Link tham khảo

* <https://dev.to/dancrtis/learn-to-usecontext-with-hooks-in-3-minutes-4c4g>
* <https://reactrouter.com/en/main/routers/create-hash-router>
* npm install --save-exact [react-error-overlay@6.0.9](mailto:react-error-overlay@6.0.9)
* https://react-bootstrap.netlify.app/docs/components/alerts/

1.Khái niệm ReactJS?

* ReactJS là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook. Nó được sử dụng để xây dựng các ứng dụng web đơn trang (Single Page Applications - SPA) và các giao diện người dùng (User Interfaces - UI) tương tác.
* ReactJS cho phép phát triển ứng dụng web dễ dàng hơn bằng cách sử dụng các thành phần (components) tái sử dụng và quản lý trạng thái (state) của các thành phần đó.
* ReactJS được xây dựng dựa trên cơ sở của ngôn ngữ lập trình JavaScript, cho phép các nhà phát triển sử dụng nó với các thư viện khác như Redux, React Router, hay Axios để xây dựng các ứng dụng phức tạp.
* Ngoài ra, ReactJS cũng cho phép tương tác với các thư viện khác bằng cách sử dụng API.

2. ES6 - Từ khóa let và const?

Trong JavaScript ES6, let và const là từ khóa được sử dụng để khai báo biến.

**Let** được sử dụng để khai báo một biến trong một block scope, có nghĩa là biến chỉ có thể truy cập được trong phạm vi của khối lệnh mà nó được khai báo.

* function foo() {

let x = 10;

if (true) {

let x = 20;

console.log(x); // 20}

console.log(x); // 10}

**Const** cũng được sử dụng để khai báo biến, nhưng nó chỉ có thể được gán một lần và không thể được gán lại giá trị khác. Biến được khai báo bằng từ khóa const cũng có phạm vi của nó là block scope.

* function bar() {

const y = 10;

y = 20; // lỗi

const z; // lỗi}

3. ES6 - khái niệm arrow function?

Arrow function (còn được gọi là "fat arrow function" hoặc "lambda function" trong một số ngôn ngữ lập trình khác) là một tính năng mới được giới thiệu trong ECMAScript 6 (ES6) của JavaScript.

Arrow function cung cấp một cú pháp rút gọn cho khai báo hàm bằng từ khóa **function**, giúp cho việc viết mã nguồn trở nên ngắn gọn và dễ đọc hơn. Ví dụ, hãy xem xét một hàm thông thường như sau:

* function add(a, b) {

return a + b;}

Cách viết bằng arrow function sẽ giúp loại bỏ từ khóa function, giảm thiểu độ dài code và giúp cho code trở nên dễ đọc hơn:

* const add = (a, b) => a + b;

Arrow function có thể có hoặc không có tham số, cũng như có thể trả về giá trị hoặc không trả về giá trị. Nếu hàm chỉ có một tham số, bạn có thể bỏ qua cặp dấu ngoặc đơn. Ví dụ:

* const double = num => num \* 2;

Nếu hàm không có tham số, bạn cần sử dụng cặp dấu ngoặc đơn trống để biểu thị danh sách tham số:

* const sayHello = () => console.log("Hello!");

Arrow function có thể được sử dụng trong nhiều tình huống khác nhau, bao gồm việc truyền hàm vào các phương thức như Array.map(), Array.filter(), Array.reduce(), v.v. để xử lý các mảng.

4. ES6 - khái niệm Import và export modules?

**Import**: Là cách để import các module được export từ các file khác để sử dụng trong file hiện tại. Để import một module, ta sử dụng từ khóa "import" cùng với tên module cần import và đường dẫn đến file chứa module đó. Ví dụ:

* import { functionName } from './module.js';

**Export**: Là cách để export các module từ file hiện tại để có thể import và sử dụng trong các file khác. Để export một module, ta sử dụng từ khóa "export" trước khai báo tên module cần export. Ví dụ:

* export function functionName() { // code here }

Ngoài ra, ta còn có thể sử dụng default export bằng cách export một module với từ khóa "export default". Khi import, ta không cần phải sử dụng dấu ngoặc nhọn để chỉ định tên module, mà có thể sử dụng bất kỳ tên nào để đặt tên cho module đó. Ví dụ:

// export default module

* export default function() { // code here }

// import default module

* import anyName from './module.js';

5. ES6 - Tìm hiểu về Class

* Trong ES6 (ECMAScript 2015) và các phiên bản mới hơn của JavaScript, bạn có thể sử dụng từ khóa "class" để định nghĩa một lớp.Là 1 bảng thiết kế của đối tượng, là sự trừu tượng hoá đối tượng
* Một lớp có thể chứa các phương thức (method) và thuộc tính (property) để định nghĩa hành vi và trạng thái của các đối tượng của lớp.
* Khi bạn tạo một đối tượng từ một lớp, đối tượng đó được gọi là một thể hiện (instance) của lớp đó.
* Class cũng hỗ trợ tính kế thừa, cho phép bạn định nghĩa một lớp con (subclass) dựa trên một lớp cha (superclass). Để kế thừa từ một lớp cha, bạn sử dụng từ khóa "extends" và nó gọi phương thức constructor của lớp cha bằng từ khóa "super".

6. ES6 - Tìm hiểu khai báo Class, Methods và Properties?

7. ES6 - Toán tử Spread và Rest?

Spread:

1. Toán tử spread được biểu thị bằng dấu ba chấm (**...**) và được sử dụng để trải một mảng thành các phần tử riêng lẻ.

* const arr1 = [1, 2, 3];

const arr2 = [4, 5, 6];

const arr3 = [...arr1, ...arr2]; // [1, 2, 3, 4, 5, 6]

1. Toán tử Spread còn được sử dụng để sao chép một mảng hoặc đối tượng.
2. Toán tử spread cũng có thể được sử dụng để truyền các phần tử của một mảng vào một hàm mà yêu cầu các đối số được truyền dưới dạng các tham số riêng lẻ.

