REACT

▼ **React là gì, ưu điểm, hạn chế?**

- React là một lib, open source đùng để phát triển giao diện người dùng. Dùng cho các app SPA. Nó hữu ích trong việc tạo các UI phức tạp và có thể tái sử dụng tuân theo mô hình component.
- Một số tính năng của react như:
  - Tăng hiệu suất của app bằng việc sử dụng Virtual DOM
  - JSX giúp code dễ đọc và dễ viết
  - Có thể render cả 2 phía client và server
  - Dễ testing hoặc kết hợp với những framework khác
  - Cộng đồng lớn và được hậu thuẫn bởi facebook
- **Ưu điểm**
  - Viết code dễ dàng hơn khi sử dụng JSX có thể nhúng mã HTML CSS và JS
  - Sử dụng component, giúp chia nhỏ ứng dụng thành phần những thành phần nhỏ hơn và có thể tái sử dụng được
  - Hệ sinh thái đa dạng từ app CSR, SSR, app native, hay app desktop với electron, …
- **Nhược điểm:**
  - React không phải là 1 framework hoàn chỉnh mà chỉ là thư viện, phụ thuộc vào cộng đồng và LIB bên ngoài nhiều

▼ **Vòng đời life cycle của REACT**

Có 3 giai đoạn trong vòng đời component React.

- **Mounting**: đề cập đến việc đưa phần tử vào DOM của trình duyệt. Vì React dùng virtual DOM, toàn bộ DOM của trình duyệt đã render sẽ không được làm mới. Bao gồm các phương thức trong giai đoạn này như: `constructor` và `componentDidMount`.

- **Updating**: Trong giai đoạn này, component sẽ được cập nhật khi có thay đổi state hoặc props của component. Các phương thức trong giai đoạn này: `getDerivedStateFromProps`, `shouldComponentUpdate`, `render`, và `componentDidUpdate`.
- **Unmounting**: Ở giai đoạn cuối, component sẽ bị xoá khỏi DOM. Giai đoạn này sẽ có phương thức là `componentWillUnmount`

Ngoài ra còn có nhiều method khác nữa nhưng đây là những method hay dùng nhất trong vòng đời react

---

Các phương thức trong vòng đời:

- `constructor()`: phương thức được gọi khi component được tạo trước khi thực hiện bất kỳ hành động gì. Nó giúp tạo state và props.
- `getDerivedStateFromProps()`: nó sẽ gọi trước khi phần tử được render vào DOM. Nó giúp thiết lập đối tượng state dựa trên props khởi tạo. Phương thức `getDerivedStateFromProps` sẽ có một state như đối số và trả về một đối tượng để thay đổi state. Nó sẽ là phương thức đầu tiên được gọi khi thực hiện cập nhật.
- `render()`: phương thức này sẽ render HTML từ DOM với thay đổi mới nhất. Phương thức `render` sẽ được gọi mỗi khi có thay đổi đến component.
- `componentDidMount()`: phương thức sẽ được gọi sau khi render component. Ta có thể chạy lệnh cần component đã được lưu trong DOM.
- `shouldComponentUpdate()`: trả về giá trị boolean để quyết định xem có render hay không. Mặc định sẽ là True.
- `getSnapshotBeforeUpdate()`: cung cáp truy cập cho props cung như state trước khi cập nhật. Nó dùng cho kiểm tra giá trị trước khi cập nhật.
- `componentDidUpdate()`: được gọi sau khi cập nhật component trong DOM.
- `componentWillUnmount()`: phương thức được gọi khi component bị xoá khỏi DOM.

▼ **Virtual DOM và Real DOM**

- `DOM` là viết tắt của Document Object Model (Mô hình Đối tượng Tài liệu) dùng để truy xuất các tài liệu dạng HTML và XML. DOM đại diện cho một tài liệu như

là một cây cấu trúc dữ liệu. Còn node thì đại diện cho một phần tử trong DOM.

- `Virtual DOM` (VDOM hay DOM ảo) , là cách thể hiện DOM thật của một trang web dưới dạng các Javascript object. Khi thay đổi state của app thì VDOM sẽ được cập nhật lại và so sánh với VDOM cũ (VDOM cũ được đồng bộ hóa với DOM thật trước đó) bằng thuật toán gọi là diffing hay change detection để tìm ra những node cần thay đổi. Cuối cùng nó sẽ cập nhật những node đó trên DOM thật.

▼ **Tại sao cần VirtualDom ?**

- Thao tác DOM là 1 phần không thể thiếu của bất ký app nào. Tuy nhiên thao tác DOM khá chậm so với các thao tác trong JS
- Dẫn đến hiệu năng của app bị ảnh hưởng khi thực hiện thao tác trực tiếp trên DOM
- Trước đó thì những Framework JS sẽ cập nhật toàn bộ lại DOM dù cho ta chỉ có thay đổi 1 hoặc 1 vài thành phần
- React đưa ra khái niệm VD để giải quyết vấn đề đó
- Đối với mỗi đối tượng DOM sẽ có 1 VD tương ứng, nó có các tính giống nhau.
- Sự khác nhau cơ bản là khi có sự thay đổi trên VD nó sẽ không phản ánh trực tiếp lên màn hình.
- React sử dụng 2 VD để hiển thị. Một cái dùng để lưu trữ trạng thái hiện tại và 1 cái là trước đó.
- Khi có sự cập nhật trên VD nó sẽ so sánh 2 bản VD đó để tìm ra node cần thay đổi.
- Và chỉ cập nhật những node đó trên DOM thật, thay vì toàn bộ.

▼ N**guyên tắc *single source of truth* ở trong React là gì ?**

- Thông thường với việc sử dụng HTML + JS thì state hoặc value của thẻ `<input />`
  được điều khiển bằng **browser** chứ không phải là do **JS**.
- Nếu bạn cũng giữ giá trị của đầu vào như vậy trong JS thì nó có nghĩa rằng có ít nhất "two sources of truth - 2 nguồn của sự thật"

- Với controlled component trong React thì **state** và **value** luôn luôn khớp với nhau.
- Bởi vì, React luôn đảm bảo rằng giá trị của element input trong browser bằng với giá trị bạn cung cấp từ JS.

→ Nó chính là "single source of truth".

