Chapter 03: Vòng đời của Component và Hook useEffect trong ReactJS

A. Mục tiêu

Sau bài học này, học viên sẽ:

- Hiểu rõ vòng đời của một component: **Mounting**, **Updating**, và **Unmounting**.
- Thành thạo việc sử dụng hook `useEffect` để quản lý các **side effects**.
- Biết cách thực hiện và dọn dẹp các side effects như gọi API, thao tác DOM, hoặc quản lý subscription.

B. Nội dung lý thuyết

1. Vòng đời của Component (Component Lifecycle)

Một component trong React trải qua ba giai đoạn chính:

- **Mounting**: Component được tạo và chèn vào DOM. Đây là lúc component được khởi tạo và render lần đầu tiên.
- **Updating**: Component được render lại khi có sự thay đổi trong **props** hoặc **state**.
- **Unmounting**: Component bị xóa khỏi DOM, thường khi nó không còn cần thiết trên giao diện.

Trong **Class Components** (chỉ để tham khảo), các phương thức như `componentDidMount`, `componentDidUpdate`, và `componentWillUnmount` được sử dụng để xử lý các giai đoạn này. Tuy nhiên, trong **Functional Components**, hook `useEffect` thay thế hoàn toàn các phương thức trên, giúp mã đơn giản và dễ bảo trì hơn.

2. Side Effects là gì?

Side effects là các hành động trong component ảnh hưởng đến "thế giới bên ngoài" (ngoài việc render JSX). Ví dụ:

- Gọi API để lấy dữ liệu.
- Thay đổi trực tiếp DOM (như cập nhật `document.title`).
- Thiết lập hoặc hủy bỏ các subscription (như `setInterval`, WebSocket).

Vấn đề nếu thực hiện side effects trong body của component:

- Side effects chạy sau mỗi lần render, dẫn đến hành vi không mong muốn (ví dụ: gọi API liên tục).
- Gây ra vấn đề về hiệu năng hoặc lỗi logic (như memory leaks).

Hook `useEffect` được thiết kế để quản lý các side effects một cách an toàn và có kiểm soát.

3. Hook `useEffect` - Quản lý Side Effects

Cú pháp

```
useEffect(() => {
  // Side effect code (setup function)
  return () => {
    // Cleanup function (optional)
  };
}, [dependencyArray]);
```

- `setupFunction`: Hàm chứa logic của side effect, chạy sau khi component render và commit lên DOM.
- `dependencyArray` (mảng phụ thuộc): Quyết định khi nào `useEffect` chạy lại.
- **Cleanup function**: Hàm trả về từ `setupFunction`, chạy trước khi component unmount hoặc trước khi `useEffect` chạy lại.

Các trường hợp sử dụng `dependencyArray`

- **Không có mảng phụ thuộc** (`useEffect(() => {...})`):
  - Effect chạy sau **mỗi lần render**.
  - Ít được sử dụng vì dễ gây ra lặp vô hạn hoặc hiệu năng kém.
- **Mảng rỗng** (`useEffect(() => {...}, [])`):
  - Effect chỉ chạy **một lần** sau khi component được mount.
  - Tương đương `componentDidMount` , lý tưởng cho các tác vụ như gọi API ban đầu.
- **Mảng có giá trị** (`useEffect(() => {...}, [prop, state])`):
  - Effect chạy lần đầu và chỉ chạy lại khi **bất kỳ giá trị nào trong mảng phụ thuộc thay đổi**.
  - Tương đương `componentDidUpdate` cho các giá trị cụ thể.

Hàm dọn dẹp (Cleanup Function)

- Được trả về từ `setupFunction` và chạy trong hai trường hợp:
  1. Trước khi `useEffect` chạy lại (do mảng phụ thuộc thay đổi).
  2. Khi component bị unmount.
- **Tầm quan trọng**: Ngăn chặn **memory leaks** (rò rỉ bộ nhớ) hoặc hành vi không mong muốn.
- Ví dụ: Gỡ bỏ `setInterval` , hủy API request, hoặc xóa event listener.

Ví dụ:

```
useEffect(() => {
  const timer = setInterval(() => console.log('Tick'), 1000);
  return () => clearInterval(timer); // Cleanup
}, []);
```

C. Bài tập thực hành

Bài 1: Tạo Component DataFetcher

- **Yêu cầu**: Sử dụng `useEffect` với mảng phụ thuộc rỗng để gọi API `https://jsonplaceholder.typicode.com/users` khi component được mount. Lưu dữ liệu vào state và hiển thị danh sách người dùng.
- **Hướng dẫn**:
  1. Tạo `src/components/DataFetcher.js`:

```
import { useState, useEffect } from 'react';

function DataFetcher() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(response => response.json())
      .then(data => setUsers(data));
  }, []);

  return (
    <div>
      <h2>Danh sách người dùng</h2>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name} - {user.email}</li>
        ))}
      </ul>
    </div>
  );
}
export default DataFetcher;
```