* function sum(a, b, c) {

return a + b + c;}

const arr = [1, 2, 3];

const result = sum(...arr); // 6

**Rest**

1. Toán tử rest cũng được biểu thị bằng dấu ba chấm (...) và được sử dụng để gom các tham số thành một mảng.

* function myFunction(...args) {

console.log(args);}

myFunction(1, 2, 3, 4); // [1, 2, 3, 4]

1. Toán tử rest cũng có thể được sử dụng để định nghĩa một hàm với số lượng đối số không xác định trước đó.

* function sum(...args) {

return args.reduce((acc, val) => acc + val);}

const result = sum(1, 2, 3, 4); // 10

8.  ES6 - Kỹ thuật desctructuring?

* Cho phép bạn rút ngắn mã và truy cập các giá trị trong các đối tượng hoặc mảng. Nó cho phép bạn tách các phần tử của một mảng hoặc thuộc tính của một đối tượng ra thành các biến riêng lẻ.
* const person = { firstName: 'John', lastName: 'Doe', age: 25};

const { firstName, lastName, age } = person;

console.log(firstName); // output: John

console.log(lastName); // output: Doe

console.log(age); // output: 25

* const numbers = [1, 2, 3, 4, 5];

const [first, second, ...rest] = numbers;

console.log(first); // output: 1

console.log(second); // output: 2

console.log(rest); // output: [3, 4, 5]

9. ES6 - Khái niệm Primitive types và reference types?

* **Primitive types** là kiểu dữ liệu không thể thay đổi (immutable), nghĩa là giá trị của chúng không thể thay đổi sau khi được tạo ra. Có sáu kiểu nguyên thủy trong JavaScript: string (chuỗi), number (số), boolean (luận lý), null (giá trị null), undefined (giá trị không xác định), symbol (ký hiệu)
* **Reference types** là kiểu dữ liệu có thể thay đổi (mutable), nghĩa là giá trị của chúng có thể thay đổi sau khi được tạo ra. Reference types là các đối tượng được tạo ra bằng từ khóa new, chẳng hạn như mảng, hàm và đối tượng.
* Một khác biệt quan trọng giữa primitive types và reference types là kiểu nguyên thủy được truyền theo giá trị (passed by value), trong khi kiểu tham chiếu được truyền theo tham chiếu (passed by reference). Khi một giá trị nguyên thủy được gán cho một biến, một bản sao của giá trị được tạo ra và lưu trữ trong biến. Tuy nhiên, khi một kiểu tham chiếu được gán cho một biến, chỉ một tham chiếu đến đối tượng được lưu trữ trong biến, không phải một bản sao của đối tượng.

10. ES6 - Khái niệm Array functions?

1. Array functions là một tập hợp các hàm được cung cấp bởi ES6 để làm việc với các mảng trong JavaScript. Các hàm này giúp bạn thực hiện các tác vụ phổ biến trên mảng như lọc, sắp xếp, ánh xạ, giảm, phù hợp và nhiều hơn nữa một cách dễ dàng và rõ ràng hơn.
2. Một số Array functions phổ biến trong ES6 bao gồm:

* map() - tạo một mảng mới bằng cách thực hiện một hành động trên mỗi phần tử của mảng ban đầu.
* filter() - lọc các phần tử của mảng ban đầu dựa trên một điều kiện nhất định và trả về một mảng mới chỉ chứa các phần tử thoả mãn điều kiện đó.
* reduce() - tính tổng các phần tử trong mảng ban đầu hoặc thực hiện một hành động phức tạp hơn trên các phần tử của mảng.
* forEach() - lặp lại mỗi phần tử của mảng và thực hiện một hành động trên từng phần tử.
* find() - trả về phần tử đầu tiên của mảng thoả mãn điều kiện được chỉ định.
* some() - kiểm tra xem có phần tử nào trong mảng thoả mãn điều kiện được chỉ định hay không.
* every() - kiểm tra xem tất cả các phần tử trong mảng có thoả mãn điều kiện được chỉ định hay không.

20. Toolchain?

1. **Toolchain** là chuỗi công cụ tích hợp trong quá trình phát triển giúp đem đến trải nghiệm tốt nhất cho người dùng và nhà phát triển như:

* Mở rộng quy mô cho nhiều file và component.
* Sử dụng thư viện của bên thứ ba từ npm.
* Phát hiện sớm các lỗi thường gặp.
* Chỉnh sửa trực tiếp CSS và JS đang được phát triển.
* Tối ưu hóa đầu ra cho production.

1. **Một số toolchain phổ biến**

* [npm](https://www.npmjs.com/) tên đầy đủ là Node Package Manager, một trình quản lý các thư viện trong môi trường Node.js. Các thư viện được đóng gói dưới dạng package và được lập trình viên đóng góp, chia sẻ trên npm registry (cơ sở dữ liệu của npm)
* Cú Pháp: npm instal (tên thư viện muốn cài)
* npx (Node Package Execute), là một công cụ để chạy package mà không cần phải cài đặt. Hãy xem một ví dụ dưới đây để hình dung được lợi ích mà npx đem lại cho chúng ta:
* Cú pháp: npx create-react-app app-name

21.Khái niệm Client Side Rendering và Server Side Rendering?

* **Server Side Rendering:** Bạn request một trang web, server xử lý nội dung thành HTML, return lại cho browser hiển thị lại lên màn hình.=> nhược điểm:
* Trang web phải xử lý lại hoàn toàn và load lại từ đầu nếu chỉ có một thay đổi nhỏ trong nội dung. (Ví dụ tiêu đề thay đổi …)
* Việc xử lý nội dung HTML khiến hao tốn tài nguyên server, gây chậm trễ khi xử lý các request khác.
* Lượng request lên server rất nhiều, do mọi tác vụ đều phải xử lý lại trên server và render lại HTML
* TTFB (Time To First Byte) cao do cần phải xử lý trên server.
* **Client-side rendering (CSR)** đề cập đến quá trình render các trang web hoàn toàn trong trình duyệt của khách hàng bằng cách sử dụng JavaScript. Việc sinh nội dung … đều được thực hiện ngay trên client, server chỉ có nhiệm vụ trả mã nguồn JS và dữ liệu thô cho client=> khắc phục được 1 số nhược điểm của SSR. Tuy nhiên vẫn có 1 số nhược điểm:
* Lượng dữ liệu lần đầu load về khá nặng
* Nội dung hiển thị ra lần đầu chậm hơn so với SSR, tuy nhiên những lần sau thì quá nhanh do việc xử lý ngay trên client.
* SEO bị ảnh hưởng, do nội dung web được sinh trên client, khiến crawler của của Search engine không tiếp xúc được nội dung.
* Gần đây mới có Google giải quyết được vấn đề này, tuy nhiên dù Google nắm phần lớn thị phần tìm kiếm thì tại những quốc gia khác nhau, sẽ có ảnh hưởng nhất định.

22. ReactDOM là gì? Tại sao phải sử dụng?

* **ReactDOM** là một thư viện JavaScript được sử dụng trong React để hiển thị các thành phần React trên trang web. Nó cho phép bạn tương tác với DOM (Document Object Model) của trang web, cập nhật nội dung của trang dựa trên sự thay đổi của các thành phần React và quản lý các sự kiện như bấm nút hoặc điều hướng.
* **ReactDOM** là một phần quan trọng của React vì nó cho phép các thành phần React được hiển thị trên trang web. Nó cũng giúp React tối ưu hóa quá trình cập nhật DOM bằng cách chỉ cập nhật các phần của trang web mà thực sự cần được cập nhật thay vì cập nhật toàn bộ trang. Điều này làm cho ứng dụng React nhanh hơn và tối ưu hơn so với các ứng dụng truyền thống được xây dựng bằng JavaScript.

