1. ReactJS là gì?

ReactJS là một thư viện JavaScript dùng để xây dựng các thành phần giao diện (UI) có thể tái sử dụng theo mô hình MVC (Model-View-Controller). React chủ yếu được sử dụng để phát triển các ứng dụng web đơn trang (Single Page Application - SPA) nhờ vào kiến trúc component-based (dựa trên thành phần), cơ chế cập nhật nhanh với Virtual DOM, và khả năng quản lý nội dung động mà không cần tải lại toàn bộ trang. React sử dụng JSX để viết mã.

Các tính năng quan trọng của React:

- **Virtual DOM:** React sử dụng Virtual DOM để cập nhật và hiển thị các thành phần một cách hiệu quả, giúp cải thiện hiệu suất bằng cách giảm thao tác trực tiếp trên DOM thực.
- **Kiến trúc Component-Based:** UI trong React được xây dựng từ các thành phần có thể tái sử dụng, giúp mã nguồn dễ bảo trì và mở rộng.
- **Hooks:** Hooks giúp các component hàm có thể quản lý state và các hiệu ứng phụ, làm cho chúng mạnh mẽ và linh hoạt hơn.
- **Server-Side Rendering (SSR):** React hỗ trợ SSR, nơi nội dung HTML được tạo trên máy chủ và gửi đến trình duyệt, giúp cải thiện hiệu suất và SEO.
- **React Router:** Công cụ này giúp định tuyến trong React, cho phép xây dựng các ứng dụng có nhiều trang trong một SPA.

2. Giải thích kiến trúc MVC

Mô hình MVC (Model-View-Controller) là một kiến trúc phần mềm giúp tách biệt ứng dụng thành ba thành phần chính:

- **Model:** Xử lý dữ liệu, logic nghiệp vụ và trạng thái của ứng dụng.
- **View:** Chịu trách nhiệm hiển thị giao diện người dùng, nhận dữ liệu từ Model và hiển thị nó cho người dùng.
- **Controller:** Nhận đầu vào từ người dùng, xử lý và cập nhật Model hoặc View tương ứng.

MVC giúp tăng tính tổ chức và bảo trì dễ dàng hơn trong quá trình phát triển ứng dụng.

3. Các thành phần chính của React

React được xây dựng từ năm thành phần cốt lõi:

1. **Component (Thành phần):** Các khối xây dựng UI có thể tái sử dụng, mỗi thành phần trả về một đoạn HTML.
2. **JSX:** Một cú pháp mở rộng của JavaScript cho phép viết mã HTML trong React.
3. **Props và State:** Props là dữ liệu truyền giữa các component, trong khi State là dữ liệu nội bộ của một component.
4. **Context:** Cung cấp dữ liệu thông qua các component mà không cần truyền qua từng cấp bằng props.
5. **Virtual DOM:** Một bản sao nhẹ của DOM thực giúp React cập nhật UI hiệu quả hơn.

4. Giải thích Props và State trong React với sự khác biệt

Props là dữ liệu được truyền từ component cha sang component con, trong khi **State** là dữ liệu nội bộ của một component và không thể truyền ra ngoài.

Đặc điểm	Props	State
Dữ liệu được truyền giữa các component	✅ Có	❌ Không
Có thể thay đổi (Mutable)	❌ Không	✅ Có
Dùng được trong component hàm và class	✅ Có	✅ Có (trước React 16.0, chỉ có class component)
Chỉ đọc	✅ Có	❌ Không, có thể ghi

5. Virtual DOM trong React là gì?

Virtual DOM là một bản sao trong bộ nhớ của DOM thực. Nó giúp React cập nhật và hiển thị UI một cách hiệu quả hơn bằng cách so sánh trạng thái hiện tại với trạng thái trước đó để tìm ra sự khác biệt (diffing) trước khi cập nhật DOM thực.

Cách Virtual DOM hoạt động:

1. **Rendering hiệu quả:** Virtual DOM lưu trữ một bản sao của DOM thực để tối ưu quá trình cập nhật giao diện.
2. **Thuật toán Diffing:** So sánh phiên bản Virtual DOM cũ và mới để xác định sự thay đổi nhỏ nhất cần áp dụng lên DOM thực.
3. **Batch Updates:** Gộp các thay đổi lại thay vì cập nhật DOM từng chút một, giúp cải thiện hiệu suất.
4. **Cập nhật nhanh hơn:** Vì thao tác trên DOM thực tốn kém, React chỉ cập nhật những phần cần thiết sau khi so sánh Virtual DOM.
5. **Giao diện khai báo:** React cho phép lập trình viên viết code theo phong cách khai báo, React sẽ quyết định cách cập nhật UI một cách tối ưu.

6. JSX là gì?

JSX là một phần mở rộng cú pháp của JavaScript được sử dụng trong React để tạo các phần tử giao diện. Nó giúp viết mã React dễ đọc hơn bằng cách cho phép sử dụng cú pháp tương tự HTML trong JavaScript. Để nhúng một biểu thức JavaScript vào JSX, ta đặt nó trong dấu ngoặc nhọn {}.

7. Component trong React và các loại của nó

Component là các khối xây dựng UI trong React, giúp chia nhỏ giao diện thành các phần nhỏ có thể tái sử dụng.

Có hai loại component chính:

- **Functional Components:** Là các hàm JavaScript đơn giản, thường được sử dụng để hiển thị giao diện dựa trên props.
- **Class Components:** Có cấu trúc phức tạp hơn, có thể quản lý state và sử dụng lifecycle methods để kiểm soát hành vi của ứng dụng.

