# ***1. ReactJS là gì?***

ReactJS là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook vào năm 2013, nhằm mục đích xây dựng giao diện người dùng (UI) cho các ứng dụng web. Thư viện này cho phép các lập trình viên tạo ra các thành phần UI độc lập và có thể tái sử dụng, giúp tăng tính linh hoạt và khả năng mở rộng của ứng dụng

# ***2. Giải thích kiến trúc MVC***

Các thành phần của mô hình MVC gồm có model, view, controller. Mỗi thành phần sẽ đóng một vai trò riêng.

Model: Model đại diện cho dữ liệu và logic nghiệp vụ của ứng dụng. Đây là nơi lưu trữ và quản lý trạng thái hiện tại của dữ liệu, cung cấp các phương thức để truy cập và cập nhật dữ liệu đó. Khi có thay đổi trong dữ liệu, Model sẽ thông báo cho View để cập nhật giao diện người dùng.

View: View là phần giao diện người dùng, chịu trách nhiệm hiển thị dữ liệu từ Model một cách trực quan. View không tương tác trực tiếp với Model mà chỉ nhận dữ liệu từ Controller để hiển thị cho người dùng. Các thành phần giao diện như nút bấm, biểu đồ và bảng đều thuộc về View.

Controller: Controller đóng vai trò là cầu nối giữa Model và View. Nó xử lý các yêu cầu từ người dùng, cập nhật Model khi cần thiết và thông báo cho View để cập nhật giao diện tương ứng. Khi người dùng thực hiện hành động (như nhấp chuột hay nhập dữ liệu), Controller sẽ nhận diện hành động đó và thực hiện các thao tác cần thiết

# ***3. Giải thích các thành phần cơ bản của React***

**Components (Thành phần)**

Components là các đơn vị cơ bản trong React, giúp cấu trúc giao diện người dùng. Mỗi component có thể chứa mã HTML, CSS và JavaScript riêng biệt, cho phép tái sử dụng và dễ dàng quản lý. Có hai loại component chính:

● Class Components: Được định nghĩa bằng cách sử dụng class và có thể chứa các phương thức vòng đời.

● Function Components: Được định nghĩa bằng cách sử dụng hàm, thường đơn giản hơn và dễ đọc hơn.

**JSX (JavaScript XML)**

JSX là cú pháp mở rộng của JavaScript cho phép viết HTML trong JavaScript một cách dễ dàng. Nó giúp kết hợp mã HTML với logic JavaScript, làm cho việc phát triển giao diện trở nên trực quan hơn.

**Props (Thuộc tính)**

Props là cách để truyền dữ liệu từ component cha đến component con. Chúng là bất biến, nghĩa là một khi đã được thiết lập, giá trị của props không thể thay đổi trong component con. Props giúp các component tương tác với nhau và tạo ra giao diện động.

**State (Trạng thái)**

State đại diện cho trạng thái nội bộ của một component. Khi state thay đổi, React sẽ tự động cập nhật giao diện người dùng để phản ánh sự thay đổi này. State có thể được thay đổi thông qua các sự kiện hoặc hành động của người dùng, giúp tạo ra trải nghiệm tương tác.

**Virtual DOM (DOM Ảo)**

Virtual DOM là một bản sao ảo của DOM thực tế. React sử dụng Virtual DOM để tối ưu hóa hiệu suất bằng cách so sánh sự khác biệt giữa Virtual DOM và DOM thực tế trước khi thực hiện cập nhật. Điều này giúp giảm thiểu số lần cập nhật cần thiết trên DOM thực tế, từ đó cải thiện tốc độ và hiệu suất của ứng dụng.

# ***4. Giải thích props và state trong React với sự khác biệt***

Props và state là hai khái niệm quan trọng giúp quản lý dữ liệu trong các component. Props là các thuộc tính được truyền từ component cha xuống component con. Chúng giống như các tham số của một hàm, cho phép giao tiếp giữa các component. Props là bất biến (immutable), không thể thay đổi giá trị.

State là một đối tượng lưu trữ dữ liệu nội bộ của một component. Khác với props, state có thể thay đổi trong suốt vòng đời của component. Khi state thay đổi, React sẽ tự động render lại component đó để phản ánh những thay đổi mới

Điểm khác nhau giữa props và state trong React:

|  |  |  |
| --- | --- | --- |
| Đặc điểm | Props | State |
| Quản lý | Được quản lý bởi component cha | Được quản lý bởi chính component chứa state |
| Thay đổi | Không thể thay đổi (immutable) | Có thể thay đổi thông qua setState |
| Khởi tạo | Được truyền từ component cha | Được khởi tạo bên trong component |
| Mục đích sử dụng | Dùng để truyền dữ liệu giữa các component | Dùng để theo dõi và quản lý dữ liệu động |
| Tính khả dụng | Có thể truy cập từ nhiều component | Chỉ có thể truy cập từ chính component đó |

# ***5. DOM ảo trong React là gì?***

DOM ảo trong React là một biểu diễn JavaScript của cấu trúc DOM thực tế. Khi trạng thái của ứng dụng thay đổi, React tạo ra một Virtual DOM mới để phản ánh những thay đổi này từ đó có thể giảm thiểu số lần cập nhật trực tiếp vào DOM thực tế, từ đó cải thiện tốc độ và hiệu suất của ứng dụng

# ***6. JSX là gì?***

JSX, viết tắt của JavaScript XML, là một cú pháp mở rộng cho ngôn ngữ JavaScript, cho phép lập trình viên viết mã HTML trong JavaScript một cách dễ dàng. JSX được Facebook phát triển chủ yếu để sử dụng với thư viện React, giúp tạo ra giao diện người dùng một cách trực quan và hiệu quả.

