**REPORT**

useMemo, useCallback, custom hook, Error Bounderies

Học viên: Hà Thị Phương Loan – FE104

1. **useMemo**

* useMemo được dùng để tránh cho việc tính toán lại một function lặp đi lặp lại nhiều lần mỗi lần component re-render.
* useMemo trả về kết quả của một function
* Mỗi lần component rerender nó sẽ kiểm tra giá trị tham số truyền vào function
* Nếu giá trị đó không thay đổi => return value đã caching trong memory.
* Nếu giá trị tham số truyền vào thay đổi => thực hiện tính toán lại vào trả về value, sao đó caching lại value cho những lần rerender tiếp theo.
* Sử dụng useMemo() khi bạn muốn ngăn một số thao tác nặng hoặc tốn kém tài nguyên được gọi trên mỗi lần render.
* Chỉ dùng useMemo khi thất sự phải ngăn chặn gọi lại của một hàm tiêu tốn nhiều tài nguyên hoặc cần nhiều thời gian. Bởi vì useMemo lưu trữ các kết quả của việc thực thi hàm vào bộ nhớ => điều này có thể sẽ lớn dần lên và không may nó lại làm giảm hiệu suất ứng dụng của bạn.

1. **useCallback**

* useCallback được dùng để giữ cho một hàm không được tạo lại lần nữa, dựa trên mảng các phần phụ thuộc.
* useCallback trả về một function (chính là function bạn pass vào ứng với tham số thứ nhất). Callback function này sẽ được tạo lại khi một trong số các dependencies thay đổi. Nếu dependencies không đổi, function trả về sẽ là function trước đó tức là function pass xuống component con không bị tạo mới, tương đương không có object được tạo mới -> component con không bị re-render.
* Sử dụng useCallback() khi mà bạn muốn truyền fuction vào component con và chặn không cho một hàm nào đó tiêu thời gian, tài nguyên phải tạo lại.

1. **custom hook**

* Custom Hooks là những hooks mà do lập trình viên tự định nghĩa với mục đích thực hiện một chức năng nào đó, nó thường được sử dụng để chia sẻ logic giữa các components.
* Để tự định nghĩa một hooks cho riêng mình, chúng ta chỉ cần xây dựng 1 hàm nhận và trả về các giá trị.
* Custom Hooks rất hay được sử dụng trong quá trình triển khai ứng dụng. Trong trường hợp chúng ta muốn tách biệt các phần xử lý logic riêng ra khỏi UI, hay chia sẻ logic giữa các component.
* Khi muốn sử dụng hooks này chỉ cần import nó vào component và gọi như các hooks thông thường. React dựa vào tên để xem đâu là một hooks bởi vậy nên đặt tên đúng định dạng là **use + nameHooks.**

1. **Error Bounderies**

* Error Boundary trong ReactJS cho phép bắt lỗi của tất cả cả các component con, in ra các lỗi đó, và hiển thị ra một fallback UI.
* Error Boundary bắt lỗi trong quá trình render, trong các hàm lifecycle, và trong hàm khởi tạo của tất cả các component ở dưới nó (miễn là trong cùng 1 component tree).
* Chỉ có thể tạo 1 Error Boundary bằng 1 Class component.
* Một Class component trở thành 1 Error Boundary nếu nó định nghĩa một hoặc cả 2 hàm sau:
* static getDerivedStateFromError() => Hàm này dùng để render một fallback UI nếu có lỗi được trả ra.
* componentDidCatch() => Hàm này dùng để bắt và log hoặc hiển thị ra lỗi.
* Sau khi component có cả 2 hàm này, chỉ cần bọc nó quanh một component khác để sử dụng tính năng Error Boundary.
* Error Boundary có thể đặt ở tầng trên cùng của ứng dụng để bắt toàn bộ lỗi, cũng có thể đặt sâu hơn để bắt lỗi cho cụ thể.
* Error Boundary không thể bắt lỗi cho:
* Event Handler
* Các hàm chạy bất đồng bộ (setTimeout hoặc requestAnimationFrame)
* Server side render
* Lỗi được phát ra từ chính nó
* Lỗi của các component ở bên trên nó
* Lỗi của component cách component tree của nó 8000 dặm

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Hết\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*