8. Trình duyệt đọc JSX như thế nào?

Trình duyệt không thể hiểu JSX trực tiếp mà chỉ có thể đọc JavaScript thuần. Để trình duyệt hiểu được JSX, mã JSX cần được biên dịch thành JavaScript bằng công cụ **Babel** trước khi thực thi.

9. Cách tạo một sự kiện trong React

Trong React, sự kiện được tạo bằng cách gán một trình xử lý sự kiện (`onClick`, `onChange`, v.v.) vào một phần tử JSX. Khi sự kiện được kích hoạt, hàm xử lý sự kiện sẽ thực thi một hành động cụ thể, chẳng hạn như cập nhật state hoặc thực thi logic.

11. Tạo danh sách (List) trong React

Danh sách rất quan trọng trong việc phát triển giao diện người dùng. Chúng thường được sử dụng để hiển thị menu hoặc danh sách dữ liệu trên trang web. Trong React, danh sách có thể được tạo bằng cách sử dụng phương thức `map()` của mảng để lặp qua các phần tử và hiển thị chúng dưới dạng danh sách.

12. Key trong React là gì?

Key là một thuộc tính đặc biệt trong React, được sử dụng khi tạo danh sách các phần tử. Key giúp React xác định các phần tử nào đã thay đổi, được cập nhật hoặc bị xóa khỏi danh sách. Nói cách khác, key giúp React tối ưu hóa quá trình cập nhật danh sách bằng cách xác định duy nhất từng phần tử.

13. Cách viết comment trong React

Có hai cách viết comment trong React:

- **Comment nhiều dòng:** Sử dụng `/* */` để viết comment trải dài trên nhiều dòng.
- **Comment một dòng:** Sử dụng `//` để viết comment trên một dòng duy nhất.

14. Sự khác biệt giữa React và Angular

Đặc điểm	React	Angular
Loại	Thư viện JavaScript	Framework
Cập nhật DOM	Sử dụng Virtual DOM	Cập nhật trực tiếp trên DOM thực
Kiến trúc	Theo mô hình MVC (Model-View-Control)	Theo mô hình MVVM (Model-View-ViewModel)
Khả năng mở rộng (Scalability)	Cao	Thấp hơn React
Data Binding	Một chiều (Uni-directional)	Hai chiều (Bi-directional)
Hiệu suất	Cao hơn do sử dụng Virtual DOM	Chậm hơn do thao tác trên DOM thực

15. Công dụng của phương thức `render()` trong React

Phương thức `render()` trong React được sử dụng để hiển thị HTML lên trang web. Nó giúp hiển thị mã JSX bên trong một phần tử HTML cụ thể. Trong `render()`, chúng ta có thể đọc `props` và `state` rồi trả về JSX để hiển thị trong component gốc của ứng dụng.

16. State trong React là gì?

State là một đối tượng trong React chứa dữ liệu hoặc thông tin có thể thay đổi trong suốt vòng đời của một component. State giúp React kiểm soát hành vi và hiển thị của component dựa trên sự thay đổi dữ liệu.

17. Props trong React là gì?

Props (viết tắt của **properties**) là cách React cho phép truyền dữ liệu từ component cha sang component con. Props là các đối tượng có thể được sử dụng bên trong một component để hiển thị dữ liệu động.

18. Higher-Order Component (HOC) trong React là gì?

Higher-Order Component (HOC) là một kỹ thuật nâng cao trong React để tái sử dụng logic của component. HOC nhận vào một component gốc và trả về một component mới với các chức năng được mở rộng. Điều này giúp giảm việc lặp lại logic trong nhiều component khác nhau.

19. Sự khác biệt giữa Functional Component và Class Component

Đặc điểm	Functional Components	Class Components
Định nghĩa	Là một hàm JavaScript nhận `props` làm tham số	Là một class mở rộng từ `React.Component` và sử dụng phương thức `render()`
Rendering	Không cần phương thức `render()`	Cần phương thức `render()` để trả về JSX
State	Sử dụng hooks như `useState`	Sử dụng `this.state` và `this.setState`
Lifecycle Methods	Không thể sử dụng trực tiếp phương thức vòng đời	Có thể sử dụng các phương thức vòng đời như `componentDidMount`
Constructor	Không cần constructor	Cần constructor để quản lý state

20. One-Way Data Binding trong React

React sử dụng cơ chế **one-way data binding** (ràng buộc dữ liệu một chiều), nghĩa là dữ liệu chỉ có thể truyền từ **component cha** đến **component con**. Điều này giúp đảm bảo dữ liệu luôn có luồng truyền tải rõ ràng, giúp dễ dàng kiểm soát và debug.

21. Context API trong React là gì?

Context API là một cơ chế trong React giúp chia sẻ dữ liệu (như theme, ngôn ngữ, user authentication, v.v.) giữa các component mà không cần truyền props qua từng cấp. Nó bao gồm:

- **Provider**: Cung cấp dữ liệu cho toàn bộ cây component.

- **Consumer hoặc useContext()**: Nhận dữ liệu từ Context mà không cần truyền props thủ công.

22. Vai trò của `shouldComponentUpdate()` trong React

`shouldComponentUpdate()` là một phương thức vòng đời giúp kiểm soát việc re-render của component khi có thay đổi về **props** hoặc **state**. Nếu nó trả về `false`, React sẽ bỏ qua quá trình re-render component đó, giúp tối ưu hiệu suất.

23. `dangerouslySetInnerHTML` trong React được sử dụng để làm gì?

