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:**

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

**Lớp:** 65KTPM

**Nhóm:**

**Trưởng nhóm:**

**Thành viên:**

**Hà Nội**

**05/2025**

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

1. Mounting (Gắn kết)
2. Updating (Cập nhật)
3. Unmounting (Bỏ gắn kết)

2. 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

3. Các phases trong Lifecycle của Component

Mounting: giao đoạn component được tạo ra và chèn vào DOM (DOM ảo) lần đầu tiên  
3 phương thức quan trọng trong giai đoạn này bao gồm:

* Constructor
* Render
* componentDidMount

thứ tự thực hiện:  
**1.** **constructor(props)**

* + Khởi tạo component, dùng để setup state, binding các phương thức.
  + Gọi **đầu tiên**.

1. **render()**
   * Trả về JSX để hiển thị UI.
   * **Bắt buộc** phải có.
2. **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.

Updating: Giai đoạn component được re-render do props hoặc state thay đổi.

3 phương thức quan trọng trong giai đoạn này bao gồm:

* shouldComponentUpdate
* componentDidUpdate
* render

thứ tự thực hiện

 **shouldComponentUpdate(nextProps, nextState)**

* Trả về true hoặc false để quyết định có render lại hay không.
* Dùng để **tối ưu hiệu năng**.

 **render()**

* Gọi lại để cập nhật giao diện.

 **componentDidUpdate(prevProps, prevState)**

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

Unmounting: Giai đoạn component được loại bỏ khỏi DOM.

phương thức quan trọng trong giai đoạn này bao gồm:

* componentWillUnmount ():  
   Gọi **ngay trước khi component bị gỡ bỏ** khỏi DOM.

Dùng để dọn dẹp: clear timer, cancel API request, gỡ sự kiện lắng nghe.

**7. Ưu điểm và Nhược điểm**

**Ưu điểm:**

**Kiểm soát chính xác:**

* Cho phép thực hiện logic tại đúng thời điểm cần thiết
* Tối ưu hóa hiệu suất bằng cách kiểm soát re-render

**Quản lý side effects:**

* Tách biệt logic business khỏi UI rendering
* Đảm bảo cleanup tài nguyên đúng cách

**Debugging dễ dàng:**

* Có thể log và theo dõi trạng thái component
* Dễ debug vì flow rõ ràng

**Nhược điểm:**

**Phức tạp:**

* Nhiều methods cần nhớ và hiểu
* Dễ gây confusion cho developer mới

**Boilerplate code:**

* Cần viết nhiều code cho các tác vụ đơn giản
* Class component verbose hơn functional component

**Khó test:**

* Lifecycle methods gắn chặt với component
* Khó mock và test riêng biệt

**Performance overhead:**

* Một số methods có thể được gọi không cần thiết
* Class component có overhead lớn hơn functional component

**8. Tổng kết về Lifecycle**

**Tầm quan trọng:**

Lifecycle methods là nền tảng để hiểu cách React hoạt động và quản lý component. Chúng cung cấp các hook points để can thiệp vào quá trình render và cập nhật của component.

**Xu hướng hiện tại:**

* **React Hooks** đã thay thế phần lớn lifecycle methods trong functional components
* **useEffect** hook có thể thay thế hầu hết lifecycle methods
* Functional components với hooks được ưa chuộng hơn class components

**So sánh LifeCycle với UseEffect**

So sánh tổng quan

|  |  |
| --- | --- |
| **Class Component Lifecycle** | **Function Component (Hooks)** |
| Dùng các phương thức vòng đời như componentDidMount, componentDidUpdate, componentWillUnmount,... | Dùng useEffect() để xử lý tất cả các vòng đời |
| Cú pháp phức tạp, rườm rà | Gọn gàng, dễ quản lý |
| Khó tái sử dụng logic | Dễ tái sử dụng với custom hook |

So sánh chi tiết

|  |  |  |
| --- | --- | --- |
| **Chức năng** | **Lifecycle** | **useEffect** |
| Khi component được mount | componentDidMount() | useEffect(() => {...}, []) |
| Khi component được update (state/props) | |  | | --- | | componentDidUpdate() |  |  | | --- | |  | | useEffect(() => {...}, [dependencies]) |
| Khi component unmount | |  | | --- | | componentWillUnmount() |  |  | | --- | |  | | useEffect(() => { return () => {...} }, []) |
| Gọi nhiều effect trong cùng component | Phải chia thành nhiều hàm nhỏ hoặc gộp lại | Dùng nhiều useEffect() riêng biệt |
| Dọn dẹp side effect | Trong componentWillUnmount() | Trong hàm return của useEffect() |
| Truy cập props/state hiện tại | |  | | --- | | Truy cập trực tiếp qua this.props / this.state |  |  | | --- | |  | | Truy cập trực tiếp trong scope của useEffect |