|  |  |
| --- | --- |
| logo | **TRƯỜNG ĐẠI HỌC THUỶ LỢI**  **KHOA CÔNG NGHỆ THÔNG TIN**  NỀN TẢNG WEB |

Nhiệm vụ 4: Tìm hiểu useState & Các Hook Cơ bản trong React

Thành viên nhóm và công việc

|  |  |  |
| --- | --- | --- |
| Thành viên | Mã sinh viên | Công việc |
| Lại Vương Linh Anh | 2151264639 | Tìm hiểu về nội dung nhiệm vụ  Làm docs  Làm slide  Demo |
| Nguyễn Hữu Trường Huy | 2151264662 | Thuyết trình |
| Mai Nam Khánh | 2151261306 | Tìm hiểu nội dung nhiệm vụ |
| Trần Văn Nam | 2351260677 | Tìm hiểu nội dung nhiệm vụ |

GVHD: Th.S. Tạ Chí Hiếu

1. Tổng quan

* Trong React, các hook là các hàm đặc biệt cho phép sử dụng các tính năng như trạng thái (state) và side effects trong các component functional mà không cần viết class. Các hook useState, useRef, và useLayoutEffect là những công cụ cốt lõi để xây dựng các ứng dụng React tương tác và hiệu quả.
* ‘useState’: Cho phép component functional lưu trữ và quản lý trạng thái, như giá trị của một trường nhập liệu hoặc trạng thái hiển thị của một thành phần giao diện. Đây là hook cơ bản nhất để xử lý dữ liệu thay đổi theo thời gian.
* ‘useRef’: Cung cấp một cách để lưu trữ tham chiếu đến phần tử DOM hoặc các giá trị có thể thay đổi mà không gây ra việc re-render component. Rất hữu ích khi cần truy cập trực tiếp DOM hoặc lưu trữ giá trị tạm thời.
* ‘useLayoutEffect’: Tương tự useEffect, nhưng chạy đồng bộ sau khi tất cả các thay đổi DOM được áp dụng, trước khi trình duyệt vẽ lại giao diện. Hook này phù hợp cho các tác vụ cần đo lường hoặc điều chỉnh layout ngay lập tức.

Hiểu và sử dụng thành thạo các hook này giúp các nhà phát triển xây dựng các giao diện người dùng động, phản hồi nhanh, và tối ưu hóa hiệu suất ứng dụng.

1. Chi tiết các hook

a. useState Hook

* Cú pháp: ‘const [state, setState] = useState(initialState);’
* Tham số:
* ‘initialState’: Giá trị ban đầu của trạng thái. Có thể là bất kỳ kiểu dữ liệu nào (số, chuỗi, đối tượng, mảng, v.v.). Nếu là một hàm, hàm này sẽ được gọi một lần để tính toán giá trị ban đầu, hữu ích cho các phép tính tốn kém.
* Cách sử dụng:
* Gọi ‘useState’ ở mức top-level của component hoặc trong custom hooks, không được gọi trong vòng lặp, điều kiện, hoặc hàm lồng nhau.
* Trả về một mảng chứa hai phần tử: giá trị trạng thái hiện tại (state) và hàm cập nhật trạng thái (setState).
* Hàm ‘setState’ có thể nhận giá trị mới trực tiếp hoặc một hàm updater (nhận trạng thái trước đó và trả về trạng thái mới).
* Best Practices:
* Sử dụng hàm updater khi cập nhật trạng thái dựa trên trạng thái trước đó, đặc biệt khi có nhiều cập nhật trong cùng một sự kiện.

Ví dụ: ‘setCount(prevCount => prevCount + 1)’.

* Không mutate trực tiếp các đối tượng hoặc mảng trong trạng thái; thay vào đó, tạo một bản sao mới.

Ví dụ: ‘setForm({ ...form, firstName: value })’.

* Sử dụng hàm initializer cho trạng thái ban đầu tốn kém:

‘useState(() => computeExpensiveValue())’.

* Để reset trạng thái, thay đổi key của component:

‘<Form key={version} />’.