`dangerouslySetInnerHTML` là một thuộc tính trong React cho phép chèn trực tiếp HTML vào component. Tuy nhiên, nó có thể gây ra rủi ro bảo mật như **Cross-Site Scripting (XSS)**, nên cần được sử dụng cẩn thận, đặc biệt khi hiển thị nội dung từ nguồn bên ngoài.

24. Pure Component trong React là gì?

Pure Component là một loại component trong React chỉ re-render khi **props hoặc state thay đổi**. React cung cấp `React.PureComponent`, giúp tự động thực hiện **shallow comparison** (so sánh nông) để quyết định có cần re-render hay không, từ đó giúp tối ưu hiệu suất.

25. Ý nghĩa của `setState()` trong React

`setState()` là phương thức được sử dụng để cập nhật **state** trong component. Khi state thay đổi, React sẽ **tự động re-render component** và các component con liên quan để phản ánh sự thay đổi.

🚀 Đây là những câu hỏi thường gặp trong phỏng vấn React ở mức độ trung cấp, phù hợp với cả fresher và người có kinh nghiệm từ 1-2 năm. Bạn có muốn mở rộng thêm nội dung nào không? 😊

26. Conditional Rendering trong React là gì?

Conditional Rendering là quá trình hiển thị component dựa trên một điều kiện cụ thể. Điều này giúp giao diện trở nên linh hoạt và phản hồi nhanh theo trạng thái ứng dụng.

Ví dụ:

Nếu `isLoggedIn` là `false`, component `DisplayLoggedOut` sẽ được hiển thị, ngược lại `DisplayLoggedIn` sẽ xuất hiện:

```
{isLoggedIn == false ? <DisplayLoggedOut /> : <DisplayLoggedIn />}
```

27. React Router là gì?

React Router là một thư viện dùng để quản lý điều hướng (routing) trong React. Nó giúp:

✅ Chuyển đổi giữa các trang mà không tải lại trang.

✅ Đồng bộ hóa giao diện với URL.

✅ Tạo Single Page Applications (SPA) một cách dễ dàng.

Cài đặt React Router:

```
npm i react-router-dom
```

28. Các thành phần chính của React Router

React Router bao gồm các thành phần sau:

- **Router (BrowserRouter)**: Component cha chứa các tuyến đường.
- **Switch**: Đảm bảo chỉ render **một route phù hợp nhất**, thay vì tất cả route khớp.
- **Route**: Kiểm tra URL hiện tại và hiển thị component tương ứng.
- **Link**: Dùng để tạo liên kết chuyển hướng giữa các route.

29. Các phương thức trong vòng đời Component

React Component có 4 giai đoạn vòng đời:

1. **Initialization (Khởi tạo)**: Component được khởi tạo với Props và State.
2. **Mounting (Gắn vào DOM)**: JSX được render lần đầu tiên.
3. **Updating (Cập nhật)**: Component cập nhật khi State hoặc Props thay đổi.
4. **Unmounting (Gỡ khỏi DOM)**: Component bị xóa khỏi giao diện.

30. Các phương thức trong giai đoạn Mounting

Mounting là quá trình component được thêm vào DOM. Nó bao gồm các phương thức:

- **`componentWillMount()`**: Chạy ngay trước khi component được gắn vào DOM (❌ Đã bị loại bỏ từ React 16.3).
- **`componentDidMount()`**: Chạy ngay sau khi component được render lần đầu tiên, thường dùng để gọi API hoặc cập nhật DOM.

31. `this.setState()` trong React là gì?

Phương thức `setState()` được sử dụng để cập nhật state trong React. Khi `setState()` được gọi, React sẽ cập nhật state và thực hiện quá trình **re-render** component để hiển thị dữ liệu mới.

- 🔹 React cho phép **cập nhật một phần của state** mà không cần thay đổi toàn bộ object.
- 🔹 Có thể gọi **nhiều lần `setState()`** để cập nhật từng thuộc tính riêng biệt.

32. `ref` trong React dùng để làm gì?

`ref` trong React cho phép truy cập trực tiếp vào **DOM elements** hoặc **React elements** mà không cần thông qua props.

- ✅ Dùng để **lấy giá trị của input**, thực hiện **focus**, hoặc **tương tác với phần tử DOM**.
- ✅ Hữu ích khi làm việc với thư viện bên ngoài như **chart libraries hoặc video players**.

Cú pháp sử dụng `ref`:

```
const node = this.myCallRef.current;
```

33. Hooks trong React là gì?

Hooks là tính năng được giới thiệu từ **React 16.8**, giúp sử dụng state và các tính năng của React mà **không cần viết class component**.

🔹 **Không làm thay đổi các khái niệm cũ của React**.

🔹 **Thay thế cho lifecycle methods** như `componentDidMount()`, `componentDidUpdate()`, `componentWillUnmount()`.

🔹 Các hooks phổ biến: `useState()`, `useEffect()`, `useContext()`, `useRef()`, `useReducer()`.

34. Hook `useState()` trong React là gì?

`useState()` là hook giúp **quản lý state trong functional component**.

✅ **Dùng để khai báo biến state trong function component**.

✅ **Mỗi `useState()` chỉ quản lý một biến** nhưng có thể dùng nhiều lần để quản lý nhiều state khác nhau.

Cú pháp:

```
const [state, setState] = useState(initialState);
```

- `state`: Giá trị hiện tại của state.
- `setState`: Hàm để cập nhật state.
- `initialState`: Giá trị khởi tạo của state.

35. Hook `useEffect()` trong React là gì?

`useEffect()` được sử dụng để **thay thế các lifecycle methods** trong class component như `componentDidMount()` và `componentDidUpdate()`.