# ***7. Thành phần là gì và các loại thành phần trong React?***

Component trong React là một khối xây dựng cơ bản của ứng dụng, cho phép lập trình viên chia nhỏ giao diện người dùng (UI) thành các phần riêng biệt, độc lập và có thể tái sử dụng. Mỗi component có thể nhận dữ liệu thông qua các thuộc tính (props) và trả về các phần tử React (thường là JSX) để hiển thị trên trình duyệt.

Trong React, có hai loại component chính:

* Function Component: Là các hàm JavaScript nhận props và trả về một phần tử React. Chúng không có trạng thái (state) và không sử dụng các phương thức vòng đời (lifecycle methods).
* Class Component: Là các lớp (class) JavaScript có khả năng lưu trữ trạng thái (state) và sử dụng các phương thức vòng đời.

# ***8. Trình duyệt đọc JSX như thế nào?***

Khi bạn viết mã JSX, nó sẽ được chuyển đổi thành các lệnh gọi hàm JavaScript thông qua một trình biên dịch như Babel. Ví dụ, đoạn mã JSX sau:

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

sẽ được biên dịch thành:

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

# ***9. Giải thích các bước để tạo ứng dụng React và in ra Hello World?***

Để tạo một ứng dụng React đơn giản và in ra "Hello World", bạn có thể làm theo các bước dưới đây:

**Bước 1: Cài đặt Node.js và npm**

Trước tiên, bạn cần cài đặt Node.js và npm (trình quản lý gói cho Node.js). Bạn có thể tải Node.js từ trang chính thức tại [nodejs.org](https://nodejs.org/). Khi cài đặt Node.js, npm sẽ được cài đặt tự động cùng với nó.

**Bước 2: Cài đặt Create React App**

Sau khi cài đặt Node.js và npm, mở Terminal hoặc Command Prompt và chạy lệnh sau để cài đặt Create React App:

npm install -g create-react-app

Lệnh này sẽ cài đặt Create React App trên máy tính của bạn, cho phép bạn tạo các ứng dụng React một cách dễ dàng.

**Bước 3: Tạo dự án React mới**

Tiếp theo, bạn có thể tạo một dự án React mới bằng cách chạy lệnh sau trong Terminal hoặc Command Prompt:

npx create-react-app my-app

Thay "my-app" bằng tên mà bạn muốn cho dự án của mình. Sau khi lệnh hoàn tất, bạn sẽ có một thư mục mới chứa cấu trúc dự án React chuẩn.

**Bước 4: Chạy ứng dụng**

Đi vào thư mục dự án mới tạo và khởi động ứng dụng bằng các lệnh sau:

cd my-app

npm start

Lệnh này sẽ mở trình duyệt mặc định của bạn và chạy ứng dụng tại địa chỉ: http://localhost:3000.

**Bước 5: Sửa file App.js để hiển thị "Hello World"**

Mở file src/App.js trong trình soạn thảo mã nguồn của bạn. Thay đổi nội dung của file này như sau để hiển thị "Hello World":

import React from 'react';

function App() {

 return (

    <div>

      <h1>Hello, World!</h1>

    </div>

  );

}

export default App;

Lưu file lại. Trình duyệt sẽ tự động tải lại và hiển thị "Hello, World!" trên trang web.

# ***10. Làm thế nào để tạo sự kiện trong React?***

Để tạo sự kiện trong React, trước tiên, bạn cần định nghĩa một hàm sẽ xử lý sự kiện. Hàm này sẽ chứa logic mà bạn muốn thực hiện khi sự kiện xảy ra. Ví dụ, nếu bạn muốn hiển thị một thông báo khi người dùng nhấp vào nút, bạn có thể định nghĩa hàm như sau:

function handleClick() {

   alert("Bạn đã click chuột vào tôi!");

Sau khi định nghĩa hàm, bạn cần thêm nó vào phần tử JSX mà bạn muốn gán sự kiện. Trong trường hợp này, bạn sẽ thêm thuộc tính onClick vào thẻ <button>:

<button chuột vào tôi</button>

# ***11. Giải thích việc tạo List trong React?***

Phương pháp phổ biến nhất để tạo danh sách trong React là sử dụng phương thức map() của mảng. Phương thức này cho phép bạn lặp qua các phần tử trong mảng và chuyển đổi chúng thành các phần tử JSX.

# ***12. Khóa trong React là gì?***

Khóa (keys) là một thuộc tính đặc biệt được sử dụng để định danh duy nhất cho các phần tử trong một mảng các thành phần (components). Chúng giúp React xác định những phần tử nào đã thay đổi, được thêm mới, hoặc bị xóa khi có sự thay đổi trong trạng thái (state) hoặc dữ liệu (data) của ứng dụng.

