TRƯỜNG ĐẠI HỌC THỦY LỢI

KHOA CÔNG NGHỆ THÔNG TIN

**CSE391 - NỀN TẢNG PHÁT TRIỂN WEB**

-----000-----



**BÁO CÁO BÀI KIỂM TRA SỐ 1**

**NHIỆM VỤ 3: LIFECYCLE METHODS & USEEFFECT**

**Giảng viên hướng dẫn:** Thầy Tạ Chí Hiếu

**Lớp:** 65KTPM

**Nhóm:** 10

**Thành viên:** Nguyễn Văn Huỳnh

Nguyễn Văn Trường

Lý Đình Sơn

**Hà Nội**

**03/2025**

# Phân công:

* Lý Đình Sơn: Giải thích nội dung về LifeCycle, Tổng kết – So sánh về UseEffect và LifeCycle
* Nguyễn Văn Trường: Code demo sử dụng LifeCycle, PowerPoint thuyết trình
* Nguyễn Văn Huỳnh: Giải thích về UseEffect, code demo sử dụng UseEffect

# Tổng quan kiến thức và demo dự án

1. **Khởi tạo dự án và các cấu trúc Components trong dự án**

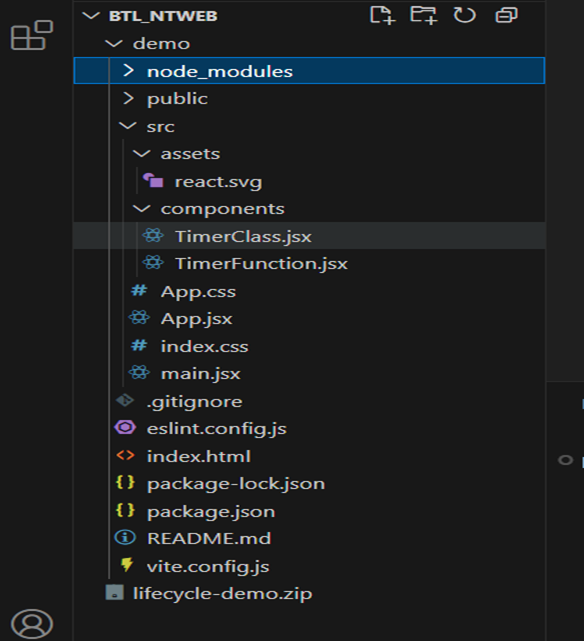


*QR dự án trên Github:*

1. **Tóm tắt kiến thức về React, Virtual DOM, JSX.**

|  |  |
| --- | --- |
| **Khái niệm** | **Mô tả** |
| React | React là một thư viện **front-end** mã nguồn mở, dùng để xây dựng các thành phần UI có thể tái sử dụng (components), đặc biệt phù hợp cho các ứng dụng đơn trang (SPA - Single Page Application). |
| Virtual DOM | Virtual DOM là một bản sao "nhẹ" (dạng JavaScript object) của DOM thật. |
| JSX | JSX (JavaScript XML) là cú pháp mở rộng của JavaScript, cho phép viết HTML ngay trong code React. Giúp code trực quan, dễ đọc hơn so với dùng hàm React.createElement() thuần túy. |