* Lỗi thường gặp:
* Cập nhật trạng thái là bất đồng bộ, nên giá trị trạng thái không thay đổi ngay sau khi gọi ‘setState’.
* Nếu giá trị trạng thái mới giống hệt giá trị cũ (so sánh bằng ‘Object.is’), React có thể không re-render.
* Gọi ‘setState’ không điều kiện trong quá trình render có thể gây lỗi "Too many re-renders".

Ví dụ: ‘setCount(count + 1)’ trong thân component thay vì trong event handler.

* Trong Strict Mode, các hàm initializer hoặc updater có thể chạy hai lần để kiểm tra tính thuần khiết (pure), nhưng điều này chỉ xảy ra trong môi trường phát triển.

b. useRef Hook

* Cú pháp: ‘const ref = useRef(initialValue);’
* Tham số:
* ‘initialValue’: Giá trị ban đầu của ref, thường là ‘null’ khi dùng cho DOM refs.
* Cách sử dụng:
* Sử dụng để lưu trữ tham chiếu đến phần tử DOM:

‘<div ref={ref} />’.

* Sử dụng để lưu trữ giá trị có thể thay đổi mà không gây re-render:

‘ref.current = someValue’.

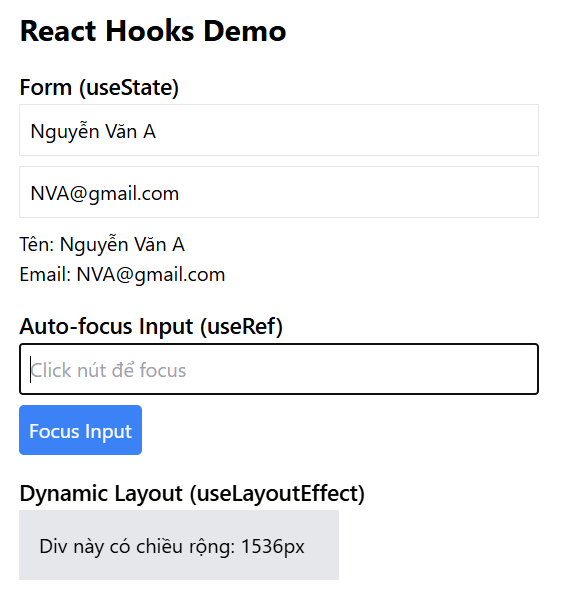
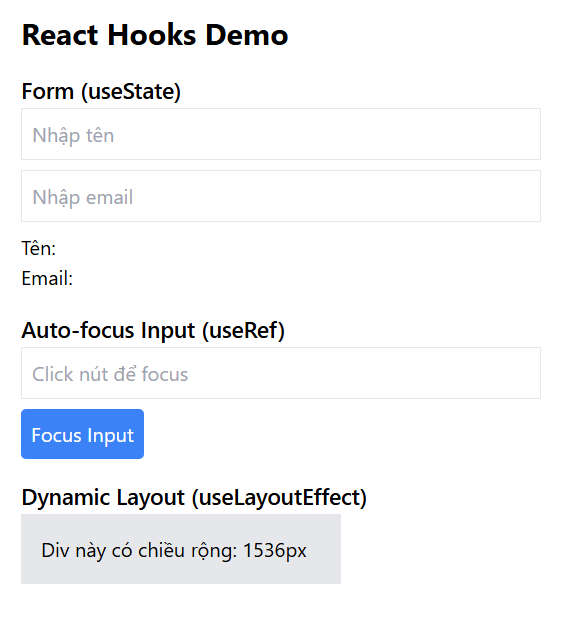
* Điểm chính:
* Đối tượng ‘ref’ tồn tại suốt chu kỳ sống của component.
* Thay đổi ‘ref.current’ không gây re-render, khác với ‘useState’.
* Trường hợp sử dụng phổ biến:
* Truy cập phần tử DOM để thực hiện các thao tác như focus, đo kích thước, hoặc điều chỉnh thuộc tính.
* Lưu trữ giá trị tạm thời, như ID của một timer hoặc giá trị trạng thái trước đó để so sánh.
* Best Practices:
* Sử dụng ‘useRef’ thay vì ‘useState’ khi giá trị không cần truyền vào props của component con hoặc không cần trigger re-render.
* Sử dụng callback refs nếu cần phản ứng với việc gắn hoặc tháo phần tử DOM.
* Lỗi thường gặp:
* Sử dụng ‘useState’ thay vì ‘useRef’ cho các giá trị không cần re-render, dẫn đến hiệu suất kém.

