**Sử dụng useEffect in React**

by traltb@fe.edu.vn

useEffect là một Hook rất quan trọng trong React, nó cho phép bạn thực hiện các "side effect" trong functional component. Trong lập trình React, "side effect" là bất kỳ tác vụ nào xảy ra bên ngoài phạm vi của việc render UI, ví dụ như:

* Gọi API để lấy dữ liệu
* Thao tác trực tiếp với DOM (Document Object Model)
* Thiết lập timer (setTimeout, setInterval)
* Đăng ký hoặc hủy sự kiện **subscriptions**
* Ghi log

**Tại sao cần useEffect?**

* **Tách biệt logic:** useEffect giúp tách biệt logic side effect khỏi logic render UI, làm cho code rõ ràng và dễ bảo trì hơn.
* **Quản lý vòng đời component:** useEffect cho phép bạn kiểm soát khi nào side effect được thực hiện (ví dụ: sau khi component được render, khi props thay đổi, trước khi component bị unmount).
* **Tránh re-render không cần thiết:** Bằng cách sử dụng mảng dependencies, bạn có thể kiểm soát khi nào useEffect chạy lại, tránh re-render không cần thiết và tối ưu hiệu suất.

**Cấu trúc của useEffect:**

useEffect(() => {

  // Logic side effect ở đây

}, [dependencies]);

* **Hàm callback:** Chứa logic side effect cần thực hiện.
* **Mảng dependencies:** Một mảng chứa các giá trị mà effect phụ thuộc vào. Khi một giá trị trong mảng thay đổi, effect sẽ chạy lại.

A black background with white text

AI-generated content may be incorrect.

3 Cách sử dụng useEffect

1. **useEffect(callback)**

Cách này sẽ thực thi hàm callback sau mỗi lần component được render lại. Nó tương đương với việc kết hợp componentDidMount và componentDidUpdate trong class component. Khi không có mảng dependencies, useEffect sẽ chạy sau mỗi lần render của component.

**Ví dụ 1:** Sử dụng useEffect mà không có dependencies:

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

function App() {

  const [count, setCount] = useState(0);

  useEffect(() => {

    document.title = `Bạn đã click ${count} lần`;

    console.log('re-render');

  }); // Hàm này sẽ chạy sau mỗi lần render

  return (

    <div>

      {console.log('render')}

      <p>Bạn đã click {count} lần</p>

      <button onClick={() => setCount(count + 1)}>Click me</button>

    </div>

  );

}

export default App;

**Giải thích**:

* useEffect được gọi sau mỗi lần component render, và ở đây nó thay đổi tiêu đề trang mỗi khi giá trị count thay đổi. Mỗi lần nhấn nút, component sẽ render lại, và useEffect sẽ được thực thi.

**2. useEffect(callback, [])**

Cách này chỉ thực thi callback một lần duy nhất sau khi component được render lần đầu tiên. Nó tương đương với componentDidMount trong class component. Mảng rỗng [] đảm bảo useEffect chỉ chạy một lần khi component được mount.

**Ví dụ 2:** Gọi API khi component mount:

import React, { useState, useEffect } from "react";

function App() {

  const [data, setData] = useState(null);

  useEffect(() => {

    fetch("https://jsonplaceholder.typicode.com/posts")

      .then((response) => response.json())

      .then((data) => setData(data));

  }, []); // Mảng rỗng [] đảm bảo useEffect chỉ chạy một lần

  return (

    <div>

      {data ? (

        data.map((post, index) => <div key={index}>{post.title}</div>)

      ) : (

        <p>Loading...</p>

      )}

    </div>

  );

}

export default App;

**Giải thích**:

* Hàm useEffect ở đây thực hiện một lần duy nhất khi component được mount. Dữ liệu được fetch từ API và sau đó được hiển thị trong component. Mảng rỗng [] trong useEffect đảm bảo rằng hiệu ứng chỉ được thực thi một lần sau khi component được mount lần đầu tiên.

**3. useEffect(callback, [deps])**

* Cách này sẽ thực thi callback khi component render lần đầu tiên và mỗi khi một trong các giá trị trong mảng deps thay đổi. useEffect chỉ chạy lại nếu các giá trị trong mảng dependencies thay đổi.