▼ **Controlled component khác gì uncontrolled component?**

- **Controlled component:**
  - giá trị của phần tử **input** được điều khiển bởi **React**
  - Ta lưu trữ trạng thái của phần tử input trong code, và sử dụng **callback,** với bất kỳ thay đổi nào đến input sẽ được phản ánh tương tự trong code.
  - Khi người dùng nhập dữ liệu vào phần tử input trong controlled component, hàm `onChange` kích hoạt
  - và ta kiểm tra giá trị nhập vào là hợp lệ hay không. Nếu hợp lệ, ta thay đổi trạng thái và re-render phần tử input với giá trị mới.
- **Uncontrolled component:**
  - gái trị của phần tử input được xử lý bởi DOM
  - các phần tử input này hoạt động giống như phần tử input HTML.
  - trạng thái của phần tử input được xử lý bởi DOM.
  - Nên khi giá trị input thay đổi, callback sẽ không được gọi.
  - Hoặc có thể nói là React không thực hiện bất cứ hành động nào khi xảy ra thay đổi.
  - Khi người dùng nhập dữ liệu vào trường input, dữ liệu cập nhật được hiển thị trực tiếp.
  - Để truy cập giá trị phần tử input, ta có thể dùng **ref**.

▼ **state và props trong react là gì ?**

- **State** là đối tượng bên trong 1 component dùng để chứa thông tin hoặc dữ liệu về component. Bất cứ sự thay đổi nào về state trong component cũng dẫn đến

việc re-render. Chỉ được khởi tạo và chỉnh sửa chỉ chính bản thân component chứa nó

- **Props** là đối tượng nhận vào của 1 component, cho phép giao tiếp những component với nhau bằng cách truyền tham số qua lại giữa các component

→ Điểm khác nhau lớn nhất giữa props và state đó là props không thể thay đổi, còn state có thể thay đổi do đó hiệu năng của props tốt hơn state.

▼ **stateless và stateful component là gì ?**

- Stateless component là các component chỉ chứa props, các component loại này chỉ dùng để render() thì sẽ hiệu quả hơn.
- Stateful Component là các component chứa cả props và state, các component này được dùng xử lý data, phản hỏi yêu cầu người dùng, phù hợp cho mô hình client server…

▼ **JSX là gì ?**

- JSX là viết tắt của JavaScript XML.
- Nó cho phép ta viết HTML trong JS và đặt nó vào DOM mà không cần dùng `appendChild()` hay `createElement()`.

▼ **Keys trong react dùng để làm gì**

- Key là một thuộc tính đặc biệt trong element được dùng khi render ra danh sách các phần tử
- Key giúp chúng ta định danh các phần tử trong 1 danh sách, mỗi phần tử là unique trong danh sách
- Nếu không dùng key thì react nó sẽ không hiểu được thứ tự của từng phần từ trong danh sách.

▼ **Sự khác nhau giữa class component và function component?**

- Trước đây, các function component được gọi là stateless component và ít dùng trong react.
- Sau khi hooks ra đời ở những phiên bản sau thì việc sử dụng function component ngày càng nhiều

- Dù function component đang là trend hiện tại, nhưng class component vẫn còn rất quan trọng.
- Một số điểm khác biệt đến từ
  - Cách khai báo: function thay vì class
  - Cách xử lý props, state
  - Cú pháp

▼ **React hook là gì và tại sao cần dùng nó?**

- RH là một tính năng mới của react được giới thiệu ở phiên bản 16.
- Giúp chúng ta viết component bằng function thay vì sử dụng class như các phiên bản trước.
- Nó giúp chúng ta viết code linh hoạt và ngắn gọn hơn
- Nó không thay thế hoàn toàn class, nó chỉ là cách viết component mới, những tính năng của class đều có trên hooks tuy nhiên cú pháp khác nhau mà thôi.

▼ **Tại sao dùng hooks thay vì class ?**

- Trước đây, các function component được gọi là stateless component.
- Chỉ các class component mới được sử dụng cho các phương thức quản lý trạng thái và vòng đời. Nhưng vì class component quá nặng nếu như chỉ cần thay đổi một vài state hay phương thức trong lifecycle.
- Điều đó dẫn đến sự ra đời của React Hooks.

▼ **Hiệu suất của hooks so với class ?**

- React Hooks sẽ tránh được rất nhiều chi phí như tạo thực thể, liên kết các sự kiện, .., có trong các lớp.
- Các hook trong React sẽ dẫn đến các cây component nhỏ hơn vì chúng sẽ tránh được việc lồng nhau tồn tại trong HOC và sẽ render props dẫn đến việc React phải thực hiện ít công việc hơn

▼ **Các quy tắc sử dụng hooks?**

- Chỉ có thể gọi hooks trong function component (không thể dùng trong class).

- Chỉ được gọi hooks ở top level, không được gọi trong 1 loop, condition hay trong 1 nested function

▼ **Giới thiệu một số hooks cơ bản của react**

Trong react hooks nó sẽ gồm 2 loại:

- **Hooks được cung cấp sẵn từ react:**
  - **useState()**: dùng để thiết lập và chỉnh sửa state trong component
  - **useEffect()**: dùng để thực hiện những side effects trên function component
  - **useContext()**: dùng để tạo dữ liệu chung có thể truy cập trong hệ thống phân cấp component mà không cần truyền dữ liệu từ trên xuống dưới.
  - **useReducer()**: dùng khi các logic state của component trở nên phức tạp thì dùng nó sẽ giúp quản lý trở nên dễ dàng hơn. Nó có thể được xem là phiên bản nâng cấp của useState()
  - useMemo(): được sử dụng để tính toán lại giá trị đã ghi nhớ khi có sự thay đổi trong một trong các dependencies, giúp tránh các tính toán tốn kém trên mỗi lần render.
  - useCallback(): giúp tránh một số trường hợp useEffect từ các component con thực thi lại khi nhận callback là một dependencies từ phía component cha. Nó mất 1 vùng nhớ nhất định để ghi nhớ được function mà chúng ta bọc ở trong useCallback.
  - **useRef():** nó sẽ cho phép tạo một tham chiếu đến phần tử DOM trực tiếp trong function component. Ngoài ra nó còn là 1 function trả về object với thuộc tính current được khởi tạo thông qua tham số truyền vào. Object được trả về này có thể mutable và sẽ tồn tại xuyên suốt vòng đời của component.
  - **useLayoutEffect()**: dùng cho đọc bố cục từ DOM và re-render bất đồng bộ
- **Custom hooks**: là một hook đặc biệt do mình tự định nghĩa ra, giúp ta tách biệt logic ra khỏi UI và có thể chia sẻ logic giữa các component.
  - Trong custom hook ta có thể sử dụng lại các hook có sẵn hoặc kết hợp với những custom hook khác.
  - Đặt tên custom hook với prefix là use.