23. React.createElement() là gì?

1. **React.createElement()** là một phương thức trong thư viện React, được sử dụng để tạo ra các thành phần React. Nó được sử dụng để tạo ra một cây ảnh hưởng (virtual DOM) của các thành phần và phần tử DOM.
2. **React.createElement()** được sử dụng để tạo ra các thành phần React bằng cách truyền vào các tham số như tên của thành phần, thuộc tính, và nội dung của thành phần.
3. Cú pháp của React.createElement() như sau: React.createElement(type, [props], [...children]) [props], [...children])

* type (bắt buộc): Tên của thành phần hoặc thẻ HTML cần tạo.
* props (tùy chọn): Đối tượng chứa các thuộc tính và giá trị được truyền cho thành phần.
* children (tùy chọn): Các thành phần con được truyền vào thành phần hiện tại.
* VD: const element = React.createElement(

"div", // Tên của thành phần cần tạo

{ className: "my-class", style: { color: "red" } }, // Thuộc tính của thành phần

"Hello, world!" // Nội dung của thành phần );

24.JSX trong React là gì?

1. **JSX** là một phần của React, cho phép viết các thành phần React bằng cú pháp giống với HTML hoặc XML. JSX được trình bày dưới dạng một ngôn ngữ đánh dấu (markup language) giúp cho việc viết code của React dễ hiểu hơn. JSX cung cấp một cách dễ đọc và dễ hiểu hơn để tạo ra các thành phần React so với việc sử dụng phương thức React.createElement().
2. **Một số lý do để sử dụng JSX trong React:**

* Dễ đọc và dễ hiểu: Với JSX, các thành phần React được viết dưới dạng một ngôn ngữ đánh dấu giống như HTML hoặc XML, giúp cho mã nguồn trông rõ ràng và dễ hiểu hơn.
* Dễ dàng tích hợp các thành phần: Với JSX, việc tích hợp các thành phần React vào nhau dễ dàng hơn so với sử dụng các phương thức tạo thành phần như React.createElement().
* Kiểm tra cú pháp: JSX giúp cho việc kiểm tra cú pháp (syntax checking) dễ dàng hơn và cho phép lập trình viên phát hiện các lỗi cú pháp nhanh chóng.
* Có thể sử dụng các thuộc tính HTML: Với JSX, lập trình viên có thể sử dụng các thuộc tính HTML như class thay vì phải sử dụng className trong React.
* Tối ưu hóa hiệu suất: JSX cung cấp một cách dễ dàng để tối ưu hóa hiệu suất của ứng dụng React, vì nó cho phép các trình biên dịch biến đổi mã JSX thành mã JavaScript tối ưu hóa cho hiệu suất tốt hơn.

Vì vậy, JSX là một phần quan trọng của React và được sử dụng rộng rãi để tạo ra các thành phần React và cấu trúc UI trong dự án React.

25.Component trong React?

* Trong React, các component hoạt động giống như các hàm trả về các thành phần HTML
* Các component là các thành phần độc lập và có thể sử dụng lại.
* Các component thực hiện công việc giống như các functions trong JavaScript nhưng chúng độc lập và nhiệm vụ chính là trả về HTML thông qua hàm render
* Có 2 loại component là Function Component và Class Component.
* **Function Component:**

// Tạo function Car

function Car() { return <h2>Hi, I am also a Car!</h2>;}

//Hiển thị ở phần tử root

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<Car />);

* **Class Component:**

//Tạo class Car

class Car extends React.Component { render() { return <h2>Hi, I am a Car!</h2>; }}

//Hiển thị ở phần tử root

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<Car />);

26. ****Khái niệm Props****

* Props là 1 từ viết ngắn gọn của properties, và là 1 khái niệm trong ReactJS.
* Props là 1 đối tượng, nó lưu trữ các giá trị của các attribute (thuộc tính) của một thẻ (Tag).
* Props cung cấp cách thức để chúng ta truyền dữ liệu từ component này tới component khác.
* Props được truyền vào component tương tự như cách chúng ta truyền một đối số vào một hàm.
* Là cách mà component có thể nhận được các giá trị của thuộc tính truyền vào từ bên ngoài vào
* Props chỉ dùng để đọc, khi bạn định nghĩa một component dù là loại function hay class, thì nó không được phép thay đổi props của chính nó
* Sử dụng một thuộc tính để chuyển một màu cho component Car và sử dụng nó trong hàm render():

class Car extends React.Component {

render() {

return <h2>I am a {this.props.color} Car!</h2>; }

}

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<Car color="red"/>);

* Sử dụng Props với Function Component

function Car(props) {

return <h2>I am a {props.color} Car!</h2>;

}

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<Car color="red"/>);

27. ****Pure function là gì?****

* Hàm luôn trả về cùng một kết quả nếu được truyền vào các tham số không đổi. Nó không hề phụ thuộc vào bất kỳ trạng thái hoặc dữ liệu nào, cũng như những sự thay đổi trong khi chương trình đang được chạy. Nó chỉ phụ thuộc vào các tham số đầu vào của nó.
* Hàm này không tạo ra bất kỳ ảnh hưởng nào đến các đối tượng khác, chẳng hạn như các request, input, output hoặc data mutation.
* Đó là định nghĩa của một pure function. Nếu nó thỏa mãn 2 điều kiện trên thì nó chính là một pure function.
* Sau đây là một ví dụ về *pure function* để tính toán giá của sản phẩm sau thuế (Thuế 20%):

function priceAfterTax(productPrice) { return (productPrice \* 0.20) + productPrice; }

* Hàm trên thỏa mãn cả hai điều kiện của một pure function như đã nêu ở trên. Hàm này ko phụ thuộc vào bất cứ input nào khác ngoài các tham số được truyền vào, và nó cũng không hề thay đổi các dữ liệu khác cũng như là các hiệu ứng phụ.
* Nếu bạn chạy hàm này với cùng một tham số cả trăm lần thì nó vẫn sẽ luôn luôn trả về cùng một kết quả.

28. ****Observable side effects là gì?****

* **Observable side effect** là bất kỳ sự tương tác nào với bên ngoài từ bên trong hàm ví dụ như việc thay đổi giá trị biến ở bên ngoài hàm hay gọi hàm khác từ bên trong hàm (Nhưng có thể gọi pure function).

*Chú ý: Nếu một pure function gọi một pure function thì đây không phải là side effect và hàm gọi vẫn là pure function*

* **Side effects có thể bao gồm các trường hợp sau:**
* Tạo HTTP request
* Thay đổi dữ liệu
* In dữ liệu ra console hay màn hình
* Thao tác với DOM
* Math.random()
* Lấy thời gian hiện tại
* *Bản thân các hiệu ứng phụ (side effect) không xấu và thường được sử dụng. Nhưng một hàm thuần túy (pure function) thì không được có bất cứ một hiệu ứng phụ (side effect) nào. Không phải hàm nào cũng cần thiết phải là Pure Function.*

29. State là gì ?

* State là một object có thể được sử dụng để chứa dữ liệu hoặc thông tin về components, từ đó bạn có thể luân chuyển dữ liệu đến các thành phần trong Component và đến các Component khác.
* State có thể được thay đổi bất cứ khi nào mong muốn
* Trong các dự án React, state được dùng để phản hồi các yêu cầu từ người dùng, hay lưu trữ một dữ liệu nào đó trong components.

