2. useEffect

2.1 Định nghĩa và mục đích

- +useEffect() là một hook quan trọng trong React cho phép bạn thực hiện các side effect (tác dụng phụ) trong các component function
- **Side effects** bao gồm:
    - Gọi API (fetch dữ liệu).
    - Thao tác với DOM.
    - Đăng ký/ hủy đăng ký event listeners.
    - Thiết lập timers (`setTimeout`, `setInterval`).
    - Thay đổi state một cách không đồng bộ

+mục đích ***useEffect*** để quản lý vòng đời của của một component và nó phục vụ chúng ta sử dụng trong *function component* thay vì các *lifecycle* như trước đây trong *class component*..

2.2 Cú pháp

+ Cú pháp:

```
useEffect(effectFunction, dependencyArray);
```

***`useEffect` nhận **tối đa hai tham số**:

1. **Tham số thứ nhất (`effectFunction`)**
    a. Là một hàm callback chứa code thực hiện side effect.
    b. **Bắt buộc phải có**.

2.Tham số thứ hai (`dependencyArray`)

- Là một mảng tùy chọn (optional) chứa các giá trị phụ thuộc.
- Quyết định khi nào `effectFunction` được thực thi lại:
    - **Không truyền**: Effect chạy sau **mỗi lần render**.
    - **Mảng rỗng** `[]`: Effect chỉ chạy **một lần sau mount**.
    - **Có dependencies** `[dep1, dep2]`: Effect chạy khi ít nhất một dependency thay đổi.

2.3 Cơ chế hoạt động:

2.3.1.**Nguyên Lý Cơ Bản**

- `useEffect` luôn chạy **sau khi component render xong** (bao gồm cả render lần đầu và các lần re-render sau đó).
- Dependency array kiểm soát việc **có nên chạy lại effect hay không** bằng cách so sánh giá trị các dependencies giữa các lần render.

2.3.2. Các Trường Hợp Dependency Array

a. **Không có dependency array (không truyền mảng)**

Nếu không có dependency array (hoặc nó là undefined), callback function sẽ chạy sau mỗi lần render của component. Điều này bao gồm cả lần render đầu tiên và các lần re-render sau đó.

- Cơ chế: React không theo dõi bất kỳ dependency nào → luôn thực thi effect sau render.

```
useEffect(() => {
  console.log("Chạy sau mọi lần render");
}); // Không có mảng phụ thuộc
```

b, **Dependency array rỗng ([ ])**

Nếu dependency array là một mảng rỗng [], callback function sẽ chạy duy nhất một lần sau lần render đầu tiên (mount). Nó sẽ không chạy lại trong các lần re-render tiếp theo. (tương đương `componentDidMount` trong class component).

- Cơ chế: Vì không có dependencies nào để so sánh, React coi effect là "không bao giờ thay đổi" → chỉ chạy lần đầu.Đây là cơ chế thiết kế có chủ đích của React để tối ưu hiệu suất!

```
useEffect(() => {
  console.log("Chỉ chạy một lần sau mount");
}, []); // Mảng rỗng
```

c, **Dependency array có giá trị (`[dep1, dep2]`)**

1. Cơ Chế Hoạt Động

Khi dependency array chứa các giá trị (state, props, hoặc biến được khai báo trong component):

```
useEffect(() => {
  // 1. Chạy sau lần render đầu tiên (mount)
  // 2. Chạy lại khi bất kỳ dependency nào thay đổi
}, [prop, state]); // Dependency array
```

- **Callback function** trong `useEffect` sẽ:
    - **Chạy sau lần render đầu tiên** (ngay sau khi component mount).
    - **Chạy lại mỗi khi ít nhất một giá trị trong dependency array thay đổi** so với lần render trước.
- React sử dụng `Object.is` (cơ chế so sánh tương tự `===` nhưng chính xác hơn) để kiểm tra sự thay đổi.
  +Primitive values (string/number/boolean): So sánh giá trị.

  ```
  Object.is(5, 5) // true
  ```

  + Reference values (object/array/function): So sánh **tham chiếu**.

  ```
  Object.is([], []) // false (vì khác tham chiếu)
  ```

```
const [count, setCount] = useState(0);
const [text, setText] = useState("");

useEffect(() => {
  console.log("Chạy khi count hoặc text thay đổi");
}, [count, text]); // ✅ Chạy lại nếu `count` hoặc `text` thay đổi
```

2.4 Cleanup function (Hàm dọn dẹp)

1. Bản Chất Của Cleanup Function

Cleanup function là cơ chế **"dọn dẹp trước khi rời đi"** trong React, hoạt động như một bảo hiểm để:

- Ngăn **memory leaks** (rò rỉ bộ nhớ): Khi component bị unmount mà không dọn dẹp các tài nguyên đã tạo
- Tránh **lỗi thực thi trên component đã unmount:**Ngăn chặn việc cập nhật state trên component không còn tồn tại
- Hủy bỏ **các tác vụ không còn cần thiết:**Dừng các hoạt động không còn liên quan khi dependencies thay đổi
- **Duy trì tính nhất quán của ứng dụng**: Đảm bảo không có nhiều phiên bản của cùng một effect chạy đồng thời

Việc dọn dẹp có thể ngăn chặn rò rỉ bộ nhớ (hay còn gọi là memory leaks mà chúng ta thường gặp) và loại bỏ những thứ không mong muốn. Một số trường hợp sử dụng cho điều này là:

- Clean up subscriptions
- Clean up modals
- Remove event listeners
- Clear timeouts.

****Cleanup function được gọi trong các trường hợp sau***

1. **Trước khi component unmount**:
    a. Khi component bị xóa khỏi DOM
    b. Đảm bảo dọn dẹp tất cả tài nguyên trước khi component biến mất
2. **Trước khi effect chạy lại** (khi có dependencies thay đổi):
    a. React sẽ thực thi cleanup function của lần effect trước
    b. Sau đó mới chạy effect mới
    c. Quy trình: Render → Cleanup (nếu có) → Effect mới
3. **Trong Strict Mode (development)**:

a. React có thể mount → unmount → mount lại component để kiểm tra cleanup function
b. Chỉ xảy ra trong môi trường development

*Cú pháp cơ bản

```
import React, { useEffect } from "React";

useEffect(() => {
  // Your effect
  return () => {
    // Cleanup
  };
}, []);
```

Tổng kết:

Trong React, `useEffect` là một Hook cung cấp cách thức để tái hiện chức năng của các phương thức vòng đời truyền thống như `componentDidMount`, `componentDidUpdate` và `componentWillUnmount` trong các component hàm. Nó cho phép bạn quản lý các tác dụng phụ (side effects) trong component của mình, chẳng hạn như lấy dữ liệu, thiết lập các subscription hoặc thao tác với DOM, đồng thời đảm bảo việc dọn dẹp phù hợp khi component unmount hoặc trước các lần re-render tiếp theo.

Chi tiết:

1. **Thay thế các phương thức vòng đời:**
    a. `useEffect` kết hợp chức năng của nhiều phương thức vòng đời vào một Hook duy nhất. Mặc định, `useEffect` chạy sau mỗi lần render, tương tự

như `componentDidUpdate`. Tuy nhiên, bạn có thể sử dụng mảng phụ thuộc (dependency array - đối số thứ hai của `useEffect`) để kiểm soát thời điểm effect chạy, mô phỏng `componentDidMount` (mảng phụ thuộc rỗng) hoặc `componentWillUnmount` (hàm return trong effect).

2. **Tác dụng phụ (Side Effects):**
    a. `useEffect` được thiết kế để xử lý các tác dụng phụ, là các hành động tương tác với thế giới bên ngoài, như lấy dữ liệu, thiết lập trình nghe sự kiện (event listeners) hoặc thay đổi DOM. Những tác dụng phụ này có thể làm gián đoạn quá trình render của React, vì vậy chúng cần được quản lý cẩn thận bằng `useEffect`.
3. **Dọn dẹp (Cleanup):**
    a. Khi component unmount hoặc trước một lần render tiếp theo, bạn có thể chỉ định một hàm dọn dẹp trong `useEffect` để ngăn chặn rò rỉ bộ nhớ (memory leaks) và các vấn đề khác. Hàm này được thực thi trước khi component được render lại hoặc unmount.
4. **Mảng phụ thuộc (Dependency Array):**
    a. Mảng phụ thuộc trong `useEffect` xác định thời điểm hàm effect được chạy lại. Bằng cách chỉ định các phụ thuộc, bạn đảm bảo rằng effect chỉ chạy khi các giá trị trong mảng thay đổi, cho phép cập nhật hiệu quả và tối ưu hóa.
5. **Ưu điểm của việc sử dụng `useEffect`:**
    a. **Ngắn gọn:** `useEffect` cung cấp một cách ngắn gọn và trực quan hơn để xử lý các tác dụng phụ so với các phương thức vòng đời truyền thống trong class component.

b. **Linh hoạt:** Bạn có thể sử dụng nhiều Hook `useEffect` trong một component, mỗi Hook có mảng phụ thuộc riêng, cho phép kiểm soát chi tiết hơn các tác dụng phụ.
c. **Tổ chức mã tốt hơn:** `useEffect` thúc đẩy một codebase có cấu trúc và dễ đọc hơn bằng cách tách biệt các tác dụng phụ khỏi logic chính của component.