**Ví dụ 3:** Theo dõi sự thay đổi của count và inputValue:

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

function App() {

  const [count, setCount] = useState(0);

  const [inputValue, setInputValue] = useState('');

  useEffect(() => {

    console.log('Count đã thay đổi:', count);

  }, [count]); // Chạy lại khi count thay đổi

  useEffect(() => {

    console.log('Input value đã thay đổi:', inputValue);

  }, [inputValue]); // Chạy lại khi inputValue thay đổi

  return (

    <div>

      <p>Bạn đã click {count} lần</p>

      <button onClick={() => setCount(count + 1)}>Click me</button>

      <input

        type="text"

        value={inputValue}

        onChange={e => setInputValue(e.target.value)}

      />

    </div>

  );

}

export default App;

**Giải thích**:

* Hai hàm useEffect được sử dụng ở đây để theo dõi sự thay đổi của hai biến count và inputValue. Mỗi khi một trong hai giá trị này thay đổi, hiệu ứng sẽ được gọi lại và ghi lại sự thay đổi vào console.

**Tổng kết về useEffect:**

* **useEffect** cho phép bạn thực hiện các tác vụ phụ (side effects) trong component mà không làm ảnh hưởng đến luồng render chính. Điều này có thể bao gồm việc gọi API, thay đổi DOM, hay thiết lập các subscription.
* **Không có mảng dependencies ([])**: useEffect sẽ chạy sau mỗi lần render.
* **Mảng dependencies rỗng ([])**: useEffect chỉ chạy một lần sau lần render đầu tiên (tương tự componentDidMount).
* **Mảng dependencies chứa các giá trị cụ thể**: useEffect sẽ chạy lại mỗi khi một trong các giá trị trong mảng dependencies thay đổi (tương tự componentDidUpdate).

**Exercise 13:**

**Exercise 1: Data Fetching (UserPosts)**

**Bước 1:** Tạo dự án

npx create-react-app exercise-useeffect

cd exercise-useeffect

npm start

**Bước 2:** Tạo component UserPosts trong thư mục /src/components/UserPosts.js

* **Tạo state**: Dùng hook useState để lưu trữ danh sách bài viết.

const [posts, setPosts] = useState([]);: Dùng để khai báo một biến trạng thái posts, bắt đầu với một mảng rỗng. setPosts là hàm để cập nhật giá trị của posts.

* **Lấy dữ liệu từ API**: Sử dụng useEffect để fetch dữ liệu khi component được mount hoặc khi userId thay đổi.
  + useEffect(() => {...}, [userId]);: Hook này sẽ chạy mỗi khi userId thay đổi. Trong useEffect, hàm fetchData sẽ là một hàm async (hàm bất đồng bộ) dùng để gọi API và lấy dữ liệu từ URL
  + Dữ liệu từ API sẽ được trả về dưới dạng JSON và được lưu vào state posts bằng hàm setPosts(data).

**Lợi ích của việc sử dụng async/await:**

* **async/await** giúp mã nguồn dễ đọc và dễ quản lý hơn so với việc sử dụng các promise chain (.then().catch()).
* await giúp bạn đợi kết quả từ fetch API trước khi tiếp tục các bước khác trong hàm, giúp tránh các vấn đề đồng bộ (synchronization issues).
* **Render dữ liệu**: Hiển thị danh sách các bài viết bằng map.
  + posts.map: Duyệt qua mảng posts và render các bài viết với id, title và body của mỗi bài viết.

import React, { useState, useEffect } from "react";

const UserPosts = ({ userId }) => {

  // Bước 1: Khởi tạo state cho posts

  const [posts, setPosts] = useState([]);

  // Bước 2: Lấy dữ liệu khi component mount hoặc userId thay đổi

  useEffect(() => {

    const fetchData = async () => {

      try {

        // Gọi API JSONPlaceholder để lấy danh sách bài viết

        const response = await fetch(

          `https://jsonplaceholder.typicode.com/posts?userId=${userId}`

        );

        if (!response.ok) {

          throw new Error("Lỗi khi lấy dữ liệu");

