Component sử dụng useState.

Sử dụng Hooks (với Function Component):

Minh họa cách dùng các Hooks như useState để quản lý state, useEffect để xử lý các tác vụ phụ thuộc vào vòng đời hoặc trạng thái.

Tạo các Component có thể tái sử dụng:

Xây dựng các UI linh hoạt, có thể dùng đi dùng lại ở nhiều nơi trong ứng dụng.

**Chương 4. Props và State Khái niệm, sự khác biệt.**

**1. Khái niệm và Sự khác biệt**

Props (Properties):

Khái niệm: Thuộc tính được truyền từ component cha đến component con, giống như các tham số trong một hàm.

Mục đích: Truyền dữ liệu từ cha xuống con, cho phép component cha cung cấp thông tin cho component con hiển thị hoặc sử dụng.

Đặc điểm:

Chỉ đọc: Component con không thể tự thay đổi giá trị của props.

Luồng dữ liệu: Chỉ truyền đi theo một chiều, từ trên xuống (cha -> con).

State:

Khái niệm: Dữ liệu nội bộ của một component, có thể thay đổi theo thời gian.

Mục đích: Quản lý dữ liệu động và tương tác của riêng component đó, chẳng hạn như giá trị người dùng nhập vào một ô input, trạng thái đóng/mở của một dropdown.

Đặc điểm:

Có thể thay đổi: Component có thể thay đổi giá trị của state thông qua các hàm như setState.

Cục bộ: Chỉ thuộc sở hữu và quản lý bởi component đó.

Bảng tóm tắt sự khác biệt:

| \*\*Đặc điểm\*\* | \*\*Props\*\* | \*\*State\*\* |

| ----------------- | ------------------------------------------ | ------------------------------------------ |

| \*\*Nguồn gốc\*\* | Truyền từ component cha | Do chính component quản lý |

| \*\*Tính chất\*\* | Chỉ đọc (Read-only) | Có thể thay đổi |

| \*\*Mục đích\*\* | Truyền cấu hình, dữ liệu cho component con | Quản lý dữ liệu nội bộ, động của component |

| \*\*Luồng dữ liệu\*\* | Từ cha xuống con | Nội bộ component |

**2. Truyền dữ liệu qua Props**

Component cha truyền dữ liệu cho component con thông qua các thuộc tính (properties) khi render component con.

Ví dụ:

Mã

// Component cha

function App() {

const userName = "Alice";

return <Welcome name={userName} />; // Truyền userName qua prop 'name'

}

// Component con

function Welcome(props) {

return <h1>Chào, {props.name}!</h1>; // Sử dụng prop 'name'

}

**3. Quản lý dữ liệu với State**

Trong các component function, chúng ta sử dụng hook useState để khai báo và quản lý state.

Ví dụ:

Mã

import React, { useState } from 'react';

function Counter() {

// Khai báo state 'count' với giá trị ban đầu là 0

// 'setCount' là hàm để cập nhật state

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

return (

<div>

<p>Bạn đã click {count} lần</p>

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

Click tôi

</button>

</div>

);

}

Trong ví dụ này, count là state, giá trị của nó thay đổi khi người dùng click vào nút, khiến component được render lại và hiển thị số lần click mới.

**Chương 5. Event Handling**

Xử lý Sự kiện (Event Handling) trong React mô tả cách xử lý tương tác người dùng bằng cách sử dụng các Event Handler trong cả Function Component và Class Component. Các sự kiện có thể có tham số bằng cách truyền chúng thông qua một hàm callback, thường được định nghĩa trong một hàm Arrow Function.

Xử lý Sự kiện trong Function Component

Trong Function Component, bạn sử dụng hook useState để quản lý trạng thái và gắn các trình xử lý sự kiện trực tiếp vào các phần tử JSX.

Định nghĩa Event Handler: Tạo một hàm để thực hiện logic khi sự kiện xảy ra.

Gắn Event Handler: Sử dụng các thuộc tính như onClick, onChange và gán hàm đã định nghĩa vào đó.

Ví dụ Code:

Mã

import React, { useState } from 'react';

function MyCounter() {

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

// Hàm xử lý sự kiện không có tham số

const increment = () => {

setCount(prevCount => prevCount + 1);

};

return (

<div>

<p>Số đếm: {count}</p>

{/\* Gắn Event Handler vào sự kiện onClick \*/}

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

</div>

);

}

export default MyCounter;

Xử lý Sự kiện có tham số trong Function Component

Để truyền tham số vào Event Handler, bạn sử dụng một Arrow Function trong thuộc tính sự kiện.

Ví dụ Code:

Mã

import React, { useState } from 'react';

function MyList() {

const [items, setItems] = useState(['Item 1', 'Item 2']);

// Hàm xử lý sự kiện có tham số là item để xóa

const deleteItem = (itemToDelete) => {

setItems(items.filter(item => item !== itemToDelete));

};

return (

<div>

<h2>Danh sách</h2>

<ul>

{items.map((item, index) => (

<li key={index}>

{item}

{/\* Sử dụng Arrow Function để truyền tham số \*/}

<button onClick={() => deleteItem(item)}>Xóa</button>

</li>

))}

</ul>

</div>

);

}

export default MyList;

Xử lý Sự kiện trong Class Component

Trong Class Component, bạn sử dụng các phương thức của lớp để định nghĩa Event Handler và sử dụng Arrow Function để đảm bảo this trỏ đến instance của lớp.