# ***13. Làm thế nào để viết bình luận trong React?***

● Để viết bình luận một dòng trong JSX, bạn có thể sử dụng cú pháp sau:

{/\* Đây là bình luận một dòng \*/}

● Nếu bạn muốn viết bình luận nhiều dòng, bạn cũng sử dụng cùng cú pháp nhưng mở rộng ra nhiều dòng:

{/\*

Đây là bình luận

nhiều dòng

\*/}

# ***14. Giải thích sự khác biệt giữa React và Angular?***

Dưới đây là bảng so sánh giữa React và Angular, giúp bạn dễ dàng nhận thấy sự khác biệt giữa hai công nghệ này:

|  |  |  |
| --- | --- | --- |
| Tiêu chí | React | Angular |
| Kiến trúc | Thư viện JavaScript cho giao diện người dùng | Framework đầy đủ với nhiều tính năng tích hợp |
| Ràng buộc dữ liệu | Một chiều (one-way data binding) | Hai chiều (two-way data binding) |
| DOM | Sử dụng DOM ảo (virtual DOM) | Sử dụng DOM thực (real DOM) |
| Hiệu suất | Nhanh hơn trong các ứng dụng nhỏ nhờ DOM ảo | Tối ưu hóa qua AOT và phát hiện thay đổi |
| Khả năng mở rộng và bảo trì | Tái sử dụng mã và tổ chức thành phần dễ dàng | Cung cấp AngularCLI để tạo và duy trì ứng dụng |
| Quản lý trạng thái | Thường cần kết hợp với thư viện như Redux hoặc MobX | Có sẵn các công cụ quản lý trạng thái |
| Học tập và cộng đồng | Dễ tiếp cận cho người mới, cộng đồng lớn | Có đường cong học tập cao hơn, cộng đồng mạnh mẽ |

# ***15. Giải thích cách sử dụng phương thức render trong React?***

Render là quá trình mà React sử dụng để hiển thị nội dung trên giao diện người dùng (UI) dựa trên sự thay đổi của thuộc tính (props) và trạng thái (state).

Để kích hoạt lần render đầu tiên, bạn sẽ sử dụng phương thức createRoot từ thư viện react-dom/client, sau đó gọi phương thức render với component.

Khi một component được render, React sẽ thực hiện các bước sau:

● Gọi component: React sẽ gọi hàm render của component để lấy output.

● So sánh với Virtual DOM: React so sánh output mới với Virtual DOM để xác định những thay đổi cần thiết.

● Cập nhật DOM thực: Cuối cùng, React sẽ cập nhật DOM thực dựa trên những thay đổi đã tính toán.

# ***16. Trạng thái trong React là gì?***

Trong React, trạng thái (state) là một khái niệm quan trọng dùng để quản lý và theo dõi dữ liệu thay đổi theo thời gian trong ứng dụng. Trạng thái được sử dụng để biểu diễn các thông tin mà có thể thay đổi, như là đầu vào của người dùng, kết quả từ API, hoặc bất kỳ dữ liệu nào cần được cập nhật và hiển thị trên giao diện người dùng.

# ***17. Giải thích về props trong React?***

Props là các tham số đầu vào của component trong React. Chúng cho phép bạn truyền dữ liệu và cấu hình cho các component con, tương tự như các thuộc tính (attributes) trong HTML. Một điểm quan trọng là props là bất biến (immutable), nghĩa là once passed to a component, they cannot be modified by that component.

# ***18. Thành phần bậc cao trong React là gì?***

Thành phần bậc cao trong React, hay còn gọi là Higher-Order Component (HOC), là một khái niệm quan trọng giúp tái sử dụng logic giữa các thành phần. Đây là những hàm JavaScript nhận vào một component và trả về một component mới, cho phép thêm các chức năng bổ sung mà không cần thay đổi cấu trúc của component gốc.

# ***19. Giải thích sự khác biệt giữa thành phần chức năng và thành phần lớp trong React?***

Trong React, có hai loại thành phần chính: Thành phần Chức năng (Function Components) và Thành phần Lớp (Class Components). Mỗi loại có những đặc điểm riêng và được sử dụng trong những tình huống khác nhau.

**Thành phần Lớp (Class Components)**

● Cú pháp: Được viết bằng cú pháp của ES6 class và yêu cầu kế thừa từ React.Component.

● Quản lý trạng thái: Có khả năng quản lý trạng thái thông qua this.state và các phương thức lifecycle như componentDidMount, componentDidUpdate, và componentWillUnmount.

● Phương thức render: Phải sử dụng phương thức render() để trả về phần tử JSX.

● Tình huống sử dụng: Thường được sử dụng khi cần quản lý trạng thái phức tạp hoặc khi cần sử dụng các lifecycle methods. Chúng thích hợp cho các thành phần có nhiều logic xử lý.

**Thành phần Chức năng (Function Components)**

● Cú pháp: Là hàm JavaScript thông thường, không yêu cầu kế thừa từ lớp nào.

● Quản lý trạng thái: Ban đầu không có khả năng quản lý trạng thái, nhưng từ phiên bản React 16.8 trở đi, có thể sử dụng Hooks (như useState và useEffect) để quản lý trạng thái và các hiệu ứng bên ngoài.

● Trả về JSX: Trả về JSX ngay từ bên trong hàm mà không cần phương thức render.

● Tình huống sử dụng: Thích hợp cho các thành phần đơn giản, không cần quản lý trạng thái phức tạp. Chúng giúp viết mã ngắn gọn và dễ đọc hơn

# ***20. Giải thích về ràng buộc dữ liệu một chiều trong React?***

Ràng buộc dữ liệu là quá trình kết nối giữa các thành phần (components) trong ứng dụng. Trong React, ràng buộc dữ liệu một chiều có nghĩa là dữ liệu chỉ được truyền từ component cha xuống component con thông qua props. Điều này có nghĩa là:

● Dữ liệu di chuyển theo một hướng: Dữ liệu được quản lý và luân chuyển từ component cha xuống component con. Component con không thể trực tiếp thay đổi dữ liệu của component cha mà chỉ có thể gửi thông tin trở lại thông qua sự kiện (events) như nhấn nút hoặc nhập liệu.

● Sự kiện thay đổi dữ liệu: Khi cần thay đổi dữ liệu, component con sẽ kích hoạt một hành động (event), và component cha sẽ xử lý việc cập nhật dữ liệu đó. Quá trình này giúp duy trì tính nhất quán và dễ dàng kiểm soát luồng dữ liệu trong ứng dụng.