        }

        const data = await response.json(); // Chuyển dữ liệu thành JSON

        setPosts(data); // Cập nhật state posts với dữ liệu lấy được

      } catch (error) {

        console.error("Có lỗi xảy ra:", error);

      }

    };

    fetchData(); // Gọi hàm fetchData

  }, [userId]); // useEffect sẽ được gọi lại khi userId thay đổi

  // Bước 3: Render danh sách các bài viết

  return (

    <div>

      {posts.length > 0 ? (

        posts.map((post) => (

          <div key={post.id}>

            <h3>{post.title}</h3>

            <p>{post.body}</p>

          </div>

        ))

      ) : (

        <p>Đang tải dữ liệu...</p>

      )}

    </div>

  );

};

export default UserPosts;

**Bước 3: Sử dụng component UserPosts trong ứng dụng**

import React, { useState } from "react";

import UserPosts from "./components/UserPosts";

const App = () => {

  const [userId, setUserId] = useState(1);

  return (

    <div>

      <h1>Danh sách bài viết của người dùng {userId}</h1>

      <UserPosts userId={userId} />

      <button onClick={() => setUserId(userId + 1)}>

        Xem bài viết của người dùng khác

      </button>

    </div>

  );

};

export default App;

**Exercise 4: Form validation**

**useEffect giúp chúng ta theo dõi sự thay đổi của giá trị và thực hiện xác thực khi cần thiết.**

**Bước 1:** Cài đặt react bootstrap

npm install react-bootstrap bootstrap

**Bước 2**: import CSS của bootstrap vào file src/index.js hoặc src/App.js:

import 'bootstrap/dist/css/bootstrap.min.css';

**Bước 3**: Tạo component ValidatedInput

Chúng ta sẽ tạo một component ValidatedInput để thực hiện xác thực và hiển thị thông báo lỗi.

**Bước 3.1: Khởi tạo state**

Sử dụng useState để tạo các state value (lưu trữ giá trị đầu vào) và isValid (theo dõi tính hợp lệ của giá trị đầu vào).

* value: Lưu trữ giá trị đầu vào từ người dùng.
* isValid: Theo dõi tính hợp lệ của giá trị đầu vào (true nếu hợp lệ, false nếu không hợp lệ).
* errorMessage: Lưu thông báo lỗi khi giá trị đầu vào không hợp lệ.

const [value, setValue] = useState(""); // State lưu trữ giá trị đầu vào

const [isValid, setIsValid] = useState(true); // State theo dõi tính hợp lệ của đầu vào

const [errorMessage, setErrorMessage] = useState(""); // State lưu thông báo lỗi

**Bước 3.2: Xác thực đầu vào với useEffect**

Sử dụng useEffect để thực hiện việc xác thực mỗi khi giá trị đầu vào thay đổi. Hàm xác thực sẽ được gọi trong useEffect và cập nhật isValid dựa trên kết quả xác thực.

  // useEffect để thực hiện xác thực mỗi khi giá trị đầu vào thay đổi

  useEffect(() => {

    const isValidInput = validateInput(value);

    setIsValid(isValidInput); // Cập nhật tính hợp lệ

    if (!isValidInput) {

      setErrorMessage("Giá trị phải có ít nhất 5 ký tự!"); // Cập nhật thông báo lỗi nếu không hợp lệ

    } else {

      setErrorMessage(""); // Xóa thông báo lỗi nếu hợp lệ

    }

  }, [value]); // useEffect sẽ chạy lại mỗi khi value thay đổi

* Mỗi khi value thay đổi, useEffect sẽ được gọi lại và thực hiện xác thực. Nếu giá trị không hợp lệ (dưới 5 ký tự), chúng ta sẽ cập nhật isValid thành false và hiển thị thông báo lỗi.
* Nếu giá trị hợp lệ, isValid sẽ được cập nhật thành true và thông báo lỗi sẽ được xóa.

**Bước 3: Tạo trường nhập liệu và hiển thị lỗi**

* Sử dụng Form.Control của React Bootstrap để tạo trường nhập liệu. Khi người dùng nhập dữ liệu, onChange sẽ cập nhật giá trị trong state value.
* isValid và isInvalid được sử dụng để điều chỉnh trạng thái hiển thị của trường nhập liệu (bao gồm màu sắc khi có lỗi).
* Form.Control.Feedback sẽ hiển thị thông báo lỗi nếu isValid là false.

