**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**A blue and white logo

AI-generated content may be incorrect.**

**NỀN TẢNG PHÁT TRIỂN WEB**

***Đề tài: Lifecycle Methods & useEffect***

**Nhóm 8**

**Thành viên nhóm:**

**• Vũ Trí Trường**

**• Phạm Văn Giang**

**• Lê Trọng Huy**

**• Lê Ngọc Tân**

**• Phan Anh**

**Giảng viên phụ trách môn học: Tạ Chí Hiếu**

**Tóm tắt nội dung**

React Class Components có ba phương thức chính để quản lý các side-effects trong từng giai đoạn của vòng đời component:

* **componentDidMount**
* **componentDidUpdate**
* **componentWillUnmount**

**I. Khởi tạo dự án React**

1. **Cài đặt môi trường**
   * Cài Node.js (>= v14) và npm hoặc Yarn.
   * Kiểm tra phiên bản: node -v, npm -v.

**2. Tạo ứng dụng với Create React App** npx create-react-app .

cd time\_counter

npm start

* + Lệnh npm start khởi động development server tại http://localhost:3000.

**3. Tạo ứng dụng với Vite (tùy chọn)** npm create vite@latest my-timer-app --template react

cd time\_counter

npm install

npm run dev

* + Vite nhanh hơn cho dev, server mặc định tại <http://localhost:3000>.

**4. Cấu trúc thư mục cơ bản**

time\_counter/

├── node\_modules/

├── public/

│ └── index.html

├── src/

│ └── components/

│ └── time.js

│ └── Pomodoro.js

│ ├── App.js

│ ├── index.js

├── package.json

└── README.md

* + src/index.js: entry point, render <App /> vào DOM.
  + src/App.js: component chính, import và hiển thị các component con.
  + src/components/time.js: chứa demo time sử dụng lifecycle/hooks.

## 

## **II. Lifecycle Methods trong Class Components**

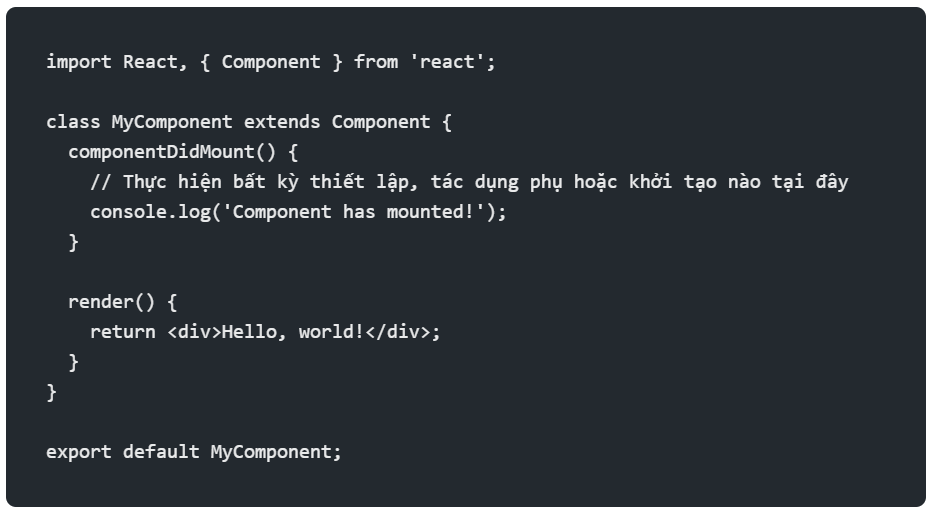
### **1. Thành phầnDidMount()**

componentDidMount là một phương thức vòng đời trong React được gọi ngay sau khi một thành phần được gắn kết (tức là được chèn vào cây DOM). Đây là một trong những cái gọi là "lifecycle hooks" do React cung cấp, cho phép các nhà phát triển chạy mã tại các điểm cụ thể trong vòng đời của một thành phần.