# ***21. Render có điều kiện trong React là gì?***

Render có điều kiện trong React cho phép hiển thị các thành phần khác nhau dựa trên trạng thái của ứng dụng. Tương tự như JavaScript, bạn có thể sử dụng câu lệnh if, toán tử điều kiện (ternary operator), hoặc toán tử logic && để quyết định xem một thành phần có nên được render hay không.

# ***22. React Router là gì?***

React Router là một thư viện quan trọng trong phát triển ứng dụng React, cho phép quản lý điều hướng giữa các trang và thành phần mà không cần phải tải lại toàn bộ trang.

# ***23. Giải thích các thành phần của react-router***

Các thành phần chính của React Router gồm có:

● BrowserRouter: Là thành phần bao bọc ứng dụng, sử dụng HTML5 History API để đồng bộ hóa URL với trạng thái ứng dụng.

● Route: Định nghĩa một đường dẫn cụ thể và kết nối nó với một component. Khi URL khớp với đường dẫn này, component tương ứng sẽ được hiển thị.

● Link: Tạo liên kết đến một đường dẫn cụ thể trong ứng dụng, giúp điều hướng giữa các trang mà không cần tải lại.

● Switch: Chọn một Route duy nhất từ nhiều Route có thể khớp với đường dẫn hiện tại, đảm bảo chỉ một component được hiển thị tại một thời điểm.

# ***24. Giải thích các phương pháp vòng đời của các thành phần***

Vòng đời sản phẩm mô tả quá trình phát triển và tồn tại của một sản phẩm từ khi ra mắt cho đến khi bị rút lui khỏi thị trường. PLC thường được chia thành bốn giai đoạn chính:

● Phát triển và giới thiệu: Đây là giai đoạn đầu tiên, nơi sản phẩm được phát triển từ ý tưởng cho đến khi ra mắt trên thị trường. Trong giai đoạn này, doanh nghiệp cần đầu tư vào nghiên cứu và phát triển, cũng như xây dựng chiến lược tiếp thị ban đầu.

● Tăng trưởng: Sau khi ra mắt, nếu sản phẩm thành công, doanh số sẽ tăng trưởng nhanh chóng. Doanh nghiệp cần tối ưu hóa các chiến dịch tiếp thị để thu hút khách hàng và mở rộng thị trường.

● Trưởng thành: Giai đoạn này xảy ra khi doanh số đạt đỉnh và bắt đầu ổn định. Doanh nghiệp cần duy trì sự cạnh tranh thông qua cải tiến sản phẩm hoặc điều chỉnh giá cả.

● Suy thoái: Cuối cùng, doanh số bắt đầu giảm do nhiều yếu tố như sự thay thế bởi sản phẩm mới hoặc thay đổi trong nhu cầu của người tiêu dùng. Doanh nghiệp cần đưa ra quyết định về việc tiếp tục duy trì sản phẩm hay rút lui khỏi thị trường.

# ***25. Giải thích các phương pháp được sử dụng trong giai đoạn lắp ráp các thành phần***

Mỗi component trong React có một vòng đời rõ ràng, bao gồm ba giai đoạn chính: Mounting, Updating, và Unmounting. Các phương thức lifecycle sẽ được gọi trong từng giai đoạn này, cho phép lập trình viên can thiệp vào quá trình khởi tạo và cập nhật component.

**Giai đoạn Mounting**

Trong giai đoạn này, component được gắn vào DOM. Có bốn phương thức chính được gọi theo thứ tự:

● constructor(): Đây là phương thức đầu tiên được gọi khi một component được khởi tạo. Tại đây, bạn có thể thiết lập state ban đầu và bind các phương thức nếu cần thiết.

● static getDerivedStateFromProps(): Phương thức này được gọi ngay sau constructor và trước khi render. Nó cho phép bạn cập nhật state dựa trên sự thay đổi của props.

● render(): Phương thức này là bắt buộc và sẽ luôn được gọi để trả về JSX mà component sẽ hiển thị.

● componentDidMount(): Phương thức này được gọi sau khi component đã được render lần đầu tiên và gắn vào DOM, thường dùng để thực hiện các tác vụ như gọi API hoặc thiết lập subscription

**Giai đoạn Updating**

Khi state hoặc props của một component thay đổi, React sẽ tiến hành cập nhật component. Các phương thức lifecycle trong giai đoạn này bao gồm:

● static getDerivedStateFromProps(): Tương tự như trong giai đoạn mounting, nhưng được gọi trước mỗi lần render lại.

● shouldComponentUpdate(): Phương thức này cho phép bạn kiểm soát xem component có cần render lại hay không. Nếu trả về false, quá trình render sẽ bị bỏ qua.

● render(): Được gọi để cập nhật giao diện người dùng với dữ liệu mới.

● getSnapshotBeforeUpdate(): Cho phép bạn lấy thông tin từ DOM trước khi thay đổi xảy ra.

● componentDidUpdate(): Được gọi ngay sau khi component đã cập nhật và render lại.

**Giai đoạn Unmounting**

Khi một component không còn cần thiết nữa và bị xóa khỏi DOM, phương thức:

● componentWillUnmount(): Được gọi để thực hiện các tác vụ dọn dẹp như hủy bỏ subscription hoặc timer

# ***26. Hàm this.setState trong React là gì?***

Hàm this.setState trong React là một phương thức quan trọng được sử dụng để cập nhật trạng thái (state) của một component. Hàm this.setState cho phép bạn thay đổi giá trị của state trong một component. Khi bạn gọi hàm này, React sẽ tự động cập nhật state mới và kích hoạt quá trình re-render component để hiển thị dữ liệu mới nhất.