✅ Dùng để **thực hiện side effects** như gọi API, thao tác với DOM, hoặc lưu dữ liệu vào localStorage.

✅ Có thể kiểm soát **khi nào effect được chạy lại** bằng cách truyền **dependency array**.

Cú pháp:

```
useEffect(callbackFunction, [dependencies]);
```

- `callbackFunction`: Hàm chứa logic thực thi.
- `[dependencies]`: Danh sách phụ thuộc quyết định khi nào effect được gọi lại. Nếu để trống `[]`, effect chỉ chạy một lần khi component mount.

36. React Fragments là gì?

Khi muốn render nhiều hơn một phần tử gốc (**root element**), trước đây chúng ta thường phải bọc chúng trong một thẻ `<div>`, điều này có thể tạo ra các **thẻ HTML dư thừa**.

✅ **Từ React 16.2**, **Fragments** được giới thiệu giúp giải quyết vấn đề này.

✅ Fragments giúp **tránh thẻ bao không cần thiết** và **cải thiện hiệu suất** khi render.

👉 **Cú pháp sử dụng:**

```
<React.Fragment>
    <h2>Child-1</h2>
    <p>Child-2</p>
</React.Fragment>
```

Hoặc viết gọn hơn:

```
<>
    <h2>Child-1</h2>
    <p>Child-2</p>
```

</>

37. Công cụ React Developer Tools là gì?

🔹 **React Developer Tools** là một extension của **Chrome DevTools** giúp **debug ứng dụng React**.

🔹 Hỗ trợ **kiểm tra cây component**, **state**, **props**, **hooks**, và hiệu suất của ứng dụng.

🔹 Dùng để kiểm tra **cách React cập nhật và render**.

👉 **Cài đặt:**

- Chrome: Tìm **React Developer Tools** trên **Chrome Web Store**.
- Dùng npm: `npm install -g react-devtools`

38. Cách sử dụng styles trong ReactJS?

React hỗ trợ nhiều cách để thêm **CSS** vào ứng dụng:

✅ **Inline Styles**

✅ **CSS Modules**

✅ **Styled Components**

👉 **CSS Modules** giúp **giới hạn phạm vi CSS** chỉ trong component đó.

🔹 **Cú pháp sử dụng CSS Modules**:

```
import styles from './App.module.css';

function App() {
  return <div className={styles.container}>Hello World</div>;
}
```

Ưu điểm của CSS Modules:

✓ Tránh xung đột tên class.

✓ Mã CSS có tính module, dễ bảo trì.

39. Styled Components trong React là gì?

Styled-components là một thư viện cho phép viết CSS trực tiếp trong file JavaScript mà không cần tách file CSS riêng.

✅ **Dễ đọc, dễ sử dụng, không cần đặt tên class thủ công**.

✅ **Hỗ trợ sử dụng props trong CSS**, giúp tăng tính linh hoạt.

👉 **Cài đặt Styled Components:**

```
npm i styled-components
```

🔹 **Ví dụ tạo button với Styled Components:**

```
import styled from 'styled-components';

const Button = styled.button`
  width: 100px;
  cursor: pointer;
  background-color: blue;
  color: white;
`;

export default Button;
```

40. Prop Drilling là gì? Nhược điểm của nó?

🔹 **Prop Drilling** xảy ra khi **dữ liệu từ component cha cần truyền xuống component con nhiều cấp** mà không cần thiết.

🔹 Component trung gian **chỉ đóng vai trò truyền dữ liệu**, không sử dụng dữ liệu đó.

Nhược điểm của Prop Drilling:

❌ **Gây khó khăn khi bảo trì**: Khi cần sửa dữ liệu, phải chỉnh sửa nhiều component.

❌ **Làm code rườm rà**, đặc biệt khi ứng dụng lớn.

❌ **Hiệu suất kém** do mỗi lần re-render sẽ ảnh hưởng đến tất cả component trung gian.

👉 **Giải pháp thay thế Prop Drilling:**

✅ **React Context API**

✅ **Redux / Zustand / Recoil**

✅ **useReducer() với useContext()**

41. Conditional Rendering in React

🔹 **Conditional Rendering** giúp hiển thị giao diện khác nhau tùy vào điều kiện nhất định.

🔹 Ví dụ: Hiển thị nút **Login** nếu người dùng chưa đăng nhập, hoặc **Logout** nếu đã đăng nhập.

👉 **Ví dụ:**

```
const isLoggedIn = true;

return (
  <div>
    {isLoggedIn ? <button>Logout</button> : <button>Login</button>}
  </div>
);
```

42. Controlled Components trong React

🔹 **Controlled Components** là các component trong đó **dữ liệu của form được quản lý bởi state của React**.

🔹 Mỗi lần giá trị thay đổi, nó được cập nhật thông qua **state**, thay vì thao tác trực tiếp với DOM như trong HTML thuần.

👉 **Ví dụ:**

```
function ControlledInput() {
  const [value, setValue] = React.useState("");

  return (
    <input
      type="text"
      value={value}
      onChange={(e) => setValue(e.target.value)}
    />
  );
}
```

✅ **Ưu điểm:** Dữ liệu có thể kiểm soát được, dễ quản lý và debug.

❌ **Nhược điểm:** Cần cập nhật state mỗi khi thay đổi, có thể gây ảnh hưởng đến hiệu suất nếu không tối ưu.

43. Custom Hooks trong React

🔹 **Custom Hooks** là các **hàm JavaScript** bắt đầu bằng `use` và có thể sử dụng các hook khác bên trong.