Sau đây là phân tích về những gì xảy ra trong quá trình componentDidMount:

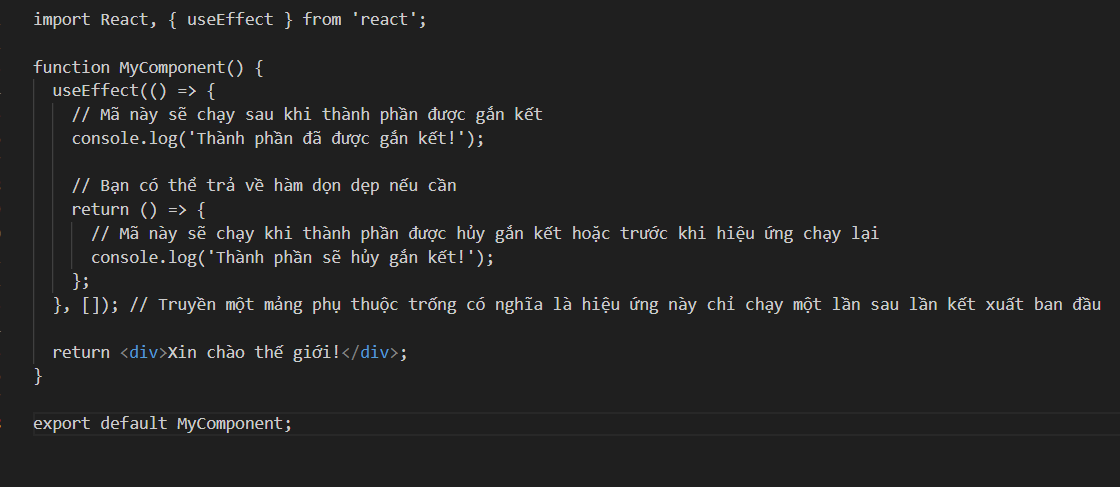
1. **Giai đoạn gắn kết:** Khi một thành phần được kết xuất và chèn vào DOM lần đầu tiên, nó sẽ trải qua một loạt các giai đoạn vòng đời. componentDidMount thuộc về giai đoạn gắn kết.
2. **Chèn DOM:** Tại thời điểm này, JSX của thành phần đã được kết xuất vào DOM và thành phần này hiện là một phần của UI có thể nhìn thấy.
3. **Tác dụng phụ và khởi tạo:** componentDidMount thường được sử dụng để thực hiện các tác vụ yêu cầu tương tác với DOM hoặc các thư viện JavaScript khác, chẳng hạn như khởi tạo các thư viện của bên thứ ba, lấy dữ liệu từ máy chủ, đăng ký sự kiện hoặc thiết lập bộ hẹn giờ.
4. **Hoạt động bất đồng bộ:** Thường khởi tạo các hoạt động bất đồng bộ, như lấy dữ liệu từ API, bên trong componentDidMount. Điều này đảm bảo rằng các hoạt động như vậy không chặn việc hiển thị ban đầu của thành phần.
5. **Cập nhật:** Không giống như một số phương thức vòng đời khác, componentDidMount chỉ được gọi một lần trong toàn bộ vòng đời của một thành phần - cụ thể là sau lần render ban đầu. Nó sẽ không được gọi lại trong các lần render lại tiếp theo do thay đổi trạng thái hoặc props.

Sau đây là một ví dụ đơn giản về cách componentDidMount có thể được sử dụng trong thành phần React:



Trong ví dụ này, phương thức componentDidMount chỉ ghi nhật ký một thông báo vào bảng điều khiển khi thành phần được gắn kết. Bạn có thể thay thế câu lệnh console.log bằng bất kỳ mã khởi tạo hoặc hiệu ứng phụ nào khác mà bạn cần thực hiện khi thành phần được gắn kết.

Bây giờ chúng ta có thể tạo componentDidMount bằng cách sử dụng useEffect() như thế nào.



Trong thành phần chức năng này, chúng ta sử dụng hook useEffect để thực hiện cùng một tác vụ mà componentDidMount thực hiện trong các thành phần lớp. Hook useEffect lấy hai đối số: một hàm biểu diễn hiệu ứng cần thực thi và một mảng phụ thuộc tùy chọn.

* Trong trường hợp này, chúng ta đã truyền một mảng phụ thuộc rỗng [], nghĩa là hiệu ứng sẽ chỉ chạy một lần sau lần kết xuất ban đầu, mô phỏng hành vi của componentDidMount.
* Bên trong hàm hiệu ứng, chúng ta ghi một thông báo vào bảng điều khiển để chỉ ra rằng thành phần đã được gắn kết.
* Ngoài ra, bạn có thể trả về hàm dọn dẹp từ hiệu ứng nếu cần. Hàm dọn dẹp này sẽ chạy khi thành phần được gỡ gắn kết hoặc trước khi hiệu ứng chạy lại (nếu có phụ thuộc).