Định nghĩa Phương thức: Tạo một phương thức cho lớp để làm Event Handler.

Gắn Event Handler: Sử dụng các thuộc tính sự kiện như onClick và gán phương thức đó.

Ví dụ Code:

Mã

import React, { Component } from 'react';

class MyCounterClass extends Component {

constructor(props) {

super(props);

this.state = {

count: 0,

};

// Bind 'this' để đảm bảo phương thức hoạt động chính xác

// Hoặc dùng Arrow Function cho phương thức (đã được đề cập ở trên)

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

}

// Phương thức xử lý sự kiện không có tham số

increment = () => { // Sử dụng Arrow Function để tự động bind 'this'

this.setState(prevState => ({

count: prevState.count + 1,

}));

};

render() {

return (

<div>

<p>Số đếm: {this.state.count}</p>

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

</div>

);

}

}

export default MyCounterClass;

Xử lý Sự kiện có tham số trong Class Component

Tương tự như Function Component, bạn sử dụng một Arrow Function để gọi phương thức và truyền tham số.

**Chương 6. Render List và Key**

Lặp qua danh sách bằng .map()

Trong React, bạn sử dụng phương thức map() của mảng JavaScript để chuyển đổi một danh sách các mục thành một danh sách các phần tử UI. Phương thức này sẽ tạo ra một mảng các phần tử JSX dựa trên mỗi mục trong mảng gốc.

Tại sao cần key?

Khi lặp qua một danh sách, bạn cần cung cấp một thuộc tính key duy nhất cho mỗi phần tử được tạo ra. key giúp React:

Nhận dạng các phần tử:

React sử dụng key để theo dõi các mục trong danh sách khi chúng được thêm, xóa hoặc cập nhật.

Cải thiện hiệu suất:

Khi có key, React biết cách cập nhật danh sách một cách hiệu quả, tránh việc render lại toàn bộ danh sách không cần thiết, từ đó tăng hiệu suất của ứng dụng.

Quản lý trạng thái:

key giúp React duy trì trạng thái (state) của từng phần tử khi thứ tự hoặc số lượng phần tử thay đổi.

Lưu ý khi sử dụng key:

Duy nhất trong mảng:

key phải duy nhất giữa các mục anh em trong cùng một danh sách.

Không thay đổi:

key không nên thay đổi giữa các lần render, tránh sử dụng index của mảng làm key nếu danh sách có thể thay đổi thứ tự hoặc thêm/xóa phần tử.

Sử dụng ID dữ liệu:

Lý tưởng nhất là sử dụng ID duy nhất của dữ liệu (như item.id) làm key.

Ví dụ code

Mã

function DanhSachHangHoa({ hanghoa }) {

return (

<ul>

{hanghoa.map(item => (

<li key={item.id}> {/\* Sử dụng ID dữ liệu làm key \*/}

{item.ten}

</li>

))}

</ul>

);

}

// Dữ liệu mẫu:

const danhSachHangHoa = [

{ id: 1, ten: 'Táo' },

{ id: 2, ten: 'Chuối' },

{ id: 3, ten: 'Xoài' }

];

// Sử dụng component:

<DanhSachHangHoa hanghoa={danhSachHangHoa} />

Trong ví dụ này, mỗi mục trong danhSachHangHoa có một id duy nhất. Chúng ta sử dụng item.id làm key cho mỗi phần tử <li>, giúp React nhận diện và quản lý hiệu quả từng mặt hàng trong danh sách.

**Chương 7. ReactJS Form**

Trong React, components được kiểm soát (controlled components) giữ dữ liệu form trong state của component và sử dụng thuộc tính value và sự kiện onChange để điều khiển input, textarea, select. Ngược lại, components không kiểm soát (uncontrolled components) dựa vào các cơ chế DOM mặc định của trình duyệt, sử dụng defaultValue hoặc các ref để truy xuất giá trị khi cần thiết, thích hợp cho các form đơn giản. Để quản lý nhiều input, bạn có thể sử dụng một state object duy nhất với tên thuộc tính được tính toán (computed property names) để cập nhật các trường dữ liệu liên quan.

**1. Component được kiểm soát (Controlled Component)**

Định nghĩa:

Dữ liệu của phần tử form được quản lý bởi state của React.

Cách hoạt động:

Thiết lập value của phần tử form bằng một biến state.

Xử lý sự kiện onChange để cập nhật state khi người dùng nhập liệu.

Khi state thay đổi, value của phần tử form cũng được cập nhật, đảm bảo dữ liệu luôn đồng bộ.

Ưu điểm:

Dữ liệu form luôn nhất quán với state, giúp dự đoán và gỡ lỗi dễ dàng hơn.

Cho phép xác thực dữ liệu tùy chỉnh, vô hiệu hóa nút gửi và kiểm soát hoàn toàn UI.