30.Component Life Cycle

Một thành phần React có thể trải qua bốn giai đoạn trong vòng đời của nó như sau.

* **Initialization**: Đây là giai đoạn mà thành phần được xây dựng với các Props đã cho và trạng thái mặc định. Điều này được thực hiện trong phương thức khởi tạo Component Class
* **Mounting**: Giai đoạn này được thực hiện sau khi quá trình initialization(khởi tạo) được hoàn thành. Nó thực hiện nhiệm vụ chuyển virtual DOM (DOM ảo) trong React thành DOM và hiển thị trên trình duyệt.
* **Updating**: là giai đoạn trạng thái của một thành phần được cập nhật và ứng dụng được render lại.
* **Unmounting:**là bước cuối cùng của vòng đời thành phần, nơi thành phần được xóa khỏi trang.

31. Hook là gì?

* Hooks là các hàm mà cho phép bạn “hook into (móc vào)” trạng thái của React và các tính năng vòng đời từ các hàm components. Hooks không hoạt động bên trong classes.
* Chúng cho phép bạn sử dụng React không cần classes
* React cung cấp một vài Hooks sẵn có như useState.
* Cũng có thể tạo Hooks của bạn để sử dụng lại những hành vi có trạng thái giữa các components khác nhau.

32. Quy tắc của Hooks?

* Hooks là các hàm Javascript, nhưng nó bắt buộc thêm hai quy tắc:
* Chỉ gọi Hooks trên cùng. Không gọi Hooks bên trong vòng lặp, điều kiện, hoặc các hàm lồng nhau.
* Chỉ gọi Hooks từ các React components dạng hàm. Không gọi Hooks từ hàm JavaScript bình thường. (Chỉ có một chỗ khác đúng để gọi Hooks)
* Với hook, bạn có thể viết các component trông sạch sẽ và dễ hiểu hơn bằng cách tách chúng thành các phần nhỏ và tái sử dụng chúng.

Các hook được bắt đầu bằng từ khóa "use" (ví dụ: useState, useEffect) để React biết rằng chúng là hook. Một số hook được sử dụng phổ biến trong React bao gồm:

* useState: cho phép bạn lưu trữ và cập nhật trạng thái của component.
* useEffect: cho phép bạn thực hiện các hành động như gọi API hoặc làm thay đổi DOM sau khi component đã được render.
* useContext: cho phép bạn chia sẻ dữ liệu giữa các component mà không cần truyền props qua nhiều tầng.
* useMemo và useCallback: giúp tối ưu hóa hiệu suất của ứng dụng bằng cách giảm số lần tính toán lại các giá trị.
* useRef: cho phép bạn tham chiếu đến một phần tử DOM cụ thể hoặc lưu trữ một giá trị không bị thay đổi khi component được render lại.

32. useState trong react? khi nào dùng useState?

Trong React, useState là một hook dùng để quản lý trạng thái (state) của component. Cụ thể, khi ta muốn cập nhật một giá trị trong component và khiến component tự động render lại với giá trị mới đó, ta có thể sử dụng useState.

**VD:** const [state, setState] = useState(initialState);

function handleClick() {

setState (state + 1); }

Ở đây, useState nhận vào giá trị ban đầu của state (trong trường hợp này là 0), và trả về một mảng gồm hai phần tử:

* Giá trị hiện tại của state (ở đây là state);
* Một hàm để cập nhật giá trị của state (ở đây là setState);
* Khi hàm setState được gọi, React sẽ cập nhật giá trị của state và render lại component với giá trị mới. Ta có thể sử dụng state và hàm cập nhật của nó bất cứ đâu trong component.

*Lưu ý:*

* *Component được render lại sau khi setState.*
* *InitialState chỉ dùng khởi tạo giá trị lần đầu.*
* *Set sate với callback( )? Có.VD:*

function handleClick() {

setState (state => state + 1);

setState (state => state + 1); } // giá trị state đc callback;

* *InitialState với callback( )? Có.Lấy giá trị được return của callback làm giá trị khởi tạo*
* *Set sate là cập nhật*

33. Mounted & Unmounted?

* "Mounting" là quá trình tạo ra một instance của một component React và đưa nó vào trong cây DOM. Quá trình này xảy ra khi component được khởi tạo và đưa vào cây DOM để hiển thị trên giao diện người dùng.
* "Unmounting" là quá trình xóa bỏ một component React khỏi cây DOM và giải phóng tài nguyên liên quan đến component đó. Quá trình này xảy ra khi component không còn được sử dụng và cần được xóa bỏ khỏi giao diện người dùng.
* Các khái niệm này rất quan trọng trong React để quản lý vòng đời của component và đảm bảo hiệu suất và tối ưu cho ứng dụng của bạn.

34.useEffect ?

* UseEffect là một hook cho phép bạn thực hiện các hành động phụ thuộc vào component lifecycle (vòng đời của component) và các props, state. Sử dụng để thực hiện các hành động liên quan đến side-effect trong một functional component.
* Các side-effect có thể bao gồm các thao tác như gọi API, tương tác với DOM, hoặc thay đổi global state.
* useEffect được sử dụng để thay thế cho các lifecycle methods trong React Class Component như componentDidMount, componentDidUpdate, và componentWillUnmount.
* Được sử dụng để cập nhật state, gọi các API bất đồng bộ, thiết lập các listeners cho các sự kiện, v.v.
* useEffect nhận vào hai tham số: một là một hàm thực hiện các side-effect, và hai là một mảng dependency chứa các biến mà nếu có sự thay đổi thì hàm được thực thi lại. Khi một component được render hoặc một biến dependency trong mảng thay đổi, hàm được truyền vào sẽ được thực thi.
* Trong đó:

1. effect function: là một hàm được thực thi khi component được render. Nó có thể trả về một hàm để dọn dẹp các tài nguyên bất đồng bộ hoặc listeners được thiết lập trong effect function.
2. cleanup function: là một hàm được thực thi khi component unmount hoặc khi effect function được thực thi lại.
3. dependencies: là một mảng các giá trị được theo dõi để định nghĩa khi nào effect function cần được thực thi lại. Nếu dependencies là một mảng rỗng, effect function chỉ được thực thi một lần khi component mount.