### **2. componentDidUpdate()**

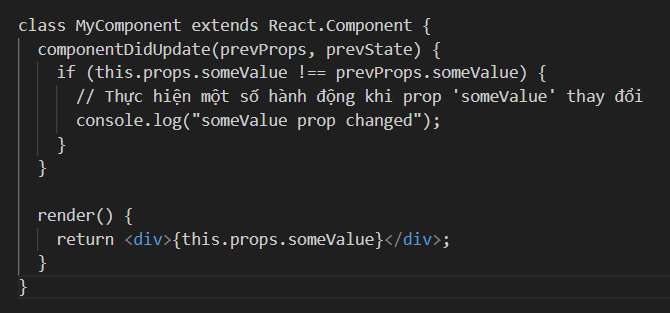
### 

componentDidUpdate là phương thức vòng đời được gọi ngay sau khi cập nhật xảy ra. Nó được gọi mỗi lần thành phần kết xuất lại, ngoại trừ lần kết xuất ban đầu. Phương thức này nhận hai tham số: prevProps và prevState, biểu diễn các prop và trạng thái trước đó của thành phần trước khi cập nhật.

Sau đây là phân tích về cách hoạt động của componentDidUpdate:

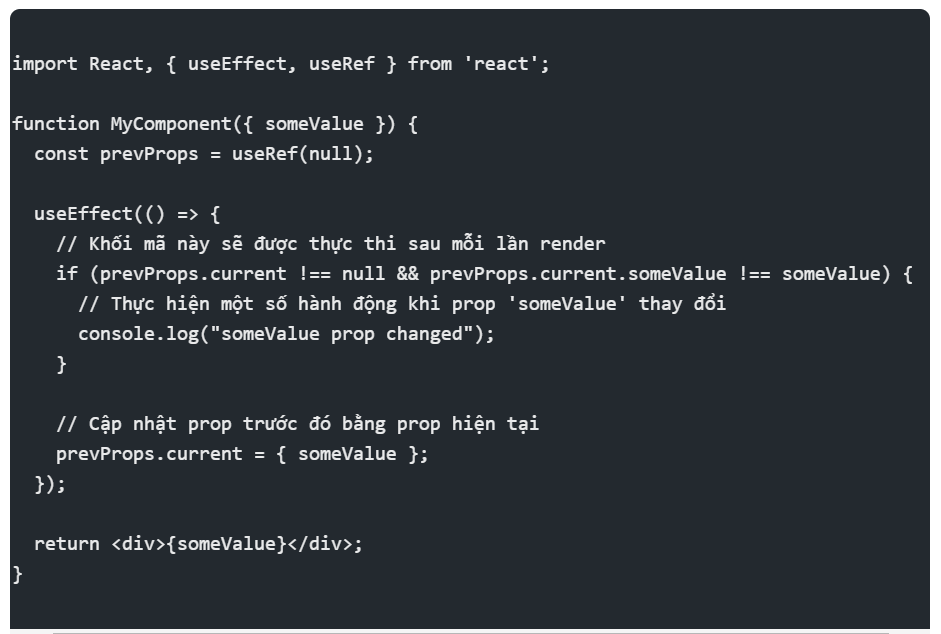
1. **Initial Render:** Khi một thành phần được render lần đầu, componentDidUpdate không được gọi. Chỉ sau khi render ban đầu, bất cứ khi nào có bản cập nhật cho props hoặc trạng thái của thành phần, componentDidUpdate mới được kích hoạt.
2. **Subsequent Renders(***Render tiếp theo***):** Sau lần render đầu tiên, bất cứ khi nào có thay đổi về props hoặc state, React sẽ render lại component. Khi render hoàn tất, componentDidUpdate sẽ được gọi.
3. **Accessing Previous Props and State(***Truy cập Props và State trước đó***):** Bên trong componentDidUpdate, bạn có thể so sánh props và state hiện tại (this.props và this.state) với props và state trước đó (prevProps và prevState). Điều này có thể hữu ích để triển khai logic phụ thuộc vào các thay đổi trong props hoặc state.
4. **Avoid Infinite Loops(***Tránh vòng lặp vô hạn***):** Hãy thận trọng khi sử dụng componentDidUpdate vì việc cập nhật trạng thái trong phương thức này có thể gây ra vòng lặp vô hạn. Đảm bảo bao gồm các điều kiện thích hợp để ngăn chặn việc kết xuất lại không mong muốn.
5. **Side Effects(***Tác dụng phụ***):** componentDidUpdate thường được sử dụng để thực hiện các tác dụng phụ như tạo yêu cầu AJAX, tương tác với DOM hoặc cập nhật trạng thái bên trong của thành phần dựa trên thuộc tính hoặc trạng thái mới.