Nhược điểm:

Có thể dẫn đến nhiều lần render và tốn kém hơn về hiệu suất với các form lớn.

**2. Component không kiểm soát (Uncontrolled Component)**

Định nghĩa:

Dữ liệu form không được lưu trữ trong state React, mà được quản lý trực tiếp bởi DOM.

Cách hoạt động:

Sử dụng thuộc tính defaultValue cho các input và defaultChecked cho checkbox/radio để đặt giá trị ban đầu.

Sử dụng ref để truy cập trực tiếp giá trị của phần tử input khi cần thiết (ví dụ, trong hàm xử lý sự kiện onSubmit).

Ưu điểm:

Dễ cài đặt và nhanh chóng, phù hợp với các form đơn giản không yêu cầu logic phức tạp về dữ liệu.

Trải nghiệm người dùng tốt hơn vì trang web không cần render lại mỗi khi giá trị input thay đổi.

Nhược điểm:

Khó truy cập và thao tác giá trị input trong thời gian thực, gây khó khăn cho xác thực tùy chỉnh.

**3. Quản lý nhiều Input với State (Controlled Components)**

Bạn có thể quản lý nhiều input trong một form bằng cách sử dụng một state object duy nhất và sử dụng thuộc tính name của input để cập nhật state tương ứng.

Mã

import React, { useState } from 'react';

function StudentForm() {

const [formData, setFormData] = useState({

name: '',

email: '',

studentId: '',

});

const handleChange = (event) => {

const { name, value } = event.target; // Lấy name và value của input

setFormData({

...formData,

[name]: value, // Cập nhật key state tương ứng với name của input

});

};

const handleSubmit = (event) => {

event.preventDefault();

console.log('Thông tin sinh viên:', formData);

// Xử lý gửi dữ liệu formData đến API

};

return (

<form onSubmit={handleSubmit}>

<div>

<label>Tên:</label>

<input

type="text"

name="name" // Thuộc tính name quan trọng để cập nhật state

value={formData.name}

onChange={handleChange}

/>

</div>

<div>

<label>Email:</label>

<input

type="email"

name="email"

value={formData.email}

onChange={handleChange}

/>

</div>

<div>

<label>Mã sinh viên:</label>

<input

type="text"

name="studentId"

value={formData.studentId}

onChange={handleChange}

/>

</div>

<button type="submit">Thêm sinh viên</button>

</form>

);

}

export default StudentForm;

**Kết luận**

Trong React, Form (biểu mẫu) đóng vai trò trung tâm trong việc cho phép người dùng nhập liệu và tương tác với ứng dụng. Các kiến thức cốt lõi bao gồm sử dụng component input và textarea để thu thập thông tin, quản lý trạng thái của các phần tử form thông qua useState, và xử lý sự kiện onChange để cập nhật dữ liệu khi người dùng nhập. Ngoài ra, việc sử dụng các thư viện như Formik hoặc React Hook Form giúp đơn giản hóa quá trình xử lý form phức tạp, bao gồm xác thực dữ liệu và xử lý lỗi.

Kiến thức cốt lõi về Form trong React:

Các phần tử form:

Sử dụng các phần tử HTML chuẩn như <input>, <textarea>, <select>, và <button> để xây dựng giao diện người dùng nhập liệu.

Controlled Components:

Đây là cách phổ biến để quản lý các phần tử form trong React.

Trạng thái (state): Sử dụng hook useState để lưu trữ giá trị hiện tại của các phần tử form như input hay textarea.

Sự kiện onChange: Gắn một hàm xử lý sự kiện onChange vào các phần tử form để cập nhật state mỗi khi người dùng thay đổi giá trị.

Thuộc tính value: Gán giá trị từ state cho thuộc tính value của phần tử form để đảm bảo giá trị trong input luôn đồng bộ với state.

Sử dụng onSubmit cho form:

Gắn một hàm xử lý sự kiện onSubmit vào thẻ <form> để xử lý dữ liệu sau khi người dùng gửi form. Hàm này thường chứa logic xử lý dữ liệu, gửi lên server, hoặc thực hiện xác thực.

Xác thực dữ liệu (Validation):

Thực hiện xác thực dữ liệu người dùng nhập để đảm bảo tính chính xác và hợp lệ trước khi gửi đi.

Xử lý lỗi:

Hiển thị thông báo lỗi cho người dùng khi họ nhập sai hoặc bỏ sót thông tin.

Vai trò của Form trong ứng dụng React:

Form đóng vai trò quan trọng trong việc tạo ra các ứng dụng có tính tương tác cao, cho phép người dùng thực hiện các hành động và nhập thông tin như:

Thu thập thông tin người dùng: Đăng ký, đăng nhập, điền thông tin cá nhân, đặt hàng.

Tương tác và điều khiển dữ liệu: Cho phép người dùng sửa đổi, thêm mới hoặc xóa dữ liệu trong ứng dụng.

Xây dựng giao diện người dùng động: Các form có thể thay đổi nội dung và chức năng dựa trên logic và dữ liệu trong ứng dụng.