- Custom hooks return data thay vì JSX như component.
- Khi nào cần dùng:
  - Khi một đoạn code (logic) được tái sử dụng nhiều nơi (dễ thấy khi bạn copy cả 1 đoạn code mà không cần sửa gì, trừ tham số truyền vào. Tách như cách mà bạn tách một function).
  - Khi logic quá dài và phức tạp. Bạn muốn viết nó ở 1 file khác, để component của bạn ngắn hơn và dễ đọc hơn vì không cần quan tâm đến logic của hook đó nữa.
- Một số custom hooks như: useAuthentication, useAuthorization, useNotification, useScroll, useFetch, …

▼ **useCallback khác gì useMemo ?**

- useCallback: ghi nhớ 1 function, thường được sử dụng để tránh việc function của component cha gây ra tình trạng re-render của 1 component con
- useMemo: ghi nhớ 1 giá trị, thường được sử dụng để tránh việc thực hiện lại các tính toán phức tạp khi dữ liệu đầu vào không hề thay đổi

▼ **Trường hợp sử dụng useEffect và useLayoutEffect như thế nào?**

- Sự khác nhau giữa **useEffect** và **useLayoutEffect** là thời điểm chúng được gọi. Để hiểu được khi nào chúng được gọi, chúng ta theo dõi các render của DOM.
- Giả sử chúng ta triển khai một hook **useEffect** sau:
  - User tương tác với App. VD: Click vào một button
  - `State` của component sẽ thay đổi
  - DOM sẽ thay đổi
  - UI được thay đổi trên màn hình
  - Hàm `cleanup` sẽ được gọi để `clean` những `effect` đã render trước đó nếu đối số thứ 2 của `useEffect` thay đổi.
  - `useEffect` hook sẽ được gọi
- Đối với **useLayoutEffect:**

- User tương tác với App. VD: Click vào một button
- `State` của component sẽ thay đổi
- DOM sẽ thay đổi
- Hàm `cleanup` sẽ được gọi để `clean` những `effect` đã render trước đó nếu đối số thứ 2 của `useEffect` thay đổi.
- `useLayoutEffect` hook sẽ được gọi
- UI được thay đổi trên màn hình

▼ **Tại sao setState không trả về async**

- **setState**: KHÔNG TRẢ VỀ ASYNC mà nó trả về 1 dispatch function. Vì:
  - Khi gọi useState, kết quả trả về là 1 mảng gồm: 1 giá trị + 1 dispatch function
  - **dispatch function**: nhận vào 1 giá trị và trả về void (lưu ý, là trả về `void`, không phải `promise`) nên `setState` không phải `async`
- **Nếu không phải async thì tại sao nó không thể update giá trị ngay?**
  - Theo như reactjs có nói: Sau khi giá trị được truyền vào, thì nó sẽ đi vào 1 hàng đợi, và chờ được xử lý
  - Đến khi component **re-render** thì giá trị mới sẽ được cập nhật

▼ **Redux, Context API, Hooks có thực sự giống nhau ?**

Thực sự 3 thứ này là khác hoàn toàn, chúng có thể bổ trợ cho nhau nhưng về bản chất thì khác.

- Redux là thư viện để quản lý state và chia sẻ state giữa các component
- Bản thân redux cũng có dựa trên context API
- Về mặt nào đó thì Context API cũng có thể làm phần việc của redux nhưng không phải là tất cả, bởi phải xử lý nhiều mới tối ưu được như thằng redux nếu dùng Context API thô
- Còn đối với hooks thì đó là cách implement mới của react giúp việc functional component thuận tiện hơn.

- Cơ bản thì vẫn có những hook để xử lý local state như useReducer, useState, useRef.
- Ngoài ra còn có những hook để xử lý context như useContext. Vậy nếu muốn dùng context bạn vẫn phải qua Context API và hook chỉ là phương tiện hỗ trợ cho dễ dàng hơn thôi

▼ **Context trong React**

- Context cung cấp phương pháp truyền data xuyên suốt component tree mà không cần phải truyền props một cách thủ công qua từng level. link
- Khi nào nên dùng:
  - Context được thiết kế để chia sẽ data khi chúng được xem là "global data" của toàn bộ ứng dụng React, chẳng hạn như thông tin về user hiện tại đang đăng nhập, theme, hoặc ngôn ngữ mà người dùng đã chọn
  - Sử dụng context, chúng ta có thể tránh được việc truyền props qua các elements trung gian

▼ **React hook có làm việc với static typing?**

- Static typing đề cập đến quá trình kiểm tra code trong suốt thời gian biên dịch để đảm bảo mọi biến đề sẽ được nhập.
- React Hook là hàm được thiết kế để đảm bảo mọi thuộc tính sẽ được nhập tĩnh. Để thực thi nhập tĩnh chặt chẽ hơn trong code, ta có thể sử dụng API React với các Hook tùy chỉnh.

▼ **Làm thế nào để giữ được state trước đó với hooks ?**

- Nếu dùng state khi thay đổi nó sẽ trigger re-render, còn nếu dùng local variable thì nó sẽ bị reset sau mỗi lần re-render
- Ta có thể dùng global variable để giữ state trước đó tuy nhiên thì cách này không khuyến khích dùng.