🔹 Mục đích: **Tái sử dụng logic** trong nhiều component mà không lặp code (**DRY - Don't Repeat Yourself**).

👉 **Ví dụ tạo custom hook useCounter:**

```
function useCounter(initialValue = 0) {
  const [count, setCount] = React.useState(initialValue);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);

  return { count, increment, decrement };
}

// Sử dụng hook trong component
function Counter() {
  const { count, increment, decrement } = useCounter(10);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
    </div>
  );
}
```

✅ **Ưu điểm:**

✓ Tái sử dụng code dễ dàng.

✓ Giảm độ phức tạp của component.

44. Cách tối ưu mã React

🔹 Một số cách giúp **tối ưu code React** để cải thiện hiệu suất:

✅ **1. Tránh tạo hàm trong JSX** (*Dùng useCallback nếu cần*)

✅ **2. Dùng React.memo() để tránh re-render không cần thiết**

✅ **3. Dùng React Fragments để tránh thẻ `<div>` không cần thiết**

✅ **4. Lazy loading với React.lazy()**

✅ **5. Code splitting để tải component khi cần thiết**

👉 **Ví dụ sử dụng Lazy Loading để tải component động**

```
const MyComponent = React.lazy(() => import('./MyComponent'));

function App() {
  return (
    <React.Suspense fallback={<p>Loading...</p>}>
      <MyComponent />
    </React.Suspense>
  );
}
```

✅ **Lợi ích:**

✓ Giảm thời gian tải trang.

✓ Giúp ứng dụng chạy mượt hơn.

45. Sự khác nhau giữa `useRef` và `createRef`

Tính năng	`useRef`	`createRef`
Loại	Hook	Hàm
Vòng đời tham chiếu	Giữ nguyên qua các lần render	Tạo mới mỗi lần component re-render
Lưu trữ dữ liệu qua render	Có thể lưu giá trị giữa các lần render	Không lưu giá trị giữa các lần render
Giá trị trả về	Một đối tượng ref có thể thay đổi	Một đối tượng ref chỉ đọc
Trường hợp sử dụng	Dùng trong functional component để giữ giá trị hoặc truy cập DOM	Dùng trong class component để truy cập DOM

👉 **Ví dụ về useRef:**

```
function InputFocus() {
  const inputRef = React.useRef(null);

  return (
    <div>
      <input ref={inputRef} />
      <button onClick={() => inputRef.current.focus()}>Focus</button>
    </div>
  );
}
```

👉 **Ví dụ về createRef:**

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.inputRef = React.createRef();
  }

  focusInput = () => {
    this.inputRef.current.focus();
  };

  render() {
    return (
      <div>
        <input ref={this.inputRef} />
        <button onClick={this.focusInput}>Focus</button>
      </div>
    );
  }
}
```

✅ **Khi nào dùng useRef và createRef?**

- **Dùng useRef** trong functional component.

- **Dùng `createRef`** trong class component.

46. What is React-Redux?

🔹 **React-Redux** là một thư viện giúp quản lý **state toàn cục** trong ứng dụng React.

🔹 Nó giúp truyền state giữa các component **mà không cần prop drilling**.

🔹 Khi ứng dụng có nhiều component, việc quản lý state bằng **React Context** có thể trở nên phức tạp, **Redux** giúp giải quyết vấn đề này bằng cách lưu trữ **state tại một nơi duy nhất (store)**.

👉 **Cách dùng React-Redux:**

1️⃣ **Tạo store** để lưu state.

2️⃣ **Sử dụng Provider** để bọc ứng dụng, cho phép mọi component truy cập store.

3️⃣ **Dùng useSelector để lấy state** và **useDispatch để gọi action**.

47. Benefits of using React-Redux

✅ **1. Quản lý state tập trung** – Toàn bộ state được lưu tại một nơi duy nhất.

✅ **2. Tối ưu hiệu suất** – Chỉ những component nào bị ảnh hưởng bởi state thay đổi mới re-render.

✅ **3. Dễ debug** – Redux DevTools giúp theo dõi trạng thái và các hành động được dispatch.

✅ **4. Giữ dữ liệu lâu dài** – Redux có thể lưu state trên localStorage hoặc sessionStorage.

48. Core Components of React-Redux

🔹 **Redux có 4 thành phần chính:**

1️⃣ **Redux Store** – Lưu trữ toàn bộ state của ứng dụng.

2️⃣ **Actions** – Định nghĩa các sự kiện để thay đổi state.

3️⃣ **Action Creators** – Hàm tạo actions để gửi đi.

4️⃣ **Reducers** – Hàm nhận action và cập nhật state mới.

👉 **Ví dụ đơn giản:**

```
// Action
const increment = () => ({ type: "INCREMENT" });

// Reducer
const counterReducer = (state = 0, action) => {
  switch (action.type) {
    case "INCREMENT":
      return state + 1;
    default:
      return state;
  }
};

// Store
const store = createStore(counterReducer);
```

49. How to Combine Multiple Reducers in React?

- Khi ứng dụng lớn, ta cần nhiều reducer để quản lý nhiều phần khác nhau của state.
- **combineReducers** giúp kết hợp nhiều reducer thành một.

👉 **Ví dụ:**

```
import { combineReducers, createStore } from "redux";

// Reducers
const booksReducer = (state = [], action) => {
```

```
  switch (action.type) {
    case "ADD_BOOK":
      return [...state, action.payload];
    default:
      return state;
  }
};

const usersReducer = (state = [], action) => {
  switch (action.type) {
    case "ADD_USER":
      return [...state, action.payload];
    default:
      return state;
  }
};