**useEffect(callback, [deps])**  
 \* callback: bắt buộc có, là càc hàm do LTV tự viết, thực hiện các side-effect (update DOM, call API, listen event...)  
 \* [deps]: không bắt buộc\*/  
**1.useEffect(callback)**  
 - Gọi callback mỗi khi component re-render => ít dùng  
 - Gọi callback sau khi thêm element vào DOM  
**2.useEffect(callback, [])**  
 - Chỉ gọi callback 1 lần khi component mounted  
**3.useEffect(callback, [deps])**  
 - [deps] có thể là biến (props) truyền từ bên ngoài vào hay state trong component => truyền vào useEffect.  
 - Callback đc gọi lại khi [deps] thay đổi (khi component re-render =>useEffect sẽ kiểm tra [deps] trước và sau  
 khi render có khác nhau hay không=> nếu khác thì gọi lại callback)  
***Lưu ý***

1.Callback luôn được gọi sau khi component đc mounted  
2.Cleanup function luôn đc gọi trướckhi component unmounted  
3.Clearup function luôn đc gọi trước khi callback được gọi (trừ lần mounted)

35. setInterval() trong react?

Trong React, bạn có thể sử dụng **setInterval()** để thực thi một hàm một số lần định kỳ trong một khoảng thời gian nhất định.

Tuy nhiên, để sử dụng setInterval trong React, bạn cần lưu ý rằng việc thay đổi state hoặc prop có thể dẫn đến việc render lại component, điều này có thể dẫn đến vấn đề hiệu năng và sự cố khác. Do đó,phải dùng clearInterval(intervalId) để ngăn React render lại component

36. setTimeout() trong react?

Trong React, **setTimeout()** là một hàm JavaScript được sử dụng để thiết lập một hàm được thực thi sau một khoảng thời gian nhất định. Nó được sử dụng để lên lịch cho việc thực thi các công việc bất đồng bộ sau một khoảng thời gian nhất định.

37. clearInterval() trong react?

Trong React, **clearInterval**() là một phương thức được sử dụng để hủy bỏ việc thực thi một hàm lặp lại với một khoảng thời gian cố định bằng cách sử dụng setInterval.Trong một số trường hợp, chúng ta có thể muốn dừng việc thực thi hàm lặp lại trước khi nó hoàn thành hoặc trước khi trang web bị xóa khỏi màn hình. Và giải phóng tài nguyên bộ nhớ được sử dụng bởi setInterval().

38. clearTimeout() trong react? Sử dụng vs setTimeout(), tương tự với clearInterval()

39. ****Web API?****

1. **Web API:** là hệ thống API được sử dụng trong các hệ thống website, cho phép bạn kết nối, trao đổi thông tin, dữ liệu qua lại hoặc cập nhật cơ sở dữ liệu.
2. Web API sẽ trả lại dữ liệu ở dạng JSON hoặc XML thông qua giao thức HTTP hoặc HTTPS.
3. Những tính năng của Web API:

* **Tự động hóa sản phẩm :** Đối với Web API, sẽ giúp người dùng có thể dễ dàng tự động quản lý được công việc.
* **Tích hợp linh động :** API cho phép lấy nội dung ở bất kỳ Website hay ứng dụng nào đó một cách dễ dàng, khiến trải nghiệm người dùng được tăng lên.
* **Cập nhật thông tin theo thời gian thực :**API giúp thay đổi và cập nhật những thông tin mới theo thời gian thực. Công nghệ này sẽ giúp những thông tin truyền đi tốt hơn, chính xác hơn và dịch vụ cung cấp cũng được linh hoạt hơn.

40. ****RESTful API là gì?****

* Một Web API được xây dựng tuân theo kiến trúc REST thì được gọi là RESTful API.
* **REST** (REpresentational State Transfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Mặc dù **REST** có thể được sử dụng với mọi giao thức, nhưng nó thường dùng HTTP cho Web API.
* **RESTful API** là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource (tài nguyên – ví dụ như là tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu di động,…
* Nó cũng bao gồm các trạng thái tài nguyên được định dạng và truyền tải qua HTTP). RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau.

41.****Những ràng buộc kiến trúc cơ bản của RESTful API****

Một RESTful API phải thỏa mãn những ràng buộc sau:

**1. Client–server (Máy khách – máy chủ)**

Ràng buộc này hoạt động dựa trên ý tưởng rằng máy khách và máy chủ phải hoàn toàn tách biệt và được phép phát triển riêng lẻ, độc lập.

**2. Stateless (Phi trạng thái)**

* Bất kỳ một RESTful API nào cũng phải phi trạng thái.
* Mỗi yêu cầu (request) từ máy khách đến máy chủ có thể được thực hiện độc lập và phải chứa mọi thông tin cần thiết để máy chủ hiểu và hoàn thành nó.
* Ngoài ra, yêu cầu của máy khách không thể lạm dụng bất kỳ thông tin nào trên máy chủ

**3. Cacheable (Lưu được vào bộ nhớ cache)**

* API phi trạng thái có thể tăng số lượng request, nhất là khi có nhiều cuộc gọi đến và đi.
* RESTful API được thiết kế để lưu trữ dữ liệu vào cache để tăng tính tái sử dụng.

**4. Uniform interface (Giao diện thống nhất)**

* REST áp dụng những nguyên tắc chung của kỹ thuật phần mềm cho giao diện thành phần để đảm bảo tổng thể kiến trúc hệ thống được đơn giản hóa, có tính thống nhất.
* REST cần nhiều ràng buộc kiến trúc cho các thành phần bên trong.

**5. Layered system (Hệ thống phân lớp)**

* Kiểu hệ thống phân lớp cho phép một kiến trúc chứa nhiều lớp phân cấp.
* Mỗi lớp sẽ có một chức năng và trách nhiệm cụ thể.
* Cách thức của REST là hạn chế hành vi của các thành phần trong một lớp.
* Mỗi thành phần hoàn toàn không thể thấy được những gì ở bên ngoài lớp mà chúng đang tương tác.

**6. Code on demand (Mã theo yêu cầu)**

* Mã theo yêu cầu là một ràng buộc tùy chọn, không bắt buộc của REST.
* Ràng buộc này cho phép người dùng mở rộng chức năng của máy khách.
* Họ có thể tải xuống và thực thi mã dưới dạng các ứng dụng nhỏ (applet) hoặc tập lệnh (scripts).