→ Sử dụng **refs**, hoặc **useRef** của hooks

- Khi giá trị của ref thay đổi, nó không trigger re-render

▼ **So sánh component và Pure component? Pure componet và React.Memo có giống nhau?**

- `React.Component` *cho phép dev override lại* `shouldComponentUpdate` *hook, mặc định hook này reference compare để quyết định re-render lại hay không.*
- `React.PureComponent` *không cho phép dev override lại* `shouldComponentUpdate` *hook, nếu bạn cố tình override thì bạn sẽ ăn ngay warning. Hook này shallow compare để quyết định re-render lại hay không.*
- **PureComponent** giúp chúng ta kiểm tra props và state xem có sự thay đổi về giá trị không để cho phép render lại UI cần thiết. Bản chất PureComponent đã override lại hàm shouldComponentUpdate và kiểm tra giá trị ở props và state để trả về true/false cho việc render UI này.
- **React.memo():** là một HOC, chứ không phải là hooks, tương tự như là PureComponent, chỉ render lại component nếu props có sự thay đổi, sử dụng cơ chế shallow comparison.
- Shallow comparison là chỉ so sánh những giá trị của các thuộc tính ngoài cùng của đối tượng, những thuộc tính lồng nhau và tham chiếu đến đối tượng khác sẽ không so sánh được
- Deep comparison cũng giống như shallow nhưng nó so sánh luôn những giá trị đối tương lồng nhau trong các thuộc tính
- Reference compare thì nó so sánh địa chỉ của đối tượng trong bộ nhớ thay vì so sánh giá trị của đối tượng.

▼ **Các kiểu side effects trong component là gì ?**

- Side effects dùng để:
  - Gọi API lấy dữ liệu
  - Tương tác với DOM
  - Subscriptions
  - setTimeOut, setInterval
- Có 2 loại side effects:
  - Effects **không cần clean up**: gọi API, tương tác với DOM
  - Effects **cần clean up**: subscriptions, setTimeOut, setInterval. Để dọn dẹp bộ nhớ khi unmouting tránh sự cố rò rĩ bộ nhớ hoặc những lỗi không rõ nguyên nhân.

▼ **Prop drilling trong react là gì ?**

- Đôi khi trong react ta cần phải truyền dữ liệu từ component cao hơn xuống sâu component bên dưới. Để truyền được như vậy ta phải truyền qua rất nhiều component trung gian cho đến khi đến component cần nhận props. Đó là prop drilling.
- Tuy nhiên khi app càng lớn prop drilling làm cho việc truy cập dữ liệu hết sức phức tạp

▼ **Strict mode trong react là gì ?**

- StrictMode là công cụ được thêm vào ở React v16.3 để highlight các vấn đề tiềm ẩn trong React. Nó thực hiện kiểm tra bổ sung lên ứng dụng.
- StrictMode giúp giải quyết các vấn đề sau:
  - Khi chúng ta gọi hàm bất đồng bộ tại 1 lifecycle không an toàn. Strictmode sẽ cung cấp cho ta cái cảnh báo về việc sử dụng đó.
  - Cảnh báo khi chúng ta sử dụng findDom() để tìm cây của node trong DOM. Vì phương thức này react không còn hỗ trợ cho nên sẽ đưa ra cảnh báo.
- Nói chung thì nó giúp chúng ta tránh những lỗi tiềm ẩn có thể xảy ra trong quá trình chạy.

▼ **Higher order component trong react là gì ? (HOC)**

- **Higher order function** là một function mà nó nhận vào tham số là function hoặc return về một function.
- **Higher order component** là 1 function và nó nhận vào tham số là 1 component nó sẽ return về một component.

⇒ Khi sử dụng **HOC** thì có 3 điểm bạn cần lưu ý khi sử dụng là:

- Không sử dụng **HOC** trong phương thức render()
- Các phương thức static cần phải được copy lại
- Refs không được truyền qua **HOC**

⇒ Có thể gặp HOC ở:

- **withRouter** của React Route

- hàm **connect** của React-redux

▼ **Các cách khác nhau để chỉnh style component trong react?**

- **Inline Styling:** ta có thể chỉnh style trực tiếp lên phần tử bằng cách dùng thuộc tính style. Nhớ giá trị của style luôn là đối tượng JavaScript
- **Javascript Object**: ta có thể tạo đối tượng JavaScript và tập mô tả thuộc tính style. Các đối tượng có thể dùng như giá trị của thuộc tính style.
- **CSS Stylesheet:** Ta sẽ tạo một file CSS riêng và viết tất cả style cho component trong file đó. Sau đó import nó vào file React.
- **CSS Module:** Tương tự như file CSS, nhưng ta sửa thành `.module.css`, với cách này tên lớp sẽ được mã hoá, đồng thời nó hỗ trợ kiểu viết tương tự **sass**.

▼ **Error boundary là gì ?**

- Được giới thiệu ở React v16, error boundary cung cấp một cách để xử lý lỗi xảy ra trong giai đoạn render.
- Bất kỳ component nào sử dụng các phương thức lifecycle cũng được xem là một error boundary. Các vị trí mà error boundary có thể được phát hiện:
  - Giai đoạn render
  - Trong một phương thức lifecycle
  - Trong constructor
- Khi không dùng error boundary khi có error xảy ra như ở trên ta sẽ thấy một trang trống thay vì lỗi.
- Bất cứ lỗi nào trong phương thức render đều dẫn đến unmounting component.
- Để hiển thị lỗi khi đó, ta sử dụng error boundary. Là một component bọc ngoài các component.

▼ **Làm thế nào để ngăn chặn re-render trong react ?**

- Nguyên nhân của việc gây ra re-render là có sự thay đổi của 1 state hoặc props trên component
- Ta có thể overide lại hook **shouldComponentUpdate()** để ngăn chặn việc re-render

- Hoặc sử dụng một số kỹ thuật như useMemo, useCallBack, …

▼ **Các kỹ thuật tối ưu hiệu suất (optimize performance) trong react là gì ?**

- **useMemo()**
  - Là hook dùng cho caching CPU.
  - Đôi khi trong các ứng dụng web, các hàm đắt (tính toán nhiều, tốn bộ nhớ) được gọi liên túc do re-render dẫn đến tốc độ render chậm, hiệu suất kém.
  - useMemo() có thể sử dụng cho cache cám hàm như vậy. Bằng cách dùng useMemo() các hàm đó chỉ được gọi khi cần thiết.
- **React.PureComponent**
  - Là class component cơ sở để kiểm tra state và props của một component để biết khi nào nó nên được cập nhật.
  - Thay vì dùng React.Component, ta có sử dụng React.PureComponent để giảm việc re-render không cần thiết.
- **Duy trì vị trí state**
  - Đây là quá trình chuyển state đến nơi bạn nhất có thể.
  - Thỉnh thoảng ta có các state không cần thiết nằm trong component cha để gây khó đọc và bảo trì hơn, thậm chí là dẫn đến re-render không cần thiết.
  - Để tốt hơn, ta chuyển các state vô nghĩa ở component cha sang một component riêng biệt.
- **Lazy Loading**
  - Đây là kỹ thuật dùng để giảm thời gian tải của ứng dụng React. Lazy loading giúp tối ưu hiệu suất ứng dụng web bằng cách chỉ tải khi cần thiết.