Sau đây là một ví dụ về cách sử dụng componentDidUpdate:



Trong ví dụ này, componentDidUpdate đang kiểm tra xem props someValue có thay đổi kể từ lần cập nhật cuối cùng không. Nếu có, nó sẽ ghi một thông báo vào bảng điều khiển.

Bây giờ chúng ta có thể tạo componentDidUpdate bằng useEffect() như thế nào.



Trong thành phần chức năng này:

* Chúng tôi sử dụng hook useRef để tạo tham chiếu có thể thay đổi (prevProps) tồn tại qua các lần kết xuất. Tham chiếu này sẽ giữ các prop trước đó.
* Bên trong hook useEffect, chúng ta kiểm tra xem prevProps.current có phải là null không (để tránh chạy khi kết xuất ban đầu) và xem prop someValue có thay đổi kể từ lần kết xuất cuối cùng không.
* Nếu thuộc tính someValue đã thay đổi, chúng ta sẽ thực hiện một số hành động (trong trường hợp này là ghi nhật ký thông báo vào bảng điều khiển).
* Cuối cùng, chúng ta cập nhật prevProps.current bằng các prop hiện tại ở cuối hiệu ứng để sẵn sàng cho lần kết xuất tiếp theo.

Cách tiếp cận này mô phỏng hành vi của componentDidUpdate, cho phép bạn thực hiện các hành động sau mỗi lần kết xuất khi các thuộc tính cụ thể thay đổi.

### **3. componentWillUnmount()**

componentWillUnmount là phương thức vòng đời được gọi ngay trước khi một thành phần được gỡ gắn và hủy. Nó cung cấp cơ hội cho thành phần thực hiện bất kỳ quá trình dọn dẹp cần thiết nào, chẳng hạn như:

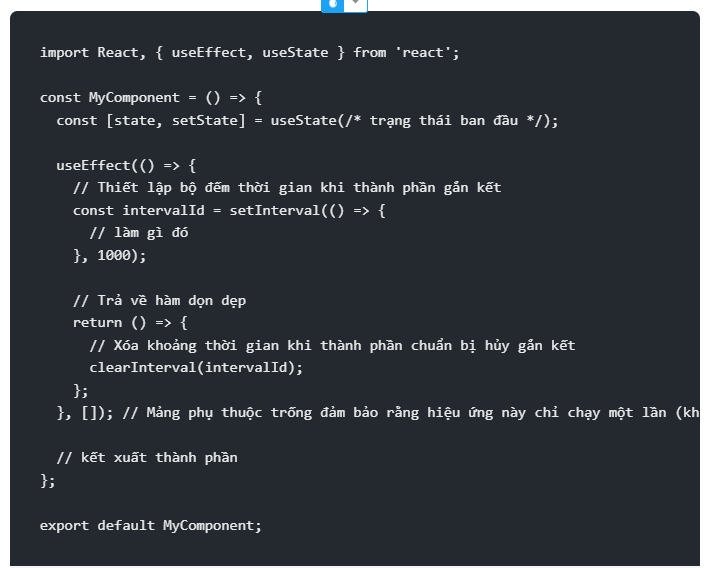
1. Hủy mọi yêu cầu mạng, bộ hẹn giờ hoặc đăng ký mà thành phần đã thiết lập.
2. Xóa trình lắng nghe sự kiện được thêm vào trong componentDidMount hoặc trong vòng đời của thành phần.
3. Xóa mọi khoảng thời gian hoặc thời gian chờ được thiết lập trong suốt vòng đời của thành phần.
4. Thực hiện bất kỳ tác vụ dọn dẹp nào khác để ngăn ngừa rò rỉ bộ nhớ hoặc hành vi không mong muốn.

Sau đây là một ví dụ về cách bạn có thể sử dụng componentWillUnmount trong thành phần React:



Trong ví dụ này, componentWillUnmount được sử dụng để xóa khoảng thời gian được thiết lập trong componentDidMount. Điều này ngăn khoảng thời gian tiếp tục chạy sau khi thành phần đã bị xóa khỏi DOM, tránh rò rỉ bộ nhớ và tính toán không cần thiết.