// Combine Reducers
const rootReducer = combineReducers({
  books: booksReducer,
  users: usersReducer,
});

// Create Store
const store = createStore(rootReducer);
```

50. What is CORS in React?

🔹 **CORS (Cross-Origin Resource Sharing)** là cơ chế **cho phép hoặc chặn** các request giữa các domain khác nhau.

🔹 Khi frontend (React) gọi API từ backend khác domain, trình duyệt **chặn request** nếu backend không cho phép.

🔹 Để bật CORS, ta cần cấu hình trên backend hoặc frontend.

👉 **Cách bật CORS trong React:**

1️⃣ **Dùng axios với proxy (trong development)**

```
// Thêm vào package.json
"proxy": "http://localhost:5000"
```

2️⃣ **Dùng fetch với mode: 'cors'**

```
fetch("http://example.com/api", { mode: "cors" })
  .then(response => response.json())
  .then(data => console.log(data));
```

3️⃣ **Cấu hình CORS trên backend (Node.js/Express)**

```
const cors = require("cors");
app.use(cors());
```

✅ **Giải pháp tốt nhất**: Cấu hình backend để trả về **CORS headers** (`Access-Control-Allow-Origin`).

51. What is Axios and How to Use It in React?

🔹 **Axios** là thư viện giúp gửi các **HTTP request** (GET, POST, PUT, DELETE) một cách **bất đồng bộ (async)** trong React.

🔹 Hỗ trợ **Promise API**, **interceptors**, và **tự động chuyển đổi JSON**.

🔹 Để cài đặt Axios, chạy lệnh:

```
npm i axios
```

👉 **Ví dụ sử dụng Axios để fetch API trong React:**

```
import axios from "axios";
import { useEffect, useState } from "react";

const App = () => {
  const [data, setData] = useState([]);
```

```
  useEffect(() => {
    axios.get("https://jsonplaceholder.typicode.com/posts")
      .then(response => setData(response.data))
      .catch(error => console.error("Error:", error));
  }, []);

  return (
    <div>
      <h2>Posts</h2>
      <ul>
        {data.slice(0, 5).map(post => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
};

export default App;
```

52. Write a Program to Create a Counter with Increment and Decrement

👉 **React Counter App Example:**

```
import React, { useState } from "react";

const Counter = () => {
  const [counter, setCounter] = useState(0);

  return (
    <div>
      <h2>Counter: {counter}</h2>
      <button onClick={() => setCounter(counter +
1)}>Increment</button>
      <button onClick={() => setCounter(counter -
1)}>Decrement</button>
```

```
    </div>
  );
};

export default Counter;
```

📌 **Điểm quan trọng:**

✅ **useState** để quản lý state.

✅ **Sử dụng sự kiện onClick** để cập nhật state khi nhấn nút.

53. Why and How to Update State of Components Using Callback?

🔹 Khi cập nhật state dựa vào **giá trị trước đó**, ta nên dùng **hàm callback trong setState** để tránh lỗi.

🔹 React **batches state updates**, nên nếu không dùng callback, ta có thể cập nhật sai giá trị.

👉 **Ví dụ cập nhật state sai cách:**

```
const increment = () => {
  setCounter(counter + 1);
  setCounter(counter + 1); // Lỗi: Cả hai dòng đều dùng counter cũ
};
```

👉 **Cách đúng sử dụng callback:**

```
const increment = () => {
  setCounter(prevCounter => prevCounter + 1);
  setCounter(prevCounter => prevCounter + 1); // Đúng: dùng giá trị
mới nhất
};
```

54. What is React Material UI?

🔹 **Material UI (MUI)** là thư viện giao diện **React UI components** dựa trên **Material Design của Google**.

🔹 Cung cấp **các component sẵn có** như buttons, cards, modals, tables, icons,... giúp phát triển nhanh hơn.

🔹 Để cài đặt Material UI, chạy lệnh:

```
npm install @mui/material @emotion/react @emotion/styled
```

👉 **Ví dụ sử dụng Button của MUI:**

```
import Button from '@mui/material/Button';

function App() {
  return (
    <Button variant="contained" color="primary">
      Click Me
    </Button>
  );
}

export default App;
```

📌 **Lợi ích của MUI:**

✅ **Thiết kế chuẩn Material Design**

✅ **Hỗ trợ chủ đề (theme) tùy chỉnh**

✅ **Hiệu suất cao, dễ tích hợp với các dự án lớn**

55. What is Flux Architecture in Redux?

🔹 **Flux Architecture** là mô hình **luồng dữ liệu một chiều (unidirectional data flow)** để quản lý state trong ứng dụng.

🔹 **Redux** dựa trên Flux nhưng đơn giản hơn và tối ưu hơn.

📌 **Các thành phần chính của Flux/Redux:**

1️⃣ **Actions** – Gửi yêu cầu cập nhật state.

2️⃣ **Dispatcher** – Gửi action đến các store.

3️⃣ **Store** – Nơi lưu trữ dữ liệu ứng dụng.

4️⃣ **View (UI)** – Hiển thị dữ liệu từ store.

👉 **Luồng dữ liệu Redux (giống Flux nhưng không có Dispatcher):**

```
UI → Dispatch Action → Reducer → Update Store → Re-render UI
```

👉 **Ví dụ đơn giản của Redux store:**