▼ **Những phương pháp giúp tối ưu performance?**

- **Code splitting**: chỉ load những page hoặc component cần thiết lúc render, không nên load hết tất cả lên, vd: khi vào homepage ta chỉ cần load page home và component liên quan đến page đó thôi
- Lazy load image: thay vì load hết tất cả img thì ta nên load những img hiển thị trên viewport, khi scroll thì tiếp tục load những hình ảnh còn lại

- Lazy size image: với mỗi screen device sẽ có những size ảnh khác nhau thay vì chỉ load 1 size ảnh cho all device
- Server side rendering
- Sử dụng CDN
- Tối ưu CSS
- Minified HTML, CSS, JS with webpack
- Tránh việc re-render nhiều lần trong app
- Thêm loading hoặc skeleton để tăng trải nghiệm người dùng

▼ **React Router là gì ?**

- Là một thư viện dùng để routing trong react. Cho phép điều hướng các trang trong app mà không cần làm mới (reload) lại toàn bộ trang.
- Nó cho phép ta thay đổi URL của trình duyệt nhưng vẫn giữ UI đồng bộ với URL

▼ **withRouter trong react-router-dom là gì?**

- `withRouter()` là một HOC cho phép truy cập thuộc tính đối tượng `history` ứng với `<Route>` gần nhất. Nó sẽ truyền `match`, `location` và `history` như props đến component được bọc bất cứ khi nào nó render.

▼ **Link và NavLink khác nhau gì ?**

- `<Link>` dùng cho điều hướng sang các trang khác nhau trong ứng dụng web, tương tự thẻ a
- `<NavLink>` khá giông link về cách sử dụng nhưng nó hỗ trợ thêm các thuộc tính như **activeClassName** và **activeStyle ,** 2 thuộc tính này giúp cho khi mà nó trùng khớp thì nó sẽ được active lên và chúng ta có thể style cho nó.

▼ **Nested routing là gì ? Khi nào cần dùng ?**

- Sử dụng **nested** route trong React Router giúp dễ dàng tạo các **nested** route trong trang web của chúng ta, giúp dễ dàng hiển thị và quản lý theo **component**.
- Ví dụ: khi cần làm một **sub menu** chúng ta có nhiều menu đa cấp thì ta sử dụng nested route giúp ta phân chia các code thành những component nhỏ bên

trong, giúp ta dễ dàng quản lý và phân chia code

▼ **Setup routing cho mấy trang login như thế nào ?**

- Tạo ra một **custom route extend** từ route thông thường, trong đó ta check xem nếu chưa login thì ta redirect sang trang login, còn login rồi thì thôi
- Hoặc tạo một middleware check, nếu chưa login thì redirect sang trang login

▼ **Handle phần authentication trong app như thế nào ? Cách lưu các token ?**

- **B1**: Check cookies nếu có JWT payload thì vào các trang member nếu không redirect ra trang login
- **B2**: Ở trang login khi user hoàn tất nhập username, pass, ta gửi lên server để thực hiện việc login, nếu thành công thì lấy mã token và lưu vào cookie sau đó redirect về trang home
- **B3**: Nếu trang /login dùng chuẩn xác thực bằng OpenID thông qua một cơ chế OAuth. Theo authorization code grant flow, trang /login sẽ redirect browser về /backend/auth/<provider>. Sau đó nếu flow OAuth xong và hợp lệ (user grant đăng nhập với Facebook), server response sẽ set authentication cookie với JWT bên trong. Sau đó browser sẽ redirect về trang của SPA. SPA sẽ quay lại check như bước 1.

▼ **Bạn thường dùng thư viện nào để quản lý form ?**

- **Formik**
- **Redux-Form**
- **React-Hook-Form**

▼ **Render có điều kiện (condition) trong react ?**

- Giúp ta hiển thị kết quả dynamic dựa vào điều kiện state, hay dữ liệu chúng ta truyền vào.
- Một số cách:
  - Sử dụng if else
  - Toán tử 3 ngôi
  - Sử dụng một biến phần tử

▼ **Cách hiển thị dữ liệu API với axios?**

- Axios là một promise dựa trên HTTP để tạo yêu cầu HTTP đến trình duyệt hay web server.
- **Tính năng**
  - **Interceptors**: Truy cập cấu hình yêu cầu hoặc phản hồi (header, dữ liệu, v.v.) khi chúng gửi đến hoặc đi. Các hàm này có thể hoạt động như các cổng để kiểm tra cấu hình hoặc thêm dữ liệu.
  - **Instances**: Tạo thực thể có thể tái sử dụng như baseUrl, headers, và cấu hình khác đã thiết lập.
  - **Defaults**: Thiết lập giá trị chung cho header chung (như Authorization) với các yêu cầu. Nó hữu ích khi bạn cần xác thực đến server trên mọi yêu cầu.

▼ **Caching trong react?**

- Ta có thể caching dữ liệu trong React bằng nhiều cách như:
  - Local Storage
  - Redux Store
  - Giữ dữ liệu giữa mounting và unmounting

▼ **Có thể dùng đc component mà không extends không ?**

- Có thể được, miễn là không sử dụng **JSX**
- Tuy nhiên sẽ không sử dụng được những lifecycle methods, cũng như các props, state và render

▼ **window.reloaded vs dom.reloaded khác gì nhau ?**

- **window** là gọi khi cả html, js đc load xong
- còn **dom** là khi mới chỉ có html chưa có gì

▼ **Redux là gì ? Thành phần trong redux ? Cách hoạt động ? Nguyên tắc?**

> 💡 *Redux là 1 thư viện giúp chúng ta quản lý các state 1 cách tốt hơn. Thay vì phải truyền state qua từng Component thì Redux sẽ tạo ra 1 store duy nhất dùng để thay đổi dữ liệu.*