# ***27. ref trong React có tác dụng gì?***

Refs (references) là một đối tượng được sử dụng để giữ tham chiếu đến một biến hoặc một component, cho phép giá trị của nó không thay đổi giữa các lần render. Chúng giúp truy cập trực tiếp vào các nút DOM hoặc các phần tử React được tạo ra trong phương thức render.

# ***28. Hook trong React là gì?***

React Hooks là một tập hợp các hàm cho phép bạn "kết nối" với trạng thái và các tính năng vòng đời của React từ các thành phần chức năng. React Hooks giúp đơn giản hóa việc quản lý trạng thái và tương tác với vòng đời của component, làm cho mã dễ đọc và dễ bảo trì hơn.

# ***29. Giải thích hook useState trong React?***

useState là một trong những hook cơ bản nhất trong React, được giới thiệu từ phiên bản 16.8. Nó cho phép bạn quản lý trạng thái (state) trong các Functional Components, điều mà trước đây chỉ có thể thực hiện trong Class Components. Hook này giúp đơn giản hóa việc quản lý trạng thái và cải thiện tính tương tác của ứng dụng.

# ***30. Giải thích hook useEffect trong React?***

useEffect là một hook trong React được sử dụng để quản lý các side effects trong các Functional Components. Side effects bao gồm những tác vụ như gọi API, thay đổi DOM, hoặc tương tác với các thư viện bên ngoài. Hook này giúp bạn thực hiện các tác vụ không thuộc về việc render giao diện người dùng và cho phép bạn kiểm soát thời điểm mà các tác vụ này được thực hiện.

# ***31. React Fragments là gì?***

React Fragments là một tính năng được giới thiệu trong React 16, cho phép bạn nhóm nhiều phần tử con (children) mà không cần tạo ra các node DOM dư thừa, giúp cải thiện hiệu suất và làm cho mã nguồn trở nên sạch sẽ hơn.

# ***32. React Developer Tools là gì?***

React Developer Tools là một tiện ích mở rộng (extension) cho trình duyệt, giúp lập trình viên dễ dàng kiểm tra và gỡ lỗi các ứng dụng React. Tiện ích này cung cấp cái nhìn tổng quan về cấu trúc cây component, cũng như các state và props hiện tại của từng component trong ứng dụng.

# ***33. Làm thế nào để sử dụng style trong ReactJS?***

Để sử dụng style trong ReactJS, bạn có thể áp dụng bốn phương pháp chính sau đây:

**Inline Styling**

Inline styling cho phép bạn áp dụng trực tiếp các kiểu vào các phần tử thông qua thuộc tính style. Các kiểu được định nghĩa dưới dạng đối tượng JavaScript, với các thuộc tính sử dụng cú pháp camelCase thay vì hyphenated như trong CSS thông thường.

**CSS Stylesheet**

Sử dụng CSS Stylesheets là phương pháp truyền thống để định kiểu cho ứng dụng. Bạn tạo một file CSS riêng biệt và nhập nó vào component của mình.

Cách thực hiện:

● Tạo file CSS (ví dụ: App.css):

body {

 background-color: #008080;

  color: yellow;

  padding: 40px;

  font-family: Arial;

  text-align: center;

}

● Nhập vào component: import './App.css';

Phương pháp này rất hữu ích khi styles không cần thay đổi theo trạng thái.

**CSS Modules**

CSS Modules cho phép bạn định nghĩa styles cho từng component một cách riêng biệt, giúp tránh xung đột tên class. Cách này giúp quản lý styles dễ dàng hơn và giữ cho từng component có styles riêng.

**Styled Components**

Styled-components là một thư viện cho phép bạn viết styles ở cấp độ component, sử dụng cú pháp ES6 như sau npm install styled-components --save. Phương pháp này rất linh hoạt và giúp dễ dàng tạo ra các component có style riêng

# ***34. Giải thích các thành phần được định kiểu trong React?***

Trong React, các thành phần (components) là những khối xây dựng cơ bản của giao diện người dùng. Chúng cho phép lập trình viên chia nhỏ giao diện thành các phần độc lập, có thể tái sử dụng và dễ quản lý. Dưới đây là giải thích về các thành phần chính trong React.

# ***35. Prop Drilling là gì?***

Prop Drilling (còn được gọi là "threading") là tình huống trong đó dữ liệu được truyền từ một thành phần cha qua nhiều cấp độ các thành phần con cho đến khi nó đến được thành phần cần sử dụng dữ liệu đó. Quá trình này có thể trở nên phức tạp và khó quản lý khi ứng dụng mở rộng, dẫn đến việc mã nguồn trở nên khó đọc và bảo trì hơn.

# ***36. Custom Hook trong React là gì?***

Custom Hook là một hàm JavaScript mà tên của nó bắt đầu bằng chữ "use" (theo quy tắc đặt tên của React), cho phép bạn sử dụng các hook khác bên trong nó. Mục tiêu chính của Custom Hook là để chia sẻ logic giữa nhiều component mà không cần phải truyền props qua nhiều cấp độ.

# ***37. Làm thế nào để tối ưu hóa mã React?***

Để tối ưu hóa mã React, có một số kỹ thuật quan trọng mà lập trình viên nên áp dụng:

● Sử dụng React.memo: React.memo giúp ngăn chặn việc re-render không cần thiết cho các component không thay đổi props. Bằng cách bọc component trong React.memo, bạn có thể cải thiện hiệu suất ứng dụng của mình.