**Ưu điểm của RESTful API:**

* Giúp cho ứng dụng rõ ràng hơn
* Dữ liệu được trả về với nhiều định dạng khác nhau như: xml, html, json….
* Code đơn giản và ngắn gọn
* Chú trọng vào tài nguyên của hệ thống

**Cách hoạt động của RESTful API**

* REST hoạt động chủ yếu dựa vào các giao thức HTTP:
* GET (SELECT): Trả về một Resource hoặc một danh sách Resource.
* POST (CREATE): Tạo mới một Resource.
* PUT (UPDATE): Cập nhật thông tin cho Resource.
* DELETE (DELETE): Xoá một Resource.
* Những phương thức hay hoạt động này thường được gọi là CRUD tương ứng với Create, Read, Update, Delete

**RESTful API không sử dụng session và cookie, nó sử dụng một access\_token với mỗi request**

42.Status code

Khi chúng ta request một API nào đó thường thì sẽ có vài status code để nhận biết sau:

* 200 OK – Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
* 201 Created – Trả về khi một Resource vừa được tạo thành công.
* 204 No Content – Trả về khi Resource xoá thành công.
* 304 Not Modified – Client có thể sử dụng dữ liệu cache.
* 400 Bad Request – Request không hợp lệ
* 401 Unauthorized – Request cần có auth.
* 403 Forbidden – bị từ chối không cho phép.
* 404 Not Found – Không tìm thấy resource từ URI
* 405 Method Not Allowed – Phương thức không cho phép với user hiện tại.
* 410 Gone – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
* 415 Unsupported Media Type – Không hỗ trợ kiểu Resource này.
* 422 Unprocessable Entity – Dữ liệu không được xác thực
* 429 Too Many Requests – Request bị từ chối do bị giới hạn

43. Mock API trong react là gì?

* Mock API trong React là một kỹ thuật giả lập dữ liệu và API trên môi trường phát triển để giúp nhà phát triển kiểm thử ứng dụng một cách hiệu quả hơn.
* Khi phát triển một ứng dụng React, việc tương tác với các API thực tế có thể gây ra nhiều rắc rối, như là tốc độ mạng chậm, dữ liệu thay đổi liên tục hoặc sự cố bảo mật.
* Để giải quyết vấn đề này, các nhà phát triển có thể sử dụng mock API để tạo ra dữ liệu giả lập và tương tác với các API một cách dễ dàng hơn.
* Mock API thường được sử dụng để thay thế các API thực tế trong quá trình phát triển ứng dụng React.
* Các mock API thường được xây dựng bằng cách sử dụng các thư viện như axios-mock-adapter, json-server hoặc msw để tạo ra các kịch bản giả lập cho các yêu cầu API và trả về các phản hồi tương ứng.

44.Tại sao sử dụng Mock API?

**Lợi ích của Mock API là:**

* Các nhà phát triển frontend và các nhà phát triển phụ trợ có thể làm việc song song, do đó phát triển nhanh chóng.
* Giao diện người dùng có thể hoạt động như một ứng dụng độc lập trong quá trình phát triển mà không có bất kỳ phụ thuộc API phụ trợ nào.
* Dễ dàng demo
* Mock API có thể dễ dàng được thay thế bằng API thực khi nó đã sẵn sàng.
* Việc sử dụng mock API giúp nhà phát triển có thể tập trung vào việc phát triển chức năng của ứng dụng mà không cần phải lo lắng về các vấn đề kết nối và truy xuất dữ liệu từ các API thực tế.

45. ****Giới thiệu Axios****

1. **Axios** là một thư viện HTTP Client dựa trên Promise. Cơ bản thì nó cung cấp một API cho việc xử lý XHR (XMLHttpRequests).
2. Nó được sử dụng để gửi các yêu cầu HTTP, đặc biệt là các yêu cầu AJAX, từ phía máy khách cũng như phía máy chủ.
3. **Axios** đơn giản hóa công việc này bằng cách cung cấp một giao diện trừu tượng dễ sử dụng trên giao diện XMLHttpRequest của Javascript với một số chức năng và tùy chọn cấu hình thân thiện với người dùng.

**Đặc điểm của Axios**

* Tạo XMLHttpRequests từ trình duyệt
* Thực hiện các http request từ node.js
* Hỗ trợ Promise API
* Chặn request và response
* Chuyển đổi dữ liệu request và response
* Hủy requests
* Tự động chuyển đổi về dữ liệu JSON
* Hỗ trợ phía client để chống lại CSRF(tấn công giả mạo)

46. ****Promise để xử lý bất đồng bộ****

**Promise** sinh ra để xử lý kết quả của một hành động cụ thể, kết quả của mỗi hành động sẽ là thành công hoặc thất bại và Promise sẽ giúp chúng ta giải quyết câu hỏi “Nếu thành công thì làm gì? Nếu thất bại thì làm gì?”

Khi một Promise được khởi tạo thì nó có một trong ba trạng thái sau:

* Fulfilled: Hành động xử lý xong và thành công
* Rejected: Hành động xử lý xong và thất bại
* Pending: Hành động đang chờ xử lý hoặc bị từ chối

Trong đó, hai trạng thái Rejected và Fulfilled ta gọi là Settled, tức là đã xử lý xong.

**Cách Tạo Promise: var promise = new Promise(**callback)

Trong đó callback là một function có 2 tham số truyền vào như sau:

**var promise = new Promise(function**(resolve, reject){ });

* resolve: là một hàm callback xử lý cho hành động thành công
* reject là: một hàm callback xử lý cho hành động thất bại

47. ****Async/Await****

1. Là một cơ chế giúp bạn thực hiện các thao tác bất đồng bộ một cách tuần tự hơn.
2. Async/await vẫn sử dụng Promise nhưng mã nguồn của bạn sẽ trong sáng và dễ theo dõi.
3. Để sử dụng, bạn phải khai báo hàm với từ khóa **async, await**

**Async** – khai báo một hàm bất đồng bộ

* Tự động biến đổi một hàm thông thường thành một Promise.
* Khi gọi tới hàm async nó sẽ xử lý mọi thứ và được trả về kết quả trong hàm của nó.

**Async cho phép sử dụng Await.**

* **Await** sẽ được sử dụng ở trước các thao tác cần đồng bộ , tạm dừng việc thực hiện các hàm async
* Khi được đặt trước một Promise, nó sẽ đợi cho đến khi Promise kết thúc và trả về kết quả.
* Await chỉ làm việc với Promises, nó không hoạt động với callbacks.
* Await chỉ có thể được sử dụng bên trong các function async.

async function() {

try {

const user = await api.getUser('sun\_user');

const posts = await api.getPostsOfUser(user);

const comments = await api.getCommentsOfPosts(posts);

console.log(comments;

} catch (err) {

console.log(err);

}

}

***Cần lưu ý là kết quả trả về của async function luôn là một Promise.***

48. nesting component trong react?

**nesting component** là một khái niệm quan trọng, cho phép bạn tạo ra các thành phần (components) lồng nhau trong các thành phần khác.

**Nesting component** được sử dụng để tách các thành phần phức tạp thành các thành phần nhỏ hơn và dễ quản lý hơn. Các thành phần lồng nhau có thể truyền props cho các thành phần con của chúng và tái sử dụng các thành phần con đó trong nhiều thành phần khác nhau.

VD:

Component con nhận props component cha:

import React from "react";

class ChildComponent extends *React*.Component {

    render() {

        return (

            <>

               <div>ChildComponent: {*this*.props.name}</div>

            </>

        )

    }

}

export default ChildComponent;

Component Cha truyền props cho component con:

 <div>

<*ChildComponent* *name*={"one"} />

<*ChildComponent* *name*={"two"} />

<*ChildComponent* *name*={"three"} /><div/>

49. Sự khác nhau giữa toán tử rest và toán tử spread?

Toán tử rest (...) được sử dụng để gom các giá trị vào trong một mảng.

