**React-hook**

1. **Cài đặt:**

Extension Vscode:

- SCSS Formatter:

NodeJS:

- npm i –save-dev node-sass.

- npm i –save query-string.

1. **Host free:**

- freevnn.com.

1. **Hook-useState:**

- Giới thiệu: là một hook cơ bản

+ Giúp mình có thể dùng state trong functonal component.

+ Input: initialState ( giá trị khởi tạo hoặc là một cái callback function ). Chỉ sử dụng cho lần đầu, những lần sau đó bị bỏ rơi.

Để tránh sự dư thừa: dùng initialState dạng callback trong useState( () => retrurn ...code ).

+ Output: một mảng có 2 phần tử tương ứng cho state và setState.

* Những điều cần lưu ý:

+ State đối với class MERGING , còn với hookState là REPLACING.

+ hookState: dùng toán tử 3 chấm để dữ lại các state cũ.

1. **Hook-useContext:**

**UseContext :** Móc diệu ra sử dụng.

**Provider:** Nhà cung cấp chứa **state** cần sử dụng.

**Consumer:** Trong sẽ có 1 **function**. Trả về một component khác.

1. **Hook-useEffect:**

**- Giới thiệu:**

+ Side effect: Những thứ tác động và chạy phía bên ngoài, nằm bên ngoài component của mình.

+ Ví dụ: Gọi API lấy dữ liệu, tương tác với DOM, subscriptions, setTimeout, setInterval.

2 nhóm side effects:

+ Effects không cần clean up : gọi API, tương tác DOM.

+ Effects cần clean up : subscription, setTimeout, setInterval.

**- Giới thiệu về hook-useEffect:**

+ Sử dụng cho side effect.

+ Mỗi hook gồm 2 thành phần: side effect và clean up ( optional )

+ Được thực thi sau mỗi lần render.

+ Được thực thi ít nhất một lần sau lần render đầu tiên.

+ Những lần render sau, chỉ được thực thi nếu có dependencies thay đổi.

+ Effect cleanup sẽ được thực thi trước run effect lần tiếp theo hoặc unmount.

**- Quá trình thực thi:**

**MOUNTING:**

+ rendering

+ run useEffect()

**UPDATING:**

+ rendering

+ run ` useEffect() cleanup ` nếu dependencies thay đổi : lần trước.

+ run ` useEffect() ` nếu dependencies thay đổi : chạy effect lần thứ tiếp theo.

**UNMOUNTING:**

+ run ` useEffect() cleanup `.

**- Dùng useEffect() kèm điều kiện:**

+ Không khai báo dependencies : useEffect() này luôn được chạy sau mỗi lần render.

+ Khai báo dependencies [ ] : useEffect() này nó chạy đúng một lần sau component đó render đầu tiên. 🡪 tương tự như componentDidMount().

UseEffect() cleaup chạy đúng 1 lần khi component đó unmount 🡪 componentWillUnmount().

+ Khai báo dependencies [dữ liệu (filters)]: Lần đầu tiên nó render, những lần tiếp theo có render không thì tùy thuộc vào dữ liệu ( filters ) có bị thay đổi hay không.

**- Ví dụ gọi API:**

Gọi API: lưu ý gọi trong try catch.

1. **Hook-ref**

**- Quy trình :**

+ Tạo ra biến tạm dữ nguyên giá trị sau mỗi lần render.

+ Tạo ra 1 object , object này sẽ được giữ nguyên và không bị thay đổi giữa những lần render từ lúc component này được tạo ra đến lúc mất đi.

**- Sử dụng :**

+ Gán vào giá trị current của nó.

1. **Custom hook**

**- Giới thiệu :**

+ Một hook mình tự định nghĩa ra.

+ Custom hook là một function hơi đặc biệt.

+ Đặt tên custom hook theo dạng: useClock(), useClickOutside(), useQuery(), ...

**- Khi nào dùng custom hooks :**

- Output chả ra tùy bạn quyết định.

+ **function component** return về : JSX.

+ **Custom hook:** return về dữ liệu , về data, object, array, ...

+ Tách riêng phần **logic** ra khỏi **UI**.

+ Chia sẻ **logic** giữa các **component**.

1. **Memoization**

**Giới thiệu về Memoization:**

Tăng tốc độ xử lý máy tính lên, bằng cách là lưu trữ lại dữ liệu, kết quả của những lần tính toán trước đó.

Khi gặp lại bộ **input** đã từng làm thì không tính toán lại, mà trả về kết quả sẵn có.

1. **React.memo( )**

Cũng sử dụng kĩ thuật **Memoization**.

**React.memo( )** Là một HOC, chứ không hooks.

**React.memo( )** tương tự như **PureComponent.**

**PureComponent** thì cho class component.

**React.memo( )** dùng cho function component.

Chỉ **render** lại component nếu **props** thay đổi.

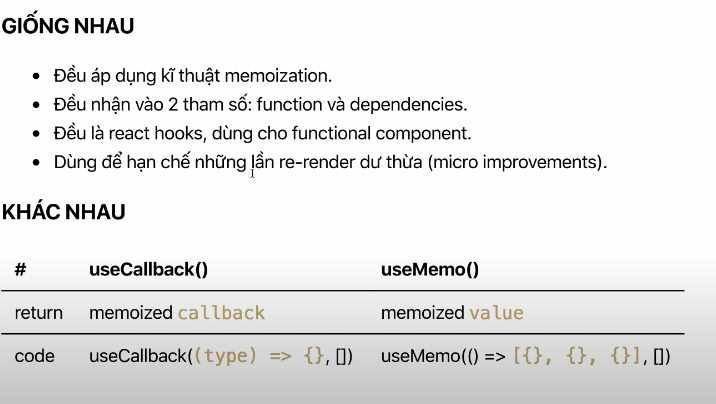
Sử dụng **shallow** **comparion**.: Trong trường hợp như đồ thị, render rất là nặng, tốn thời gian, chart,

1. **useCallback và useMemo**

**useCallback :** là môt react hook giúp mình tạo một memoized callbackvà chỉ tạo ra callback mới khi dependencies thay đổi.

* Nhận vào 2 tham số 1 là function, 2 là dependencies ( một mảng dependencies ).
* Return memoized callback.
* Chỉ tạo ra function mới khi dependencies thay đổi.
* Nếu dùng empty dependencies thì không bao giờ tạo ra function mới. Chỉ tạo ra function lần đầu tiên mà thôi.

**useMemo :** Trả về một giá trị memoized value và chỉ tính toán ra value mới khi dependencies thay đôi.



**Có nên sử dụng useCallback( ) vs useMemo( ) hay không ?**

* Không nên dùng cho tất cả components.
* Nên dùng cho: đồ thị, biểu đồ, animations, những component nặng phần render.

1. **useSelector và useDispatch**

**useSelector() :** gọi **state** từ **store** ra sử dụng.

Gom **state** lại luôn luôn **rerender** thì sử dụng **shallowEqual** trong react-redux. Để nó sử so sánh 2 state bên trong luôn.

**useDispatch : dispatch action** ra sử dụng

  const hobbyList = useSelector((state) => state.hobby.list);

  const dispatch = useDispatch();

  const action = addNewHobby(newHobby);

  dispatch(action);

1. **sas**