- **Đặt điểm:**
  - State trong redux là có thể dự đoán được
  - Redux sử dụng kiến trúc 1 chiều: uni-directional data flow
  - Redux state là READ-ONLY. Muốn update phải dispatch một action (js object)
  - Những thay đổi của redux state được thực hiện bởi Pure functions (reducer)
  - Redux có thể dùng cho các javascript apps, không chỉ riêng gì ReactJS.
- **Thành phần:**
  - `Store` gồm có:
    - `State`: là dữ liệu hiện tại được lưu trên state
    - `Reducer`: là hàm biến đổi state cũ thành state mới
    - Dispatcher: quản lý `middlewares` và chuyển dữ liệu xuống reducer.
  - `Action`: tạo ra các action dùng để mô tả event do người dùng tạo ra
  - `View`: hiển thị dữ liệu được cung cấp bởi Store.
- **Nguyên lý hoạt động của Redux:**
  - **B1**: Khi có 1 sự kiện (event) như là GET, POST, UPDATE, DELETE... thì thằng `action` creators sẽ sinh ra 1 action mô tả những gì đang xảy ra.
  - **B2**: `Action` sẽ thực hiện điều phối `Reducer` xử lý event thông qua hàm `dispatch(action)`.
  - **B3**: `Reducer` dựa vào những mô tả của `Action` để biết cần thực hiện thay đổi gì trên `State` và thực hiện update.
  - **B4**: Khi `State` được update thì các trigger đang theo dõi state đó sẽ nhận được thông tin update và tiến hành render lại phần `view` để hiển thị ra cho người dùng
- **3 Nguyên tắc trong redux:**
  - **Store** luôn là nguồn dữ liệu đúng và tin cậy duy nhất.

    - **State** chỉ được phép đọc, cách duy nhất để thay đổi **State** là phát sinh một Action, và để Reducer thay đổi State.
    - Các function Reducer phải là **Pure function** (với cùng 1 đầu vào chỉ cho ra 1 đầu ra duy nhất)
- **Khi nào cần sử dụng Redux:**
    - Dữ liệu được sử dụng ở nhiều nơi
    - Có hỗ trợ chức năng `undo` / `redo`
    - Cần `cache` dữ liệu để tái sử dụng cho những lần sau.

▼ **Một số middleware trong redux ?**

> 💡 *`middleware` là một lớp nằm giữa `Reducers` và `Dispatch Action`, nó sẽ modify và được gọi trước khi action được dispatch. Thường được dùng trong việc logging, reporting, async api, routing, ...*

- **Logging, Reporting, Redux-saga, Redux-thunk, redux-persist**

▼ **Tại sao phải sử dụng middleware như redux-saga hay redux-thunk?**

Khi sử dụng Redux ta gặp một số ràng buộc như:

- Các xử lý trong `Reducers` phải là các hàm đồng bộ và pure, trả về state mới
- `Reducers` sẽ không được sử dụng các hàm `async` vì không được thay đổi `global state`

⇒ Để giải quyết các side effects cần phải thực hiện ở `middleware`

▼ **Redux saga là gì?**

`Redux-Saga` là một thư viện redux middleware, giúp quản lý những side effect trong ứng dụng redux trở nên đơn giản hơn. Bằng việc sử dụng tối đa tính năng Generators (function*) của ES6, nó cho phép ta viết async code nhìn giống như là synchronous.

> 💡 `Generator function` *là function có khả năng hoãn lại quá trình thực thi mà vẫn giữ nguyên được context. (Nói một cách đơn giản thì generator function là 1 function có khả năng tạm ngưng trước khi hàm kết thúc và có thể tiếp tục chạy tại một thời điểm khác, khác với pure function khi được gọi sẽ thực thi hết các câu lệnh trong hàm)*

▼ Một số helper trong redux-sage ?

- `takeEvery()` : thực thi và trả lại kết quả của mọi actions được gọi.
- `takeLastest()` : có nghĩa là nếu chúng ta thực hiện một loạt các actions, nó sẽ chỉ thực thi và trả lại kết quả của của actions cuối cùng.
- `take()` : tạm dừng cho đến khi nhận được action
- `put()` : dispatch một action.
- `call()`: gọi function. Nếu nó return về một promise, tạm dừng saga cho đến khi promise được giải quyết.
- `race()` : chạy nhiều effect đồng thời, sau đó hủy tất cả nếu một trong số đó kết thúc

▼ Redux thunk là gì?

- Redux Thunk là một Middleware cho phép bạn viết các Action trả về một function thay vì một plain javascript object bằng cách trì hoãn việc đưa action đến reducer.
- Redux Thunk được sử dụng để xử lý các logic bất đồng bộ phức tạp cần truy cập đến Store hoặc đơn giản là việc lấy dữ liệu như từ server

▼ Tại sao lại cần dùng redux toolkit ?

Redux tool kit là một thư viện giúp chúng ta viết redux tốt hơn, dễ hơn và đơn giản hơn (tiêu chuẩn để viết redux).

Ba vấn đề làm nền tảng để RTK ra đời:

- Việc Configure một store trong redux rất phức tạp.
- Phải cài thêm nhiều package để làm việc với redux tốt hơn.

- Redux yêu cầu quá nhiều boilerplate code

▼ **Ngoài redux ra còn có thư viện nào hỗ trợ quản lý state?**

- CÓ
- Như: mobx, zustand, Context API của react, recoil

▼ **Làm thế nào để tạo menu đa cấp bằng đệ quy trong react ?**

▼ **Tree shaking là gì?**

- “Tree shaking" là một tối ưu hóa hiệu suất bắt buộc phải có khi đóng gói JavaScript.
- Nói một cách đơn giản, Tree shaking có nghĩa là xóa code mà không sử dụng đến, hay gọi là code thừa.

▼ **Webpack là gì?**

- Webpack được biết đến là một công cụ phần mềm được sử dụng để quản lý các module JavaScript. Nó sẽ đóng gói tất cả các mã nguồn của chương trình cũng như CSS, font, image,... khi nó hoạt động. Assets chính là tên để gọi những thứ được đóng gói này và chúng sẽ được Webpack đóng gói thành 1 file hoặc một vài file.
- **Tác dụng:** mặc dù đóng gói rất nhiều dữ liệu nhưng chúng được đóng gói một cách rất cẩn thận, bài bản và ngăn nắp, nó được sắp xếp với cấu trúc tương tự như viết mã code. Những dữ liệu này được lập trình sẵn xem cái nào chạy trước, cái nào chạy sau và phần nào sẽ phụ thuộc vào nhau.