2. Sử dụng trong `App.jsx`:

```
import DataFetcher from './components/DataFetcher';

function App() {
  return <DataFetcher />;
```

```
}
export default App;
```

Bài 2: Tạo Component DocumentTitleChanger

- **Yêu cầu**: Tạo một ô input với state `text`. Sử dụng `useEffect` để cập nhật `document.title` khi `text` thay đổi.
- **Hướng dẫn**:

1. Tạo `src/components/DocumentTitleChanger.js`:

```
import { useState, useEffect } from 'react';

function DocumentTitleChanger() {
  const [text, setText] = useState('');

  useEffect(() => {
    document.title = text || 'React App';
  }, [text]);

  return (
    <input
      type="text"
      value={text}
      onChange={(e) => setText(e.target.value)}
      placeholder="Nhập tiêu đề trang"
    />
  );
}
export default DocumentTitleChanger;
```

2. Sử dụng trong `App.jsx`:

```
import DocumentTitleChanger from './components/DocumentTitleChanger';
```

```
function App() {
  return <DocumentTitleChanger />;
}
export default App;
```

Bài 3: Tạo Component Timer

- **Yêu cầu**: Sử dụng `useEffect` để tạo một `setInterval` đếm giây tăng dần. Triển khai cleanup function để `clearInterval` khi component unmount. Thêm nút để ẩn/hiện component và kiểm tra cleanup qua console.
- **Hướng dẫn**:

1. Tạo `src/components/Timer.js`:

```
import { useState, useEffect } from 'react';

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(prev => prev + 1);
    }, 1000);
    return () => {
      console.log('Cleaning up interval');
      clearInterval(interval);
    };
  }, []);

  return <p>Thời gian: {seconds} giây</p>;
}
export default Timer;
```

2. Sử dụng trong `App.jsx` để kiểm tra cleanup:

```jsx
import { useState } from 'react';
import Timer from './components/Timer';

function App() {
  const [showTimer, setShowTimer] = useState(true);

  return (
    <div>
      <button onClick={() => setShowTimer(!showTimer)}>
        {showTimer ? 'Ẩn Timer' : 'Hiện Timer'}
      </button>
      {showTimer && <Timer />}
    </div>
  );
}
export default App;
```

Bài 4: Tạo Component WindowWidthLogger

- **Yêu cầu**: Lắng nghe sự kiện `resize` của `window` để hiển thị chiều rộng hiện tại. Sử dụng `useEffect` để thêm và gỡ bỏ event listener.
- **Hướng dẫn**:

1. Tạo `src/components/WindowWidthLogger.js`:

```jsx
import { useState, useEffect } from 'react';

function WindowWidthLogger() {
  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {
    function handleResize() {
      setWidth(window.innerWidth);
    }
    window.addEventListener('resize', handleResize);
    return () => {
      console.log('Cleaning up resize listener');
```

```
      window.removeEventListener('resize', handleResize);
    };
  }, []);

  return <p>Chiều rộng cửa sổ: {width}px</p>;
}
export default WindowWidthLogger;
```

2. Sử dụng trong `App.jsx`:

```
import WindowWidthLogger from './components/WindowWidthLogger';

function App() {
  return <WindowWidthLogger />;
}
export default App;
```

Bài 5: Ứng dụng tìm kiếm người dùng (Nâng cao)

- **Yêu cầu**: Kết hợp các bài tập trước để tạo ứng dụng tìm kiếm người dùng. Nhập `userId`, gọi API `https://jsonplaceholder.typicode.com/users/{userId}` khi `userId` thay đổi, hiển thị thông tin người dùng, trạng thái loading, và xử lý lỗi.
- **Hướng dẫn**:

1. Tạo `src/App.jsx`:

```
import { useState, useEffect } from 'react';

function App() {
  const [userId, setUserId] = useState('');
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  useEffect(() => {
```

```
    if (!userId) {
      setUser(null);
      setError(null);
      return;
    }

    setLoading(true);
    fetch(`https://jsonplaceholder.typicode.com/users/${userId}`)
      .then(response => {
        if (!response.ok) throw new Error('Không tìm thấy người dùng');
        return response.json();
      })
      .then(data => {
        setUser(data);
        setError(null);
      })
      .catch(err => {
        setUser(null);
        setError(err.message);
      })
      .finally(() => setLoading(false));
  }, [userId]);

  return (
    <div>
      <h1>Tìm kiếm người dùng</h1>
      <input
        type="text"
        value={userId}
        onChange={(e) => setUserId(e.target.value)}
        placeholder="Nhập ID người dùng"
      />
      {loading && <p>Đang tải...</p>}
      {error && <p>Lỗi: {error}</p>}
      {user && (
        <div>
          <h2>{user.name}</h2>
          <p>Email: {user.email}</p>
          <p>Địa chỉ: {user.address.street}, {user.address.city}</p>
        </div>
      )}
    </div>
  );
```

```
}
export default App;
```