● Tối ưu hóa render với useMemo và useCallback: Sử dụng useMemo để ghi nhớ giá trị tính toán tốn kém và useCallback để giữ nguyên hàm callback giữa các lần render. Điều này giúp giảm thiểu việc tạo lại hàm và giảm số lần render cho các component con.

● Lazy Loading và Code Splitting: Lazy loading cho phép tải các phần của ứng dụng khi cần thiết, trong khi code splitting chia nhỏ mã thành các bundle nhỏ hơn, giúp giảm thời gian tải trang ban đầu.

● Sử dụng Transitions: Với React 18, bạn có thể sử dụng startTransition để phân biệt giữa các cập nhật khẩn cấp và không khẩn cấp, giúp React xử lý tốt hơn.

# ***38. Sự khác nhau giữa useref và createRef trong React là gì?***

Sự khác nhau giữa useRef và createRef trong React chủ yếu nằm ở cách chúng hoạt động và ngữ cảnh sử dụng:

**Ngữ cảnh sử dụng**

● useRef: Là một hook được sử dụng trong các component chức năng (functional components) và giữ nguyên giá trị giữa các lần render.

● createRef: Được sử dụng chủ yếu trong các component lớp (class components) và tạo ra một ref mới mỗi khi component được render.

**Hành vi giữa các lần render**

● useRef: Giữ nguyên cùng một ref object qua các lần render, cho phép bạn lưu trữ giá trị mà không gây ra re-render cho component.

● createRef: Tạo ra một ref mới mỗi khi component render, do đó không giữ lại giá trị giữa các lần render.

**Cách khởi tạo**

● useRef(initialValue): Có thể nhận một giá trị khởi tạo và sẽ trả về cùng một ref object trong suốt vòng đời của component.

● createRef(): Không nhận giá trị khởi tạo; giá trị current sẽ được đặt là null ban đầu.

# ***39. React-redux là gì?***

React-Redux là một thư viện kết nối giữa React và Redux, giúp quản lý trạng thái (state) trong các ứng dụng React một cách hiệu quả. React-Redux cung cấp các API để dễ dàng tích hợp Redux vào ứng dụng React, cho phép các component truy cập vào trạng thái toàn cục được lưu trữ trong Redux store mà không cần phải truyền props qua nhiều cấp độ component.

# ***40. Lợi ích của việc sử dụng react-redux là gì?***

Lợi ích của việc sử dụng React-Redux trong phát triển ứng dụng:

● Quản lý trạng thái phức tạp: Redux giúp quản lý hiệu quả các trạng thái phức tạp, cho phép các component truy cập và cập nhật từ một nguồn duy nhất, giảm thiểu việc truyền props.

● Dự đoán trạng thái: Nguyên tắc "state is read-only" và các reducer đảm bảo việc dự đoán trạng thái dễ dàng, hỗ trợ các tác vụ như undo và redo.

● Gỡ lỗi dễ dàng: Redux cung cấp tính năng theo dõi lịch sử action và state, giúp lập trình viên xác định nguyên nhân lỗi. Công cụ Redux DevTools hỗ trợ gỡ lỗi theo thời gian.

● Tính nhất quán: Nguyên tắc "single source of truth" duy trì tính nhất quán của dữ liệu qua việc gửi action xử lý bởi reducer, giảm thiểu lỗi.

● Hiệu suất tối ưu: Redux chỉ cập nhật các component cần thiết khi có thay đổi trạng thái, cải thiện hiệu suất, đặc biệt trong các ứng dụng lớn.

● Khả năng mở rộng: Redux hỗ trợ tích hợp middleware để mở rộng chức năng, cho phép xử lý bất đồng bộ, ghi log, và định dạng dữ liệu, giúp cải tiến tính tùy chỉnh và mở rộng.

# ***41. Giải thích các thành phần cốt lõi của react-redux?***

**Store (Ngăn chứa)**

Store là nơi lưu trữ toàn bộ trạng thái của ứng dụng. Nó đại diện cho cây trạng thái duy nhất trong Redux.

Chức năng của store là:

● Lưu trữ trạng thái hiện tại của ứng dụng.

● Cung cấp các phương thức như getState() để lấy trạng thái, dispatch(action) để gửi hành động và subscribe(listener) để đăng ký lắng nghe thay đổi trạng thái.

**State (Trạng thái)**

State là một đối tượng chứa toàn bộ dữ liệu của ứng dụng, được quản lý tại một nơi duy nhất trong store. State trong Redux là không thay đổi (immutable), có nghĩa là bạn không thể thay đổi trực tiếp mà phải thông qua các hành động.

**Actions (Hành động)**

Actions là các đối tượng JavaScript mô tả sự kiện hoặc ý định thay đổi state. Mỗi action cần có thuộc tính type để xác định loại hành động.

Một action có thể được định nghĩa như sau:

{ type: 'ADD\_TODO', payload: 'Buy groceries' }

Actions giúp xác định cách mà state sẽ thay đổi khi một sự kiện xảy ra.

**Reducers**

Reducers là các hàm thuần túy nhận vào state hiện tại và action, sau đó trả về state mới. Chúng xác định cách mà state sẽ thay đổi dựa trên action nhận được. Reducers không thay đổi trạng thái gốc mà trả về một trạng thái mới dựa trên trạng thái hiện tại và hành động.

**Provider**

Provider là một thành phần từ thư viện react-redux giúp kết nối Redux store với toàn bộ ứng dụng React. Khi sử dụng Provider, bạn bao bọc ứng dụng với nó để mọi component có thể truy cập vào store.