Bây giờ chúng ta có thể tạo componentWillUnmount bằng cách sử dụng useEffect() như thế nào.



Trong phiên bản thành phần chức năng này, hook useEffect thay thế cả hai phương thức vòng đời componentDidMount và componentWillUnmount. Hiệu ứng được thiết lập để chạy một lần khi thành phần được gắn kết và nó trả về một hàm dọn dẹp sẽ được thực thi khi thành phần được gỡ gắn kết. Hàm dọn dẹp này xóa khoảng thời gian, đảm bảo rằng nó không tiếp tục chạy sau khi thành phần bị xóa khỏi DOM.

**Tóm gọn:**

| **Giai đoạn** | **Phương thức** | **Khi nào gọi** | **Mục đích chính** |
| --- | --- | --- | --- |
| **Mount** | componentDidMount() | Sau lần render đầu tiên, component đã vào DOM | Khởi tạo timer, gọi API, đăng ký sự kiện, thao tác DOM… |
| **Update** | componentDidUpdate(prevProps, prevState) | Mỗi khi state hoặc props thay đổi, sau render | So sánh prevProps/prevState để cập nhật side-effects |
| **Unmount** | componentWillUnmount() | Trước khi component bị gỡ khỏi DOM | Cleanup: clear timer, huỷ subscription, gỡ event listeners |

**Trình tự gọi đầy đủ**

1. Mounting: constructor() → render() → componentDidMount()
2. Updating: shouldComponentUpdate() → render() → getSnapshotBeforeUpdate() → componentDidUpdate()
3. Unmounting: componentWillUnmount()

**III. Cú pháp và cơ chế hoạt động của useEffect với dependency array**

**1. Cú pháp chuẩn của 'useEffect'**

useEffect(() => {

// Side effect: logic cần thực hiện

return () => {

// Cleanup function (tuỳ chọn)

};

}, [dependencies]);

‘useEffect()' là một Hook được gọi sau khi component được render ra giao diện.

Hàm truyền vào 'useEffect' có thể trả về một function dùng để dọn dẹp (cleanup).

Tham số thứ hai '[dependencies]' là mảng phụ thuộc (dependency array) để kiểm soát khi nào effect được thực thi.

**2. Cơ chế hoạt động theo 'dependency array'**

Trường hợp 1: Không có dependency array

useEffect(() => {

console.log("Luôn chạy sau mỗi lần render");

});

> Tương đương gọi sau mỗi lần render → dễ gây tốn tài nguyên nếu không kiểm soát.

Trường hợp 2: Dependency array rỗng '[]'

useEffect(() => {

console.log("Chạy 1 lần duy nhất sau lần render đầu tiên");

}, []);

> Tương đương 'componentDidMount' trong Class Component.

> Thường dùng để gọi API, khởi tạo giá trị, đăng ký event,...

Trường hợp 3: Có giá trị trong dependency array

useEffect(() => {

console.log(`Giá trị count đã thay đổi: ${count}`);

}, [count]);

> Tương đương 'componentDidUpdate'.

>"Effect sẽ chạy lại mỗi khi giá trị trong mảng phụ thuộc (ở đây là count) thay đổi.

**Cleanup Function (Dọn dẹp tài nguyên)**

Khi 'useEffect' trả về một function, hàm này sẽ được gọi trước lần chạy effect kế tiếp hoặc khi component bị huỷ (unmount).

useEffect(() => {

const id = setInterval(() => console.log('tick'), 1000);

return () => {

clearInterval(id); // Dọn dẹp interval

console.log('cleaned up');

};

}, []);

> Tương đương với 'componentWillUnmount'.

## **IV. Ví dụ minh hoạ:**

*Tạo bộ đếm ngược thời gian theo phương pháp Pomodoro*:[link\_git](https://github.com/giangpv102/2025_CSE391_NentangWeb.git)



### **V. Kết luận**

* useEffect là hook mạnh mẽ, thay thế cho nhiều lifecycle methods của class.
* Cung cấp quy trình rõ ràng cho việc thực hiện side-effects và cleanup.
* Là cách tiếp cận hiện đại, ngắn gọn, dễ quản lý hơn trong Function Component.