```
const initialState = { count: 0 };

const counterReducer = (state = initialState, action) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    default:
      return state;
  }
};
```

📌 **Lợi ích của Flux/Redux:**

✅ **Giúp quản lý state dễ dàng**

✅ **Tránh prop drilling**

✅ **Dễ debug và bảo trì**

56. What Are Custom Hooks in React?

🔹 **Custom hooks** là **các hàm JavaScript tùy chỉnh** sử dụng các **built-in hooks** (như `useState`, `useEffect`) để tái sử dụng logic.

🔹 **Mục đích:** Giúp chia sẻ logic giữa các components mà **không cần lặp lại code**.

🔹 **Quy tắc:** Tên phải bắt đầu bằng `"use"` (ví dụ: `useFetchData`).

👉 **Ví dụ về Custom Hook (useFetch):**

```
import { useState, useEffect } from "react";
import axios from "axios";

const useFetch = (url) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    axios.get(url)
      .then(response => {
        setData(response.data);
        setLoading(false);
      })
      .catch(error => console.error("Error fetching data", error));
  }, [url]);

  return { data, loading };
};

export default useFetch;
```

👉 **Cách sử dụng trong Component:**

```
const App = () => {
  const { data, loading } =
useFetch("https://jsonplaceholder.typicode.com/posts");
```

```
  return (
    <div>
      {loading ? <p>Loading...</p> : <p>{JSON.stringify(data.slice(0,
5))}</p>}
    </div>
  );
};
```

📌 **Lợi ích của Custom Hooks:**

✅ **Giúp tách biệt logic tái sử dụng**

✅ **Dễ quản lý & bảo trì code**

✅ **Có thể dùng lại ở nhiều components khác nhau**

57. How Can You Optimize React Performance?

✅ **1. React.memo()** – Ngăn component **re-render** không cần thiết.

✅ **2. useMemo()** – Ghi nhớ giá trị tính toán lại **chỉ khi dependencies thay đổi**.

✅ **3. useCallback()** – Tránh tạo lại hàm mới trong mỗi lần render.

✅ **4. Lazy Loading (`React.lazy()`)** – Tải component **chỉ khi cần thiết**.

✅ **5. Code Splitting** – Giảm kích thước bundle bằng `React.lazy()` và `Suspense`.

✅ **6. Virtualization** – Dùng thư viện **React Virtualized** để render **chỉ phần tử hiển thị**.

👉 **Ví dụ tối ưu hiệu suất với `React.memo()`:**

```
import React, { useState, memo } from "react";

const ChildComponent = memo(({ value }) => {
  console.log("Re-rendered!");
  return <h2>{value}</h2>;
```

```
});

const App = () => {
  const [count, setCount] = useState(0);
  return (
    <div>
      <ChildComponent value="Hello World" />
      <button onClick={() => setCount(count + 1)}>Increment
{count}</button>
    </div>
  );
};
export default App;
```

📌 **Lợi ích:**

- 🔹 **`React.memo()`** chỉ re-render khi props thay đổi.
- 🔹 Cải thiện **hiệu suất** trong ứng dụng lớn.

58. What Is the Strict Mode in React?

🔹 **Strict Mode** giúp phát hiện **các lỗi tiềm ẩn** trong React **khi phát triển** (không ảnh hưởng đến production).

- 🔹 Kiểm tra các vấn đề như:

✅ **Sử dụng lifecycle methods cũ (componentWillMount, componentWillReceiveProps, ...)**

✅ **Gọi `useEffect()` không đúng cách**

✅ **Gọi `setState()` không an toàn**

👉 **Cách bật Strict Mode:**

```
import React from "react";
import ReactDOM from "react-dom";

const App = () => <h1>Hello, World!</h1>;

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);
```

📌 **Lợi ích:**

✅ Giúp **debug** dễ hơn trong quá trình phát triển.

✅ Phát hiện lỗi liên quan đến **rendering và lifecycle methods**.

59. What Is Redux and How Does It Work With React?

🔹 **Redux** là một thư viện **quản lý state toàn cục** cho ứng dụng React.

🔹 Giúp **các components truy cập state mà không cần prop drilling**.

🔹 Redux hoạt động dựa trên 3 thành phần chính:

✅ **1. Actions** – Gửi yêu cầu thay đổi state.

✅ **2. Reducers** – Xử lý logic cập nhật state.

✅ **3. Store** – Lưu trữ state toàn cục.

👉 **Cách sử dụng Redux với React:**

1️⃣ **Cài đặt Redux:**

```
npm install redux react-redux
```

2️⃣ Tạo Redux Store:

```
import { createStore } from "redux";

const initialState = { count: 0 };

const counterReducer = (state = initialState, action) => {
  switch (action.type) {
    case "INCREMENT":
      return { count: state.count + 1 };
    case "DECREMENT":
      return { count: state.count - 1 };
    default:
      return state;
  }
};

const store = createStore(counterReducer);
export default store;
```

3️⃣ Tích hợp Redux với React:

```
import { Provider, useSelector, useDispatch } from "react-redux";
import store from "./store";

const Counter = () => {
  const count = useSelector(state => state.count);
  const dispatch = useDispatch();

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => dispatch({ type:
"INCREMENT" })}>Increment</button>
      <button onClick={() => dispatch({ type:
"DECREMENT" })}>Decrement</button>
    </div>
  );
};
```