**Selectors**

Selectors là các hàm dùng để chọn một phần của state từ store và truyền vào component. Chúng giúp tối ưu hóa việc truy cập và quản lý dữ liệu trong Redux

# ***42. Làm thế nào chúng ta có thể kết hợp nhiều reducer trong React?***

Để kết hợp nhiều reducer trong React, bạn có thể sử dụng hai phương pháp chính: Redux và React Hooks với useReducer và useContext. Dưới đây là 2 cách để bạn có thể kết hợp nhiều reducer trong React:

**Sử dụng Redux**

Redux cho phép bạn kết hợp nhiều reducer thành một reducer lớn thông qua hàm combineReducers. Điều này giúp quản lý trạng thái phức tạp hơn trong ứng dụng. Dưới đây là cách thực hiện:

● Cài đặt Redux: Đầu tiên, bạn cần cài đặt Redux và react-redux.

● Tạo các Reducer: Định nghĩa các reducer riêng lẻ cho từng phần của trạng thái ứng dụng.

const counterReducer = (state = { count: 0 }, action) => {

   switch (action.type) {

        case 'INCREMENT':

            return { ...state, count: state.count + 1 };

        case 'DECREMENT':

            return { ...state, count: state.count - 1 };

        default:

            return state;

    }

};

const userReducer = (state = { name: '' }, action) => {

    // Logic cho userReducer

};

● Kết hợp Reducers: Sử dụng combineReducers để kết hợp các reducer.

import { combineReducers } from 'redux';

const rootReducer = combineReducers({

   counter: counterReducer,

    user: userReducer,

});

● Tạo Store: Tạo store với rootReducer đã kết hợp.

import { createStore } from 'redux';

const store = createStore(rootReducer);

**Sử dụng React Hooks với useReducer và useContext**

Nếu bạn không muốn sử dụng Redux, bạn có thể kết hợp useReducer và useContext để quản lý trạng thái trong React:

* Tạo Context: Đầu tiên, tạo context để chia sẻ trạng thái giữa các component.

import React, { createContext, useReducer } from 'react';

const StateContext = createContext();

* Định nghĩa Reducer: Tạo một reducer để xử lý các hành động.

const initialState = { count: 0 };

const reducer = (state, action) => {

   switch (action.type) {

        case 'INCREMENT':

            return { ...state, count: state.count + 1 };

        case 'DECREMENT':

            return { ...state, count: state.count - 1 };

        default:

            return state;

    }

};

● Cung cấp Context: Sử dụng một provider để cung cấp trạng thái cho các component con.

const StateProvider = ({ children }) => {

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

    return (

        <StateContext.Provider value={{ state, dispatch }}>

            {children}

        </StateContext.Provider>

    );

};

● Sử dụng Context trong Component: Các component con có thể sử dụng context để truy cập trạng thái và dispatch hành động.

import React, { useContext } from 'react';

const CounterComponent = () => {

   const { state, dispatch } = useContext(StateContext);

    return (

        <div>

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

            <button => dispatch({ type: 'INCREMENT' })}>Increment</button>

            <button => dispatch({ type: 'DECREMENT' })}>Decrement</button>

        </div>

    );

};

# ***43. Context API là gì?***

Context API là một tính năng tích hợp trong React, cho phép chia sẻ dữ liệu giữa các component mà không cần phải truyền props qua từng cấp (hay còn gọi là "prop drilling") giúp đơn giản hóa việc quản lý trạng thái toàn cục trong ứng dụng và cải thiện khả năng tái sử dụng mã nguồn.

# ***44. Giải thích về provider và consumer trong ContextAPI?***

**Provider**

Provider là một component của React, được sử dụng để cung cấp giá trị cho các component con trong cây component. Nó nhận một prop là value, chứa dữ liệu mà bạn muốn chia sẻ.

Cách sử dụng: Khi bạn tạo một đối tượng context bằng React.createContext(), bạn sẽ có thể sử dụng Provider để bao bọc các component cần truy cập vào giá trị đó.

**Consumer**

Consumer là một component cho phép bạn truy cập giá trị được cung cấp bởi Provider. Nó sử dụng một hàm render prop để nhận giá trị từ context.

Cách sử dụng: Bạn có thể sử dụng Consumer để lấy giá trị từ context như sau:

const ChildComponent = () => (

 <ThemeContext.Consumer>

    {theme => <div>Our theme is: {theme}</div>}

  </ThemeContext.Consumer>

Trong ví dụ này, ChildComponent sẽ hiển thị "Our theme is: dark" nếu nó nằm trong cây con của ThemeContext.Provider.

# ***45. Giải thích về CORS trong React?***

CORS, hay Cross-Origin Resource Sharing, là một cơ chế bảo mật trong trình duyệt web, cho phép hoặc chặn các yêu cầu từ một nguồn (origin) khác với nguồn của tài nguyên mà ứng dụng đang chạy. Đây là một vấn đề phổ biến khi phát triển ứng dụng React, đặc biệt khi frontend và backend được triển khai trên các miền khác nhau.

# ***46. Axios là gì và cách sử dụng nó trong React?***

Axios là một thư viện HTTP client phổ biến trong JavaScript, được xây dựng dựa trên đối tượng Promise. Nó cho phép các nhà phát triển dễ dàng gửi các yêu cầu HTTP bất đồng bộ tới các REST API, hỗ trợ thực hiện các thao tác CRUD (Create, Read, Update, Delete) một cách linh hoạt trong các ứng dụng frontend như React, Angular và Vue.js.