1. **Một vài hình ảnh về dự án:**

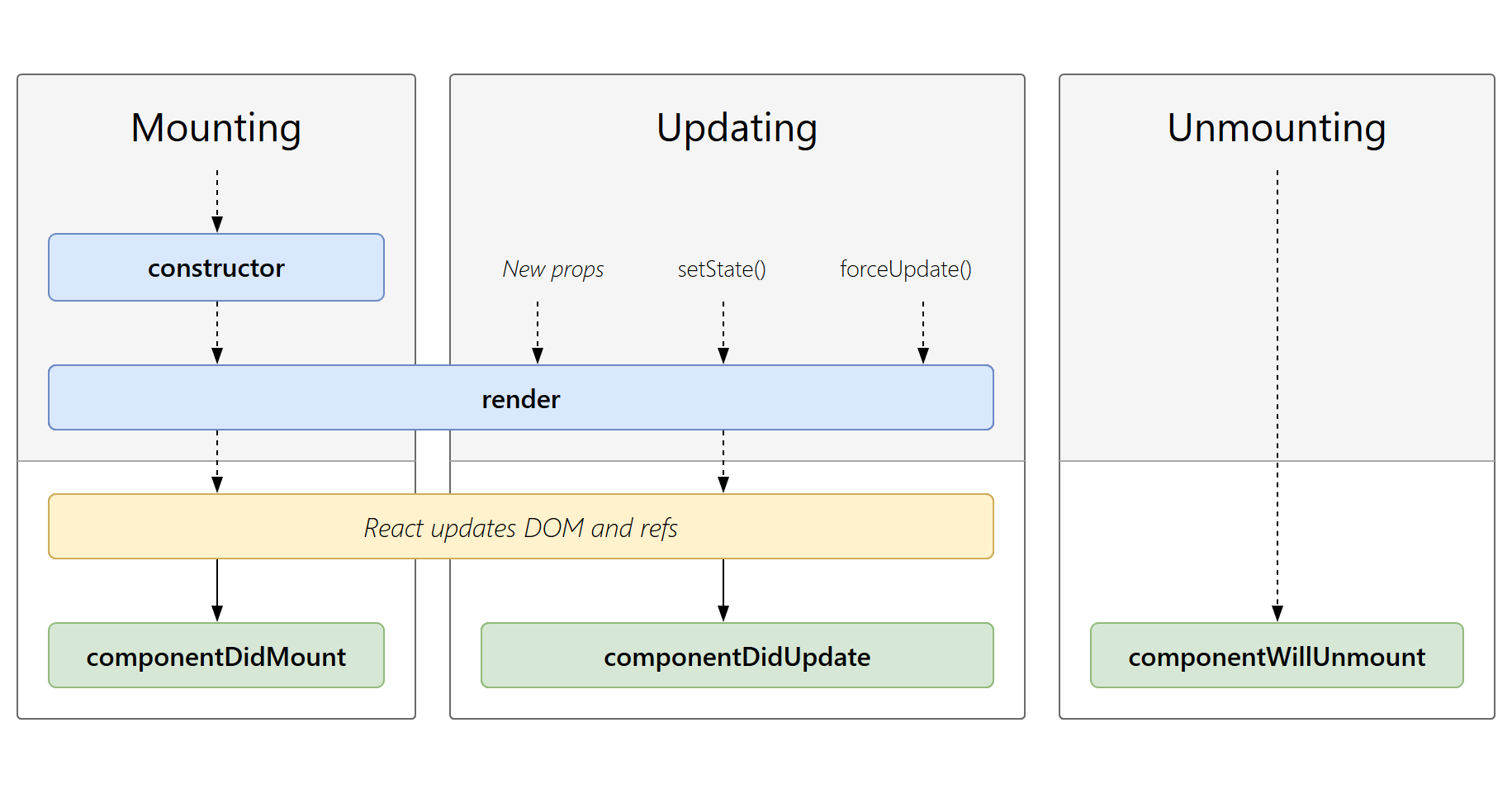


*Khởi tạo thư mục dự án*

*Giao diện dự án*

1. **LifeCycle**
2. **LifeCycle là gì?**

Lifecycle trong React là chuỗi các pha trong quá trình tồn tại của một thành phần React. Các pha này xảy ra khi một thành phần được tạo, cập nhật, hoặc bị xóa khỏi cây DOM.



1. **Mục đích của LifeCycle**

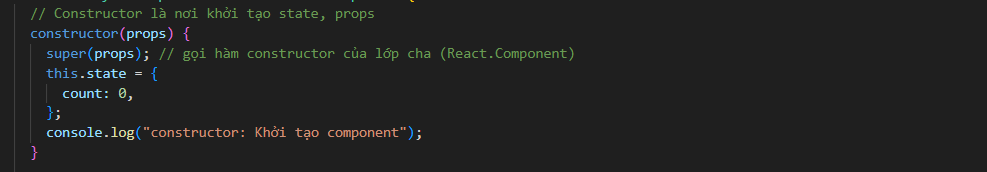
* **Quản lý trạng thái component**: Cho phép thực hiện các tác vụ tại đúng thời điểm trong vòng đời component
* **Tối ưu hóa hiệu suất**: Kiểm soát khi nào component cần được re-render
* **Xử lý side effects**: Thực hiện các tác vụ như API calls, event listeners, timers
* **Dọn dẹp tài nguyên**: Loại bỏ event listeners, hủy subscriptions, clear timers để tránh memory leaks
* **Đồng bộ hóa với external data**: Cập nhật component khi dữ liệu bên ngoài thay đổi