```
const App = () => (
  <Provider store={store}>
    <Counter />
  </Provider>
);

export default App;
```

📌 **Lợi ích của Redux:**

✅ **Quản lý state toàn cục** dễ dàng.

✅ **Tránh prop drilling**.

✅ **Dễ debug với Redux DevTools**.

60. How Does React Handle Concurrency?

🔹 **React Concurrent Mode** giúp ứng dụng **luôn phản hồi nhanh** dù có nhiều tác vụ phức tạp.

🔹 **Cho phép tạm dừng & ưu tiên rendering**, giúp UI mượt hơn.

👉 **Các tính năng chính của Concurrent Mode:**

✅ **Time Slicing** – Tạm dừng rendering để tránh lag.

✅ **Suspense** – Chờ dữ liệu tải xong rồi render.

✅ **useTransition()** – Điều khiển rendering UI mượt mà hơn.

👉 **Ví dụ sử dụng `useTransition()`:**

```
import { useState, useTransition } from "react";

const App = () => {
  const [input, setInput] = useState("");
```

```
  const [list, setList] = useState([]);
  const [isPending, startTransition] = useTransition();

  const handleChange = (e) => {
    setInput(e.target.value);
    startTransition(() => {
      const newList = Array(20000).fill(e.target.value);
      setList(newList);
    });
  };

  return (
    <div>
      <input type="text" value={input} onChange={handleChange} />
      {isPending ? <p>Loading...</p> : list.map((item, index) => <p
key={index}>{item}</p>)}
    </div>
  );
};
```

📌 **Lợi ích của Concurrent Mode:**

✅ UI **mượt hơn** khi xử lý nhiều dữ liệu.

✅ Hạn chế **UI bị lag khi cập nhật state**.

✅ **Tăng trải nghiệm người dùng** trên web/mobile.

🚀 **Tóm tắt:**

- 🔹 **Custom Hooks** giúp tái sử dụng logic.
- 🔹 **Redux** quản lý state toàn cục.
- 🔹 **Strict Mode** giúp debug dễ hơn.
- 🔹 **React.memo, useMemo, Lazy Loading** tối ưu hiệu suất.

🔹 **Concurrent Mode** giúp UI mượt mà.

61. How Does React Handle Server-Side Rendering (SSR)?

🔹 **Server-Side Rendering (SSR)** là quá trình **render React trên server** và gửi HTML hoàn chỉnh đến trình duyệt.

🔹 **Lợi ích:**

✅ **Tăng tốc độ tải trang ban đầu** (First Contentful Paint - FCP).

✅ **Cải thiện SEO** vì các công cụ tìm kiếm dễ dàng thu thập thông tin.

✅ **Tối ưu hóa hiệu suất trên thiết bị chậm.**

👉 *Cách triển khai SSR với Next.js*

📌 Next.js là framework phổ biến hỗ trợ SSR dễ dàng trong React.

📌 **Cài đặt Next.js:**

```
npx create-next-app my-ssr-app
cd my-ssr-app
npm run dev
```

📌 **Tạo trang sử dụng SSR với `getServerSideProps`:**

```
export async function getServerSideProps() {
  const res = await
fetch("https://jsonplaceholder.typicode.com/posts/1");
  const data = await res.json();

  return { props: { post: data } };
}

const Post = ({ post }) => (
  <div>
```

```
    <h1>{post.title}</h1>
    <p>{post.body}</p>
  </div>
);

export default Post;
```

📌 **Cách hoạt động:**

- `getServerSideProps` **fetch data trên server** trước khi render.
- Next.js **gửi HTML đã render sẵn** đến client.
- **Không cần gọi API trên client**, giúp **cải thiện SEO** và **tăng tốc độ tải trang**.

62. How to Create Forms in React?

- React Forms sử dụng state để lưu dữ liệu nhập vào và cập nhật chúng với onChange.
- Tránh reload trang bằng `preventDefault()`.

👉 Ví dụ Form Đơn Giản trong React

```
import React, { useState } from "react";

function MyForm() {
  const [formData, setFormData] = useState({ name: "", email: "",
message: "" });

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Form Data Submitted:", formData);
  };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <label>Name: <input type="text" name="name"
value={formData.name} onChange={handleChange} /></label>
      <label>Email: <input type="email" name="email"
value={formData.email} onChange={handleChange} /></label>
      <label>Message: <textarea name="message"
value={formData.message} onChange={handleChange} /></label>
      <button type="submit">Submit</button>
    </form>
  );
}

export default MyForm;
```

📌 **Lợi ích:**

✅ **State quản lý dữ liệu form**.

✅ **onChange cập nhật dữ liệu** theo từng input.

✅ **onSubmit xử lý logic khi gửi form**.

63. What Is Lazy Loading in React?

🔹 **Lazy Loading** là kỹ thuật chỉ tải **những component cần thiết**, giúp **giảm thời gian tải ban đầu**.

🔹 **React.lazy()** cho phép **tải component khi cần**.

🔹 **Suspense** hiển thị "Loading..." cho đến khi component tải xong.

👉 *Ví dụ Lazy Loading Component*

```
import React, { Suspense, lazy } from "react";
```

```
const LazyComponent = lazy(() => import("./HeavyComponent"));

function App() {
  return (
    <div>
      <h1>Lazy Loading in React</h1>
      <Suspense fallback={<p>Loading...</p>}>
        <LazyComponent />
      </Suspense>
    </div>
  );
}

export default App;
```

📌 **Cách hoạt động:**

✅ `React.lazy()` chỉ tải HeavyComponent **khi nó được render**.

✅ Suspense hiển thị "Loading..." trong khi chờ tải xong.

🚀 **Lợi ích của Lazy Loading:**

✅ **Cải thiện hiệu suất ứng dụng.**

✅ **Giảm dung lượng tải ban đầu.**

✅ **Chỉ tải component khi cần, giúp tăng tốc trang.**