Để sử dụng Axios trong ứng dụng React, bạn cần thực hiện các bước sau:

● Bước 1: Cài đặt Axios

Mở terminal và chạy lệnh sau để cài đặt Axios:

npm install axios

hoặc nếu bạn sử dụng Yarn:

yarn add axios

● Bước 2: Sử dụng Axios để gọi API

Dưới đây là ví dụ về cách sử dụng Axios để thực hiện yêu cầu GET trong một component React:

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

import axios from 'axios';

function App() {

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

    useEffect(() => {

        const fetchData = async () => {

            try {

                const response = await axios.get('https://jsonplaceholder.typicode.com/users');

                setData(response.data);

            } catch (error) {

                console.error("Error fetching data:", error);

            }

        };

        fetchData();

    }, []);

    return (

        <div>

            <h1>List of Users</h1>

            <ul>

                {data.map(user => (

                    <li key={user.id}>{user.name}</li>

                ))}

            </ul>

        </div>

    );

}

export default App;

**Trong đó:**

● useEffect: Hook này được sử dụng để gọi API ngay sau khi component được mount. Trong trường hợp này, nó sẽ gọi hàm fetchData để lấy dữ liệu từ API.

● axios.get(): Phương thức này được sử dụng để gửi yêu cầu GET tới API. Dữ liệu phản hồi sẽ được lưu vào state thông qua setData.

● Error Handling: Sử dụng try-catch để xử lý lỗi khi gọi API.

● Bước 3: Thực hiện yêu cầu POST

Để gửi dữ liệu đến API, bạn có thể sử dụng phương thức POST như sau:

const postData = async (newUser) => {

   try {

        const response = await axios.post('https://jsonplaceholder.typicode.com/users', newUser);

        setData(prevData => [...prevData, response.data]);

    } catch (error) {

        console.error("Error posting data:", error);

    }

};

Trong đoạn mã trên, newUser là đối tượng chứa dữ liệu bạn muốn gửi đến API.

Sử dụng Axios trong React giúp đơn giản hóa việc giao tiếp với các dịch vụ web và quản lý dữ liệu hiệu quả hơn.

# ***47. React-Material UI là gì?***

Material-UI, hiện nay được biết đến với tên gọi MUI, là một thư viện mã nguồn mở cung cấp các React components thiết kế dựa trên Material Design của Google. Thư viện này giúp lập trình viên dễ dàng xây dựng giao diện người dùng đẹp mắt và nhất quán cho các ứng dụng web.

# ***48. Kiến trúc Flux trong Redux là gì?***

Kiến trúc Flux là một mô hình quản lý trạng thái được phát triển bởi Facebook, chủ yếu để hỗ trợ cho việc xây dựng ứng dụng với React. Đây không phải là một thư viện hay framework mà là một cách tổ chức mã nguồn theo một luồng dữ liệu một chiều, giúp dễ dàng quản lý và theo dõi trạng thái của ứng dụng.

# ***49. Giải thích về kết xuất đồng thời trong React***

Kết xuất đồng thời (Concurrent Rendering) là một tính năng quan trọng được giới thiệu trong React 18, nhằm cải thiện hiệu suất và trải nghiệm người dùng trong các ứng dụng React. Tính năng này cho phép React thực hiện nhiều bản cập nhật giao diện người dùng (UI) cùng một lúc mà không làm chậm lại luồng chính của ứng dụng.

# ***50. Cách tạo một bộ đếm đơn giản bằng React***

Để tạo một bộ đếm đơn giản bằng React, bạn có thể sử dụng hook useState để quản lý trạng thái của bộ đếm. Dưới đây là hướng dẫn chi tiết với mã nguồn mẫu.

**Bước 1: Tạo thành phần Counter**

Trước tiên, bạn cần tạo một thành phần Counter để hiển thị và điều khiển bộ đếm. Mã nguồn dưới đây minh họa cách thực hiện điều này:

import React, { useState } from 'react';

const Counter = () => {

   const [count, setCount] = useState(0); // Khởi tạo state cho bộ đếm

    return (

        <div>

            <h1>{count}</h1> {/\* Hiển thị giá trị bộ đếm \*/}

            <button => setCount(count + 1)}>Tăng</button> {/\* Nút tăng \*/}

            <button => setCount(count - 1)}>Giảm</button> {/\* Nút giảm \*/}

            <button => setCount(0)}>Đặt lại</button> {/\* Nút đặt lại \*/}

        </div>

    );

};

export default Counter;

**Bước 2: Sử dụng thành phần Counter trong ứng dụng chính**

Sau khi đã tạo thành phần Counter, bạn cần sử dụng nó trong ứng dụng chính của bạn. Dưới đây là mã nguồn cho ứng dụng chính:

import React from 'react';

import ReactDOM from 'react-dom';

import Counter from './Counter'; // Nhập thành phần Counter

const App = () => {

   return (

        <div>

            <h1>Bộ Đếm Đơn Giản</h1>

            <Counter /> {/\* Sử dụng thành phần Counter \*/}

        </div>

    );

};

ReactDOM.render(<App />, document.getElementById('root'));

**Trong đó:**

● useState: Hook useState được sử dụng để tạo một biến trạng thái count, khởi tạo với giá trị 0. Hàm setCount được sử dụng để cập nhật giá trị của count.

● Các nút điều khiển: Mỗi nút có một hàm xử lý sự kiện onClick để tăng, giảm hoặc đặt lại giá trị của bộ đếm.

● Hiển thị giá trị: Giá trị hiện tại của bộ đếm được hiển thị trong thẻ H1.

Hy vọng rằng qua bài viết này, bạn đã có thêm nhiều thông tin hữu ích về các câu hỏi phỏng vấn ReactJS. Việc nắm vững những kiến thức từ cơ bản đến nâng cao sẽ giúp bạn không chỉ vượt qua vòng phỏng vấn mà còn phát triển kỹ năng lập trình của mình.