Toán tử spread (...) được sử dụng để truyền(sao chép) các giá trị của một mảng hoặc đối tượng vào một mảng hoặc đối tượng khác

50.Vitural DOM là gì?

Virtual DOM là một khái niệm trong lập trình web, đặc biệt là trong các framework JavaScript như React. Virtual DOM là một bản sao của DOM (Document Object Model) được tạo ra và lưu trữ trong bộ nhớ của trình duyệt, được tạo ra để mô phỏng cấu trúc của DOM.

Trong quá trình phát triển ứng dụng web, khi dữ liệu thay đổi, Virtual DOM sẽ được cập nhật và so sánh với DOM thực tế. Các thay đổi sẽ được tìm thấy và chỉ áp dụng cho phần của DOM thực tế cần được cập nhật. Việc sử dụng Virtual DOM giúp tối ưu hóa hiệu suất của ứng dụng web, đặc biệt là khi có nhiều thay đổi xảy ra trên trang web.

51Khái niệm React Lifecycle Methods?

React Lifecycle Methods là một tập hợp các phương thức mà React cung cấp để cho phép các thành phần (components) trong ứng dụng React thực hiện các hành động tương ứng với các trạng thái khác nhau trong quá trình rendering và updating.

Các Lifecycle Methods của React có thể được chia thành ba nhóm chính:

1. Mounting Methods: được gọi khi một component được tạo và được đưa vào DOM. Những phương thức này bao gồm: constructor, static getDerivedStateFromProps, render, componentDidMount.
2. Updating Methods: được gọi khi một component được cập nhật. Những phương thức này bao gồm: static getDerivedStateFromProps, shouldComponentUpdate, render, getSnapshotBeforeUpdate, componentDidUpdate.
3. Unmounting Methods: được gọi khi một component bị xóa khỏi DOM. Phương thức này là componentWillUnmount.

Những phương thức này giúp cho React có thể quản lý được các thành phần của ứng dụng và đảm bảo rằng chúng sẽ được cập nhật và hiển thị đúng cách. Việc sử dụng đúng Lifecycle Methods cũng giúp cho ứng dụng React hoạt động hiệu quả và dễ bảo trì hơn.

52. khái niệm componentDidMount() trong react và code ví dụ?

**componentDidMount**() là một trong những Lifecycle Method của React được gọi khi một component được hiển thị lần đầu tiên trên trang web.

Phương thức này được sử dụng để thực hiện các hành động sau khi component đã được render và được đưa vào DOM.

Các hành động thường được thực hiện trong phương thức này bao gồm tải dữ liệu từ một API, gửi các yêu cầu AJAX, hoặc khởi tạo các plugin của bên thứ ba.

Ví dụ về cách sử dụng componentDidMount() trong React:

import React, { Component } from 'react';

class ExampleComponent extends Component {

constructor(props) {

super(props);

this.state = { data: null };

}

componentDidMount() {

// Gửi yêu cầu AJAX để lấy dữ liệu

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

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

.then(data => {

// Lưu dữ liệu vào state của component

this.setState({ data });

});

}

53. Khái niệm componentDidUpdate() trong react và code ví dụ?

**componentDidUpdate**() là một trong những Lifecycle Method của React được gọi khi một component đã được render lại(không được gọi cho lần render đầu tiên).

Phương thức này được sử dụng để thực hiện các hành động sau khi component đã được cập nhật và được đưa vào DOM. Các hành động thường được thực hiện trong phương thức này bao gồm xử lý dữ liệu, kiểm tra các props hoặc state mới, hoặc cập nhật các thuộc tính CSS.

Ví dụ về cách sử dụng componentDidUpdate() trong React:

import React, { Component } from 'react';

class ExampleComponent extends Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

componentDidUpdate(prevProps, prevState) {

// Kiểm tra nếu state của component đã thay đổi

if (this.state.count !== prevState.count) {

console.log('Count has been updated.');

}

}

handleButtonClick = () => {

// Tăng count lên 1

this.setState(prevState => ({ count: prevState.count + 1 }));

}

54. khái niệm componentWillUnmount() trong react và code ví dụ?

**componentWillUnmount**() là một trong những Lifecycle Method của React được gọi khi một component sắp bị xóa khỏi DOM.

Phương thức này được sử dụng để thực hiện các hành động để giải phóng bộ nhớ hoặc dọn dẹp các hoạt động của component trước khi nó bị xóa.

Ví dụ về cách sử dụng componentWillUnmount() trong React:

import React, { Component } from 'react';

class ExampleComponent extends Component {

constructor(props) {

super(props);

this.state = { intervalId: null, count: 0 };

}

componentDidMount() {

// Tạo một interval để tăng count lên 1 mỗi giây

const intervalId = setInterval(() => {

this.setState(prevState => ({ count: prevState.count + 1 }));

}, 1000);

this.setState({ intervalId });

}

componentWillUnmount() {

// Xóa interval khi component sắp bị xóa khỏi DOM

clearInterval(this.state.intervalId);

}

render() {

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

}

}

export default ExampleComponent;

57. khái niệm và cách sử dụng BrowserRouter, Route, Routes trong react?

1. **BrowserRouter** là một thành phần trong React Router giúp cho việc quản lý định tuyến trong ứng dụng dễ dàng hơn.

Nó sử dụng HTML5 history API để đồng bộ hóa URL với trạng thái của ứng dụng và giúp cho các thao tác điều hướng trở nên trơn tru hơn.

Để sử dụng BrowserRouter, chúng ta cần import nó từ thư viện react-router-dom và bọc toàn bộ ứng dụng trong BrowserRouter:

import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(

<BrowserRouter>

<App />

</BrowserRouter>,

document.getElementById('root')

);

1. **Route** là một thành phần trong React Router giúp cho việc xác định định tuyến và hiển thị nội dung trên trang dễ dàng hơn.

Route sẽ kiểm tra xem URL hiện tại có khớp với đường dẫn mà bạn đã cấu hình hay không và nếu có, nó sẽ hiển thị nội dung tương ứng.

Để sử dụng Route, chúng ta cần import nó từ thư viện react-router-dom và khai báo các đường dẫn mà chúng ta muốn xác định trong ứng dụng:

import { Route } from 'react-router-dom';

function App() {

return (

<div>

<Route exact path="/" component={Home} />

<Route path="/about" component={About} />

<Route path="/contact" component={Contact} />

</div>

);

}

1. Routes là một thành phần quan trọng được giới thiệu nhằm thay thế cho Route, Switch và Redirect trong phiên bản trước đó. Thành phần này cho phép chúng ta quản lý định tuyến (routing) trong ứng dụng một cách dễ dàng và linh hoạt hơn.