1. **Các pha trong LifeCycle của Component**

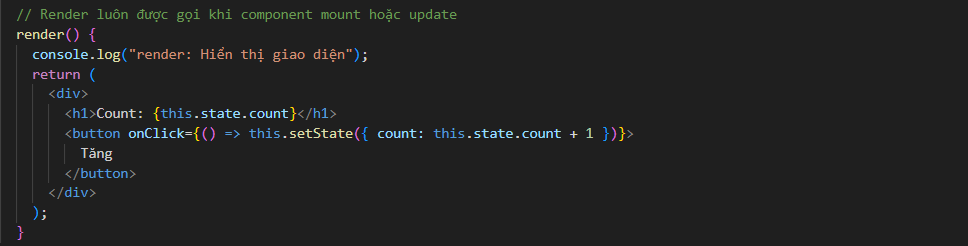
Có ba pha chính trong Lifecycle của một thành phần React:

* Mounting (Gắn kết)
* Updating (Cập nhật)
* Unmounting (Bỏ gắn kết)

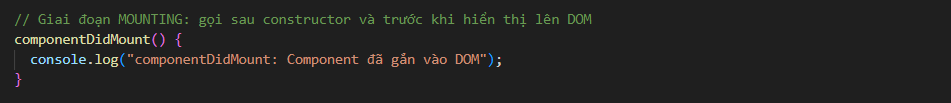
1. **Mounting:** giao đoạn component được tạo ra và chèn vào DOM lần đầu tiên  
   3 phương thức quan trọng trong giai đoạn này bao gồm: Constructor, Render, componentDidMount  
   1. **Thứ tự thực hiện**
      1. Constructor (props)  
           
         **-** Khởi tạo component, dùng để setup state, binding các phương thức  
         - Gọi lần đầu tiên



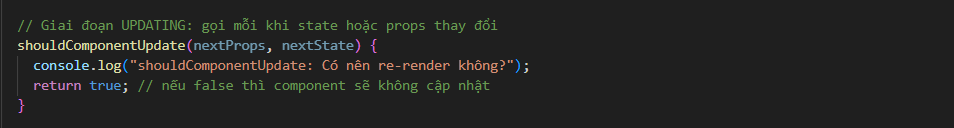
* + 1. Render ()  
         
       - Trả về JSX để hiển thị   
       - Bắt buộc phải có



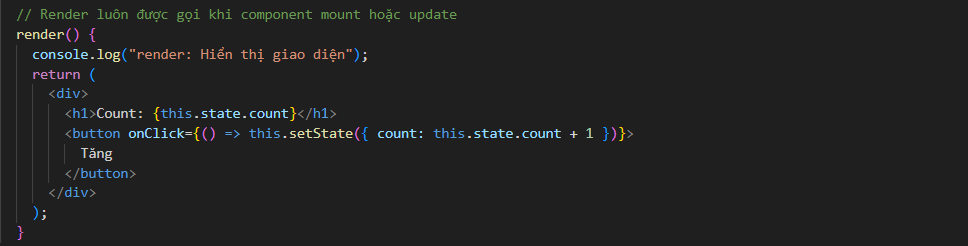
* + 1. componentDidMount ()  
         
       - Gọi ngay sau khi component được render lần đầu và gắn vào DOM  
       - Dùng để gọi API, setup sự kiện, khởi tạo dữ liệu ban đầu



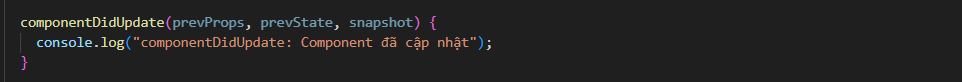
1. **Updating:** Giai đoạn component được re-render do props hoặc state thay đổi.  
   Có 3 phương thức trong giai đoạn này: shouldComponentUpdate, componentDidUpdate, render  
   1. **Thứ tự thực hiện**
      1. shouldComponentUpdate (nextProps, nextState)  
           