c. useLayoutEffect Hook

* Cú pháp: ‘useLayoutEffect(() => { /\* effect \*/ }, [dependencies]);’
* Tham số:
* Hàm effect: Mã cần chạy sau khi DOM được cập nhật nhưng trước khi trình duyệt vẽ lại.
* Dependencies: Mảng các giá trị xác định khi nào effect sẽ chạy lại.
* Cách sử dụng:
* Dùng cho các tác vụ cần đồng bộ, như đo lường kích thước phần tử DOM và cập nhật trạng thái dựa trên kết quả đó.
* Chạy đồng bộ sau tất cả các thay đổi DOM, trước khi trình duyệt vẽ lại giao diện.
* Khi nào sử dụng:
* Khi cần thực hiện các thao tác DOM phải được phản ánh ngay lập tức, như điều chỉnh vị trí của một tooltip hoặc ngăn chặn hiện tượng nhấp nháy giao diện.
* Ví dụ: Đo chiều cao của một phần tử và cập nhật vị trí của một thành phần khác dựa trên kết quả đó.
* Best Practices:
* Sử dụng tiết kiệm vì nó có thể làm chậm rendering do tính chất đồng bộ.
* Ưu tiên ‘useEffect’ cho các side effects không cần đồng bộ.
* Đảm bảo tất cả các dependencies được liệt kê đầy đủ để tránh lỗi logic.
* Lỗi thường gặp:
* Sử dụng ‘useLayoutEffect’ trong server-side rendering (SSR) có thể gây ra lỗi hoặc hiệu suất kém, vì nó không hỗ trợ tốt SSR.
* Thực hiện các tác vụ nặng trong ‘useLayoutEffect’ có thể làm chậm giao diện người dùng.

1. Demo

* Để có thể minh hoạ cách sử dụng các hooks, em sẽ tạo một ví dụ minh hoạ đơn giản:
* Form với State Update Thời Gian Thực sử dụng ‘useState’: Tạo một form với các trường nhập liệu như tên và email, sử dụng useState để quản lý trạng thái và hiển thị giá trị nhập liệu thời gian thực.
* Auto-focus Input sử dụng ‘useRef’: Tạo một component với một trường nhập liệu, sử dụng ‘useRef’ để lấy tham chiếu đến input và tự động focus khi component được render.
* Điều Chỉnh Layout Động sử dụng ‘useLayoutEffec’: Tạo một component với một div có thể thay đổi kích thước, sử dụng useLayoutEffect để đo kích thước của div và cập nhật giao diện dựa trên kết quả đó.



1. Kết luận

Tóm lại, ‘useState’, ‘useRef’, và ‘useLayoutEffect’ là các hook quan trọng trong React, cho phép các component functional quản lý trạng thái và side effects một cách hiệu quả. ‘useState’ là công cụ cơ bản để xử lý các thay đổi trạng thái, ‘useRef’ cung cấp cách truy cập DOM và lưu trữ giá trị có thể thay đổi mà không gây re-render, và ‘useLayoutEffect’ đảm bảo rằng một số hoạt động được thực hiện đồng bộ trước khi trình duyệt vẽ lại, giúp duy trì giao diện nhất quán.

|  |  |  |  |
| --- | --- | --- | --- |
| Hook | Mục đích chính | Trường hợp sử dụng | Lưu ý |
| useState | Quản lý trạng thái trong component functional. | Form nhập liệu, đếm số, hiển thị/ẩn thành phần giao diện. | Cập nhật bất đồng bộ, không mutate trực tiếp state. |
| useRef | Lưu trữ tham chiếu DOM hoặc giá trị có thể thay đổi mà không gây re-render. | Focus input, lưu trữ timer ID, đo lường DOM. | Thay đổi .current không gây re-render. |
| useLayoutEffect | Thực hiện side effects đồng bộ sau khi DOM được cập nhật, trước khi trình duyệt vẽ lại. | Đo kích thước phần tử, điều chỉnh layout để tránh nhấp nháy. | Sử dụng tiết kiệm, không phù hợp với server-side rendering. |