**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**

**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.