         **-** Trả về true hoặc false để quyết định có render lại hay không  
         - Dùng để tối ưu hiệu năng



* + 1. Render ()  
         
       - Gọi lại để cập nhật giao diện



* + 1. componentDidUpdate (prevProps, prevState)  
         
       - Gọi ngay sau khi component render xong  
       - Dùng để xử lý khi có thay đổi ( ví dụ: cập nhật dữ liệu, thao tác DOM sau render)



1. **Unmounting:** Giai đoạn component được loại bỏ khỏi DOM.  
   Phương thức quan trọng trong giai đoạn này: componentWillUnmount  
      
   componentWillUnmount ():  
   - Gọi ngay trước khi component bị gỡ bỏ khỏi DOM.  
   - Dùng để dọn dẹp: clear timer, gỡ sự kiện lắng nghe.



1. **Ưu điểm và nhược điểm của LifeCycle**

**Ưu điểm:**

* Kiểm soát chi tiết vòng đời
* Tách biệt rõ ràng theo giai đoạn
* Dễ dàng tối ưu hóa hiệu suất
* Dễ dàng làm việc với DOM

**Nhược điểm:**

* Code bị chia nhỏ ra thành nhiều phương thức
* Tăng độ phức tạp với state
* Khó tái sử dụng logic giữa các component
* Khó cho người mới học (Có nhiều phương thức với thời điểm chạy khác nhau → dễ nhầm lẫn)

1. **Tổng kết về LifeCycle**

**Lifecycle** trong React là **các giai đoạn mà một component trải qua** từ khi được tạo, hiển thị, cập nhật, cho đến khi bị xóa khỏi DOM.

React chia vòng đời component thành 3 **giai đoạn chính**:

Mounting (gắn vào DOM)

Updating (Cập nhật lại)

Unmounting (Gỡ bỏ khỏi DOM)

**Ngày nay, LifeCycle không còn khuyến khích dùng cho các dự án. Thay vào đó, Hooks được ra đời giúp viết lại side-effect đơn giản hơn với useEffect, useState,… và được khuyến khích thay thế cho lifecycle trong class component.**

**3.UseEffect**

**a,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.*.  
  
**b, Cú pháp**



UseEffect nhận **tối đa hai tham số**:

1. **Tham số thứ nhất (**effectFunction**)**

-Là một hàm callback chứa code thực hiện side effect.

# -Bắt buộc phải có.

1. **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.

**c,Cơ chế hoạt động:**

# **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).

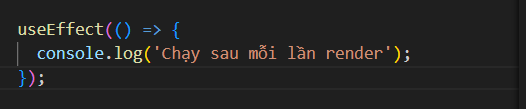
-Dependency array kiểm soát việc **có nên chạy lại effect hay không** bằng cách so sánh giá trị các dependencies giữa các lần render

**Các trường hợp Dependency array**

**1. 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.

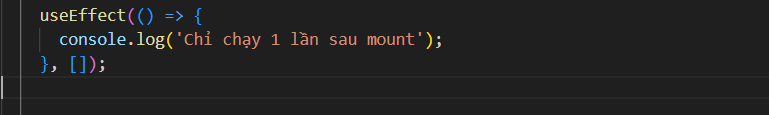
* Cơ chế: React không theo dõi bất kỳ dependency nào → luôn thực thi effect sau render.



# **2. Dependency array rỗng ([])**

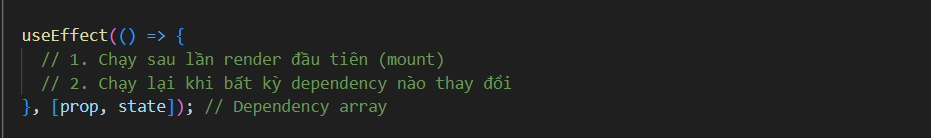
Nếu dependency array là một mảng rỗng [], callback function sẽ chạy duy nhất một lần sau lần render đầu tiên (mount). Nó sẽ không chạy lại trong các lần re-render tiếp theo. (tương đương componentDidMount trong class component).

* 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!