Routes có thể chứa nhiều Route và cho phép chúng ta xác định đường dẫn (path) một cách linh hoạt hơn, bao gồm cả đường dẫn phân cấp (nested routes). Để sử dụng Routes, chúng ta cần import nó từ thư viện react-router-dom: import { Routes, Route } from 'react-router-dom';

* Sử dụng Routes và Route chung vs nhau

function App() {

return (

<div>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

<Route path="/contact" element={<Contact />} />

<Route path="/products" element={<Products />}>

<Route path="/" element={<AllProducts />} />

<Route path=":id" element={<Product />} />

</Route>

<Route path="\*" element={<NotFound />} />

</Routes>

</div>

);

}

58. Links & NavLinks Trong React.JS Routers?

1. Links và NavLinks là hai thành phần quan trọng giúp tạo liên kết (link) giữa các trang và thực hiện định tuyến trong ứng dụng React.
2. **Links** là thành phần đơn giản nhất, nó được sử dụng để tạo liên kết (link) giữa các trang trong ứng dụng. Khi người dùng nhấp vào liên kết, React Router sẽ chuyển hướng đến đường dẫn (path) tương ứng.
3. **NavLinks** tương tự như Links, nhưng nó được thiết kế để tạo ra các liên kết (link) điều hướng được tạo thành bằng HTML. NavLinks hỗ trợ trạng thái hoạt động (active state), giúp người dùng dễ dàng nhận biết trang hiện tại đang được truy cập.
4. Để sử dụng **Link,NavLink** chúng ta cần import nó từ thư viện react-router-dom:

import { NavLink, Link } from 'react-router-dom';

59. Khái niệm redux trong react?

+ **Redux** là một thư viện quản lý trạng thái (state management) cho các ứng dụng JavaScript, đặc biệt là cho các ứng dụng React. Redux giúp các nhà phát triển xây dựng các ứng dụng lớn và phức tạp một cách dễ dàng và hiệu quả hơn.

+ **Redux** lưu trữ toàn bộ trạng thái của ứng dụng trong một trạng thái (state) duy nhất nằm ngoài các thành phần React, và cung cấp các phương thức để cập nhật trạng thái này. Khi một hành động (action) được gửi đến Redux, Redux sẽ sử dụng reducer để cập nhật trạng thái. Các thành phần React có thể truy cập trạng thái bằng cách sử dụng các props.

Cơ chế hoạt động : React thực hiện 1 or nhiều hành động (Action) => Redux sẽ đón hành động và lưu sự thay đổi vào state của Redux => chuyển sự thay đổi về React và React hiển thị lên cho người dùng.

**1 số khái niệm của Redux:**

* **Redux Store** là nơi lưu trữ trạng thái của ứng dụng và quản lý các hành động (actions) được gửi đến để thay đổi trạng thái đó. Store được tạo ra bằng cách sử dụng hàm createStore() của Redux.
* **Action** đại diện cho một sự kiện trong ứng dụng. Mỗi action đều phải có một thuộc tính type để định danh loại hành động đó. Các thuộc tính khác của action có thể được sử dụng để truyền các dữ liệu hoặc thông tin khác cần thiết cho việc xử lý hành động đó.

+ Mỗi action creator trả về một object đại diện cho action. Object này phải có thuộc tính type để định danh loại hành động đó, và có thể có các thuộc tính khác để truyền các dữ liệu hoặc thông tin khác cần thiết cho việc xử lý hành động đó.

+ Sau khi khai báo các action, chúng ta có thể sử dụng chúng để dispatch các hành động tương ứng đến Redux Store.

* **Reducers** là các hàm pure function được sử dụng để xử lý các hành động (actions) và thay đổi trạng thái của Redux store. Reducers nhận vào hai tham số là trạng thái hiện tại (state) và một hành động (action), và trả về trạng thái mới của ứng dụng.

+ Để sử dụng reducer này, chúng ta cần kết hợp nó với Redux store bằng cách sử dụng hàm createStore() của Redux.

60. Khái niệm và cách hoạt động Provider trong react-redux?

* **Provider** là một component của react-redux được sử dụng để cung cấp Redux store cho toàn bộ ứng dụng React.
* **Provider** nhận vào một store object và được bọc ngoài cùng của ứng dụng React.
* Khi Provider được sử dụng, toàn bộ component con trong ứng dụng có thể truy cập đến store thông qua hàm connect() của react-redux.

Cách hoạt động của Provider như sau:

1. Provider nhận vào một Redux store object thông qua props.
2. Provider lưu trữ store object trong context của React.
3. Provider render các component con bên trong nó.
4. Các component con sử dụng connect() để truy cập đến store object thông qua context của React.

61.Khái niệm useContext() ?

**useContext()** là một hook được cung cấp bởi thư viện React để truy cập vào giá trị của một Context. Context trong React là một cách để chia sẻ dữ liệu giữa các thành phần con trong cây thành phần mà không cần truyền dữ liệu qua các props qua lại.

Khi sử dụng **useContext()** bạn cần truyền vào đối tượng Context đã được tạo trước đó. useContext() sẽ trả về giá trị hiện tại của Context đó.

62.So sanh useContext() và redux ?

**useContext()** và **Redux** là hai cách khác nhau để quản lý trạng thái (state) trong ReactJS.

1. **Quản lý trạng thái:**

* **useContext**():là một hook được cung cấp bởi React để truy cập vào giá trị của một Context. Context được sử dụng để chia sẻ trạng thái giữa các thành phần con trong cây thành phần.
* **Redux**: là một thư viện quản lý trạng thái nâng cao và mạnh mẽ. Nó sử dụng một cơ chế gọi là "Store" để lưu trữ và quản lý trạng thái ứng dụng. Trạng thái được lưu trữ trong một "store" duy nhất và các thành phần có thể truy cập và thay đổi trạng thái thông qua các hành động (actions) và bộ xử lý (reducers).

1. **Cơ chế cập nhật trạng thái:**

* **useContext**(): Khi trạng thái được thay đổi trong Context, React sẽ tự động render lại các thành phần con được kết nối với Context đó.
* **Redux**: Khi trạng thái được thay đổi trong Redux, Redux sẽ thông báo cho React biết và React sẽ render lại các thành phần bị ảnh hưởng. Quá trình này được gọi là "flux" trong Redux.

1. **Quy mô và phức tạp:**

* **useContext**():là một cách đơn giản để quản lý trạng thái trong một ứng dụng nhỏ hoặc trong một phạm vi hạn chế.
* **Redux**: là một giải pháp phức tạp hơn và mạnh mẽ hơn để quản lý trạng thái trong các ứng dụng lớn và phức tạp. Nó cung cấp một kiến trúc rõ ràng, cho phép quản lý trạng thái trên toàn bộ ứng dụng một cách dễ dàng và hiệu quả.

Bottom of Form