Code hoàn chỉnh:

import React, { useState, useEffect } from "react";

import { Form, Button } from "react-bootstrap";

// Hàm xác thực đầu vào (ví dụ: kiểm tra độ dài tối thiểu)

const validateInput = (value) => {

  return value.length >= 5; // Giả sử giá trị phải có ít nhất 5 ký tự

};

function ValidatedInput() {

const [value, setValue] = useState(""); // State lưu trữ giá trị đầu vào

  const [isValid, setIsValid] = useState(true); // State theo dõi tính hợp lệ của đầu vào

  const [errorMessage, setErrorMessage] = useState(""); // State lưu thông báo lỗi

  // useEffect để thực hiện xác thực mỗi khi giá trị đầu vào thay đổi

  useEffect(() => {

    const isValidInput = validateInput(value);

    setIsValid(isValidInput); // Cập nhật tính hợp lệ

    if (!isValidInput) {

      setErrorMessage("Giá trị phải có ít nhất 5 ký tự!"); // Cập nhật thông báo lỗi nếu không hợp lệ

    } else {

      setErrorMessage(""); // Xóa thông báo lỗi nếu hợp lệ

    }

  }, [value]); // useEffect sẽ chạy lại mỗi khi value thay đổi

  return (

    <Form>

      <Form.Group controlId="validatedInput">

        <Form.Label>Nhập một giá trị</Form.Label>

        <Form.Control

          type="text"

          value={value}

          onChange={(e) => setValue(e.target.value)} // Cập nhật giá trị khi người dùng thay đổi

          isValid={isValid} // Hiển thị trạng thái hợp lệ

          isInvalid={!isValid} // Hiển thị trạng thái không hợp lệ

        />

        <Form.Control.Feedback type="invalid">

          {errorMessage} {/\* Hiển thị thông báo lỗi nếu không hợp lệ \*/}

        </Form.Control.Feedback>

      </Form.Group>

      <Button variant="primary" type="submit" disabled={!isValid}>

        Gửi

      </Button>

    </Form>

  );

}

export default ValidatedInput;

**Exercise 5:**

Sử dụng useEffect và React Bootstrap cho việc xác thực đầu vào của một form, trong đó người dùng phải nhập một email hợp lệ và mật khẩu có ít nhất 8 ký tự, chi tiết như sau:

* Xác thực email (phải là một địa chỉ email hợp lệ).
* Xác thực mật khẩu (phải có ít nhất 8 ký tự).
* Cung cấp thông báo lỗi khi các trường không hợp lệ. Lỗi chỉ sẽ hiển thị màu đỏ sau khi người dùng nhập và nhập sai, thay vì hiển thị lỗi ngay khi trường nhập liệu còn trống.
* Nút "Submit" sẽ chỉ được kích hoạt khi tất cả các trường đầu vào đều hợp lệ.
* Disable nút "Sumit" khi các trường đầu vào không hợp lệ.

**A white background with red lines

AI-generated content may be incorrect.**

**See:** [**https://mailtrap.io/blog/validate-emails-in-react/**](https://mailtrap.io/blog/validate-emails-in-react/)

**Exercise 6: Yêu cầu tạo form như sau:**

Form này sẽ thực hiện xác thực cho các trường nhập liệu và chỉ cho phép người dùng gửi form khi tất cả các trường hợp hợp lệ.

Yêu cầu:

* Textbox: Kiểm tra xem người dùng đã nhập một tên hợp lệ.
* Radio Button: Kiểm tra xem người dùng đã chọn một trong các lựa chọn.
* Dropdown (Select): Kiểm tra xem người dùng đã chọn một giá trị từ danh sách.
* Checkbox: Kiểm tra xem người dùng đã đánh dấu checkbox đồng ý với điều khoản.
* Button: Button chỉ được phép nhấn khi tất cả các trường đều hợp lệ.

**A white background with green lines

AI-generated content may be incorrect.**