**3. Dependency array có giá trị ([dep1, dep2])**

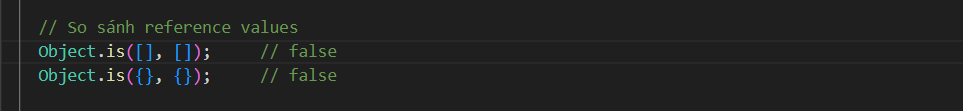
Nếu dependency array chứa các giá trị (state, props, biến được khai báo trong component), callback function sẽ chạy sau lần render đầu tiên và sau đó sẽ chạy lại bất cứ khi nào một trong các giá trị trong mảng dependency thay đổi so với lần render trước



* + 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.

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



**d, 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

#### **2. Các trường hợp sử dụng phổ biến của cleanup function**

- Hủy **subscriptions** (WebSocket, API, RxJS, v.v.)

-Đóng **modals hoặc popups**

- Gỡ bỏ **event listeners** (e.g. window.addEventListener)

- Clear **timeout hoặc interval** (clearTimeout, clearInterval)

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

# 3.1 Trước khi component unmount:

* 1. Khi component bị xóa khỏi DOM
  2. Đảm bảo dọn dẹp tất cả tài nguyên trước khi component biến mất

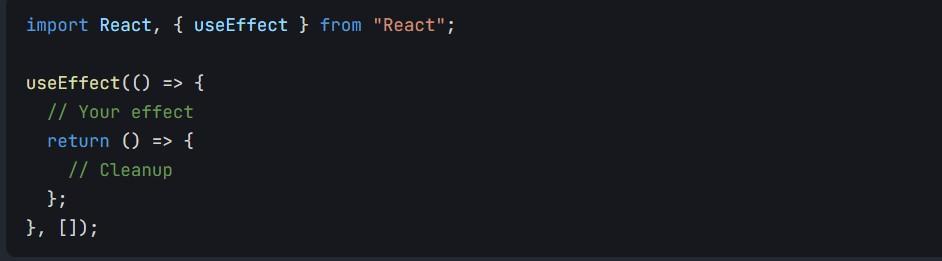
3.2 Trước khi effect chạy lại(khi có dependencies thay đổi):

* 1. React sẽ thực thi cleanup function của lần effect trước
  2. Sau đó mới chạy effect mới
  3. Quy trình: Render → Cleanup (nếu có) → Effect mới

# 3.3 Trong Strict Mode (development):

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

**Cú pháp cơ bản**



**e, Tổng kết về UseEffect:**

1.UseEffect là một Hook quan trọng và linh hoạt trong React, giúp bạn thực thi các tác vụ có tác dụng phụ (side effects) trong function component, thay thế cho các lifecycle methods như componentDidMount, componentDidUpdate, và componentWillUnmount trong class component.

2.UseEffect không chỉ là một công cụ để chạy code bất đồng bộ mà còn là cầu nối giữa React component và thế giới bên ngoài (DOM, network, subscriptions...). Việc hiểu rõ useEffect và cleanup sẽ giúp bạn viết ứng dụng hiệu quả, tối ưu và dễ bảo trì hơn.

**\*\*\*Tổng kết về UseEffect và Lifecycle\*\*\*:**

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:**

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).

# Tác dụng phụ (Side Effects):

* 1. 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.

# Dọn dẹp (Cleanup):

* 1. 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.

# Mảng phụ thuộc (Dependency Array):

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.

1. **Ưu điểm của việc sử dụng** useEffect**:**
   1. **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.
   2. **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.
   3. **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.

**So sánh LifeCycle với useEffect**

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Lifecycle** | **useEffect** |
| Cách dùng | Dùng các phương thức vòng đời như componentDidMount, componentDidUpdate, componentWillUnmount,... () | Dùng hook useEffect() để xử lý mọi giai đoạn của component |
| Loại component áp dụng | |  | | --- | | Chỉ dùng với class Component |      |  | | --- | |  | | Dùng với function component |
| Tổ chức code | |  | | --- | | Phân tách logic theo từng method |      |  | | --- | |  | | Gom logic liên quan vào cùng một hoặc nhiều useEffect |
| Dọn dẹp side effect | Trong componentWillUnmount() | Trong hàm return của useEffect() |
| Tái sử dụng logic | Khó tái sử dụng (nhiều code trùng lặp)   |  | | --- | |  | | Dễ tái sử dụng với custom Hooks |