▼ **Data binding trong react**

> 💡 Dữ liệu trong React sẽ được truyền theo một chiều duy nhất, đó là từ component cha đến component con, mà không có chiều ngược lại. Việc truyền dữ liệu theo hướng ngược lại được hiểu là "truyền sự kiện". Việc truyền dữ liệu theo 1 hướng duy nhất sẽ giúp ứng dụng hoạt động một cách có kiểm soát hơn

---

- Dữ liệu trong React sẽ chỉ được truyền theo 1 chiều duy nhất, đó là từ `component` cha đến `component` con thông qua `props`. Không có chiều ngược lại

(thực ra là bạn có thể làm ngược lại nhưng như vậy là trái với quan điểm của React).

- Câu hỏi đặt ra, liệu dữ liệu có thực sự được truyền theo 1 chiều duy nhất?
  - Thường là không! Nhưng trong React, chúng ta coi việc truyền "thông tin" từ component con đến component cha là truyền "sự kiện".
  - Trong sự kiện đó component con có thể đính kèm các thông tin của sự kiện (có thể là dữ liệu). Quá trình đó được hiểu là truyền sự kiện, không phải truyền dữ liệu.
- Tóm lại, trong React, dữ liệu sẽ được truyền từ trên xuống, và sự kiện được truyền từ dưới lên.

▼ **Style design (css, scss, styled), The ways struct css module?**

- Có 4 cách để style css trong react:
  - **CSS stylesheet**: đơn giản là viết 1 file css và import vào component bạn muốn style
  - **Inline styling**: với react thì inline style không được thể hiện bằng 1 string mà là 1 object. Ta có thể tạo một biến để lưu trữ những style object và truyền vào element bất kỳ bằng cú pháp style={name_variable}
  - **CSS module**:
    - Là kiểu viết module hóa stylesheet thành từng file nhỏ, không còn sử dụng một file stylesheet tập trung nữa. Thêm vào đó, tất cả tên class lúc này sẽ được scope lại local.
    - Nói tóm lại, Module CSS sẽ được viết ở cùng folder với Component.
    - Một số lợi ích khi dùng css module:
      - Chỉ tồn tại ở một nơi
      - Chỉ được sử dụng ở component đó mà không sử dụng ở bất kì chỗ nào
      - Không nhất thiết phải dùng scss (muốn dùng vẫn được) vì bản chất css module đã chia nhỏ từng file css theo từng component khác nhau

- Không sợ bị trùng tên giữa các class vì khi build với webpack tên class của CSS và element đều là duy nhất với hash code đi kèm.

  > `[Tên component]_[Tên value trong file css]__[hash string]`

- **styled component**:
  - là một lib giúp bạn có thể tổ chức và quản lý code css 1 cách dễ dàng và hiệu quả.
  - Nó được xây dựng với mục tiêu giữ cho các style của component trong react gắn liền với các component đó
  - Không chỉ thay đổi việc implement các component mà còn thay đổi cả tư duy trong việc xây dựng styles cho các component đó.
  - Lợi ích:
    - cho phép ta encapsulate (đóng gói) style vào trong component trong js nhưng vẫn giữ được các tính năng của css như nesting, media query, pseudo-selector, ... Nó giải quyết được vấn đề global scope của css vì ta không cần phải viết selector cho class hay id, bởi styled component sẽ generate class ngẫu nhiên và truyền component thông qua property là className
    - Thay đổi style dựa trên thuộc tính hoặc trạng thái của component dễ dàng hơn. Ta có thể truyền props để thực hiện việc thay đổi style dễ dàng hơn
  - Bất lợi:
    - Tên class được generate ngẫu nhiên nên sẽ gây khó chịu cho người quen debug css bằng tên class. (ta có thể giải quyết bằng việc kết hợp css selector với styled component)
    - Còn khá non trẻ nên chưa được kiểm duyệt tính scale trong các project lớn
    - Nhiều người vẫn không thích css trong js
    - Không được dùng `ref` trên component phải chuyển sang `innerRef` bởi vì ref sẽ được truyền vào wrapper của styled

component thay vì component mình muốn.

▼ **State management (flux, redux)**

- **FLUX** là một kiến thức quen thuộc được thêm bởi Facebook để sử dụng và làm việc với React.
  - https://kipalog.com/posts/Huong-dan-va-giai-thich-Flux-bang-hinh-ve
  - Flux không được xem là một Framework hay thư viện mà nó chỉ đơn giản là một kiểu kiến trúc hỗ trợ thêm cho React.
  - Đồng thời, nó xây dựng các ý tưởng về luồng dữ liệu một chiều (tên tiếng anh là Unidirectional Data Flow).
  - Cấu trúc Flux bao gồm:
    - Actions: Có nhiệm vụ làm dẫn truyền dữ liệu đến với Dispatcher (nó được xem tương tự như Helper Method).
    - Dispatcher: Nhận những thông tin truyền đạt từ Actions để truyền tải dữ liệu tới các nơi đã thực hiện đăng ký nhận các thông tin.
    - Stores: Là nơi có nhiệm vụ lưu trữ cho trạng thái và các logic của hệ thống, đây là một trong những nơi có nhiệm vụ nhận đăng ký dữ liệu với Dispatcher.
    - Controller Views: Được cho là các React Components có nhiệm vụ nhận các trạng thái từ Stores và truyền dữ liệu cho các thành phần con.
- **REDUX:**
- **FLUX ≠ REDUX**
  - Flux có kiến trúc mang tính tổng quát còn redux thì lại chi tiết hơn vì là một phiên bản được implement từ flux và sử dụng immutable state.
  - Mặc dù phát triển dựa trên flux nhưng redux chỉ có duy nhất 1 store và đã lược bỏ đi dispatcher

▼ **SPA, SSA, PWA**

https://haodev.wordpress.com/2019/03/20/ssr-vs-csr/

- **Single Page Aplication**:

- Ứng dụng sẽ render HTML, CSS ở phía client, FE sẽ xử lý nhưng logic cơ bản như như get data, validation, navigate và render. Còn BE sẽ xử lý logic để lấy data và trả về cho client thông qua API.
- Pros:
  - Ít tốn tài nguyên của hệ thống, vì client sẽ chịu trách nhiệm render
  - Vì giao tiếp qua API nên lượng request đến server sẽ được giảm thiểu
  - Nhanh, vì các HTML, css, JS, chỉ được tải 1 lần duy nhất
  - Không cần phải load lại trang, làm tăng trải nghiệm người dùng.
- Cons:
  - Khó SEO vì nội dung web được render phía client
  - Trình duyệt sẽ xử lý nhiều, nên vấn đề hiệu năng cần được chú ý
  -

- **Server Side Rendering**:
  - Mọi logic về validation, đọc dữ liệu, navigate, hay render đều được xử lý ở phía server
  - Cơ chế hoạt động:
    - Khi user vào trang web, browser gửi GET request tới server
    - Server nhận request, đọc dữ liệu, truy vấn database, xử lí logic, ...
    - Server sẽ render ra HTML và trả về cho client
  - Pros:
    - Hỗ trợ mạnh về SEO vì khi bot google, bing vào web sẽ thấy toàn bộ dữ liệu dưới dạng html
    - Initial load nhanh, dễ optimize vì toàn bộ dữ liệu đều đã được xử lý ở phía server, client chỉ render lại.
    - Sẽ rất thích hợp với những static page, có dữ liệu ít bị thay đổi
    - Chỉ cần code trong 1 project ko cần tách biệt ra FE và BE
  - Cons:

        - Web sẽ xử lý và load lại hoàn toàn nếu có một thay đổi nhỏ xảy ra
        - Khi lượng traffic quá lớn, làm cho server nặng và quá tải vì mọi logic đều xử lý phía server
        - Trải nghiệm người dùng không tốt, vì trang web phải refresh và load lại nhiều lần
    - **Progressive web apps**
        - Hiểu đơn giản PWA là cách làm cho web app trở nên ngon hơn, ngon ở đây là khả năng web app chưa làm được.
        - Hiện tại vấn đề lớn nhất mà web app chưa làm được đó là trải nghiệm chưa được mượt mà như native app
        - Để làm được điều này PWA phải đảm bảo được 3 yếu tố:
            - Reliable: app load nhanh và có thể dụng offline
            - Fast: app phải load rất nhanh, nhấn cái chuyển trang liền hoặc animation load vù vù
            - Engaging: có khả năng dụ user sử dụng. Có thể gửi notification, badge
        - Nếu vậy sao không làm native app cho nhanh ?
            - Trên thực tế số lượng người dùng mobile sẽ nhiều hơn web, tuy nhiên thì mỗi user thường chỉ dùng những app top chart và trung bình 1 tháng chỉ cài thêm từ 1 hoặc 2 app.
            - Chi phí để tiếp cận 1 user và dụ user đó dùng app trên web sẽ rẻ hơn trên app (việc chạy quảng cáo trên web sẽ dễ dàng hơn)
            - Về mặt kỹ thuật: đỡ học 2 ngôn ngữ ios và android, có thể tạo bằng RN nhưng PWA sẽ tận dụng src code của web, RN thì không

▼ **Phương pháp SEO**

- Sử dụng thẻ <title />
- Sử dụng thẻ meta description
- Sử dụng những thẻ heading (h1 → h6)

- Broken links: không nên để link của các trang ngừng hoạt động vì ảnh hưởng đến trải nghiệm người dùng cũng như ranking của trang
- Alt attribute image: khi image không thể hiển thị thì alt của image sẽ cung cấp thông tin thay thế
- Sử dung các thẻ HTML5 như header, footer, main, section, nav, ... thay các thẻ div, span truyền thống để trang có ngữ nghĩa hơn cho các search engines
- Loại bỏ inline css trong các thẻ html
- Tối ưu hoá url cho page, url nên chứa keyword liên quan, không nên chứa space hay kí tự đặc biệt

▼ **Immutable và mutable trong react**

- Mutable trạng thái/ dữ liệu của Object có thể thay đổi được.
  - `Object` và `Array` trong JavaScript mặc định đã được *mutate*
  - **Pros**: Mutate Object tạo ra side effect dẫn tới nhiều bugs không mong muốn.
- Immutable trạng thái/ dữ liệu không thể bị thay đổi.
  - Trong JavaScript, tất cả các kiểu dữ liệu nguyên thủy (primitive) đều là immutability.
  - Mỗi khi chúng ta thay đổi dữ liệu, nó sẽ tạo ra một instance mới hoàn toàn và không ảnh hưởng tới instance cũ.
  - **Pros**: Immutability là rất tốt, nó tránh được nhiều bugs nhưng vô hình chung lại làm giảm performance của app. Immutability tạo ra một bản sao giống hoàn toàn so với bản gốc, sau đó edit dữ liệu mà chúng ta muốn thay đổi trên bản sao này. Điều có có nghĩa là nó sẽ tốn rất nhiều memory cho việc copy các `Object`
    hoặc `Array.`Thử tưởng tượng chúng ta muốn thay đổi 1 giá trị trong một Array bao gồm 1 triệu phần tử thì sẽ tốn nhiều memory như thế nào nhỉ 🤔

  ⇒ Để giải quyết được vấn đề memory lead. Immutable sử dụng 1 cấu trúc dữ liệu có lên là **“trie data structures”**. Cấu trúc dữ liệu này sử dụng một concept là **Structure Sharing** (tối ưu memory bằng cách tái sử dụng).

Theo cách Immutability thông thường, mỗi khi thay đổi một thuộc tính nào đó, chúng ta phải clone toàn bộ `Object` hoặc `Array` thành một bản sao, sau đó thực hiện modify trên chính bản sao đó.

Hiện tại có khá nhiều thư viện hỗ trợ chúng ta thực hiện công việc này. Immutablejs và mori là 2 thư viện phổ biến nhất implement immutability sử dụng cấu trúc Structure Sharing.

https://blog.daovanhung.com/post/ban-da-thuc-su-hieu-mutable-va-immutable

---

Link tham khảo:

- **Pattern in react**
- **Câu hỏi hooks thường gặp**
- **Trick interview JS**
- **Một số bài viết hay với react**
- **Một số khái niệm cần biết trong react**
- https://2kvn.com/react-js-nhung-cau-hoi-phong-van-thuong-gap-phan-1-p5f33393635