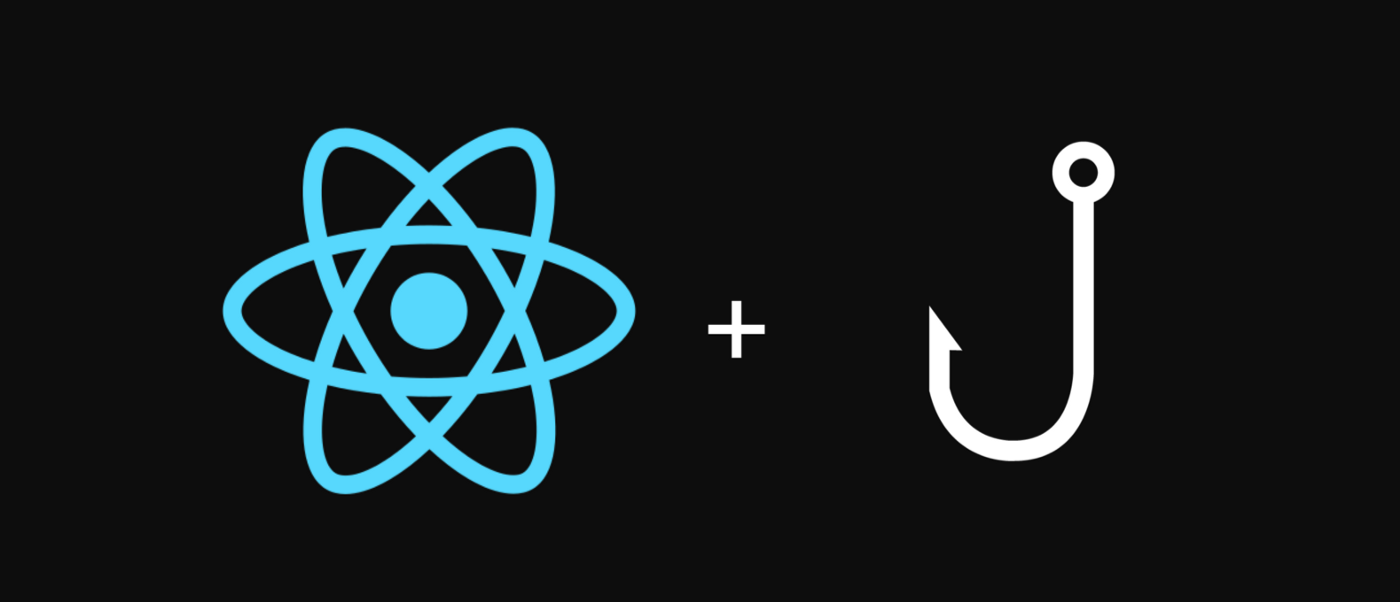
Day 4

Tổng quan về react native hooks****

**React Hook**

Chào mọi người đã quay trở lại với khóa học react native và  ở bài trước chúng ta đã được tìm hiểu về **Composition** và **Cấu trúc thư mục** và đã biết được cách phân chia thư mục để sau này có thể dễ dàng chỉnh sửa và phát triển. Hôm nay, chúng ta sẽ bắt đầu với một khái niệm mới đó là hook cùng … tìm hiểu nhé !

Sau đây là những phần chúng ta sẽ tìm hiểu trong buổi học ngày hôm nay.

**[1. Tổng quan về hooks]**

**[Đưa nội dung lý thuyết giải thích hook là gì]**

**1.**Khái niệm tổng quan về hook và lý do nên sử dụng hook

React Native Hook là một phần mở rộng của React Native, cho phép bạn sử dụng các tính năng và trạng thái của React trong các thành phần chức năng (functional components) thay vì trong các thành phần dựa trên lớp (class components). Hooks giúp bạn quản lý trạng thái, tác động phụ, và các khía cạnh khác của ứng dụng React Native một cách dễ dàng và hiệu quả hơn.

**Dưới đây là một số lý do tại sao bạn nên sử dụng React Native Hooks:**

**Sự đơn giản và gọn gàng**: Sử dụng hooks làm cho mã của bạn trở nên ngắn gọn hơn và dễ đọc hơn. Thay vì phải sử dụng các lớp và phải xử lý **this** trong các thành phần dựa trên lớp, bạn có thể viết mã ngắn hơn bằng cách sử dụng các hook trong các thành phần chức năng.

T**ách biệt các phần của logic:** Hooks cho phép bạn tách biệt các phần của logic của thành phần thành các hook khác nhau. Ví dụ, bạn có thể sử dụng **useState** để quản lý trạng thái và **useEffect** để quản lý các tác động phụ, giúp mã của bạn trở nên dễ quản lý hơn.

**Hiệu suất tốt hơn:** Hooks giúp bạn tối ưu hóa hiệu suất bằng cách tránh việc render lại không cần thiết. Bằng cách sử dụng **useMemo** và **useCallback**, bạn có thể tránh việc tính toán lại giá trị khi không cần thiết.

**Chia sẻ logic giữa các thành phần:** Bạn có thể tạo các custom hooks để chia sẻ logic giữa các thành phần chức năng khác nhau. Điều này giúp tái sử dụng mã một cách dễ dàng và duyệt quản logic ứng dụng của bạn.

**Tích hợp dễ dàng với các thư viện bên ngoài:** Hooks thường được hỗ trợ tốt bởi các thư viện và plug-in bên ngoài, giúp bạn dễ dàng tích hợp chúng vào dự án của mình.

**Cải thiện quá trình kiểm tra và bảo trì:** Hooks làm cho việc kiểm tra và bảo trì ứng dụng trở nên dễ dàng hơn bằng cách tạo các component chức năng có tính độc lập và dễ kiểm tra.

**Thúc đẩy các quy tắc tốt hơn cho React Native:** React Native đã chuyển sang khuyến nghị sử dụng hooks và các component  chức năng để phát triển ứng dụng. Việc sử dụng hooks giúp bạn tuân theo quy tắc và ứng dụng các mẫu phát triển tốt nhất của React Native.

**[2. Khái niệm về useState (hook)]**

**Nào chúng ta sẽ cùng đến với hook đầu tiên và hook này các bạn cần phải theo dõi và nắm thật chắc vì nó sẽ được sử dụng xuyên suốt trong quá trình xây dựng ứng dụng react của chúng ta.**

**[Giải thích]**

Hook ***useState*** trong React Native hoạt động giống như trong React.js, là một hook dùng để quản lý trạng thái (state) của một thành phần chức năng (functional component). Nó cho phép bạn theo dõi và cập nhật trạng thái trong các thành phần chức năng một cách đơn giản và hiệu quả.

**[Cách sử dụng]**

**Dưới đây là cách sử dụng useState trong React Native:**

Import useState từ thư viện React Native:

IMG_257

Sử dụng useState trong function component:

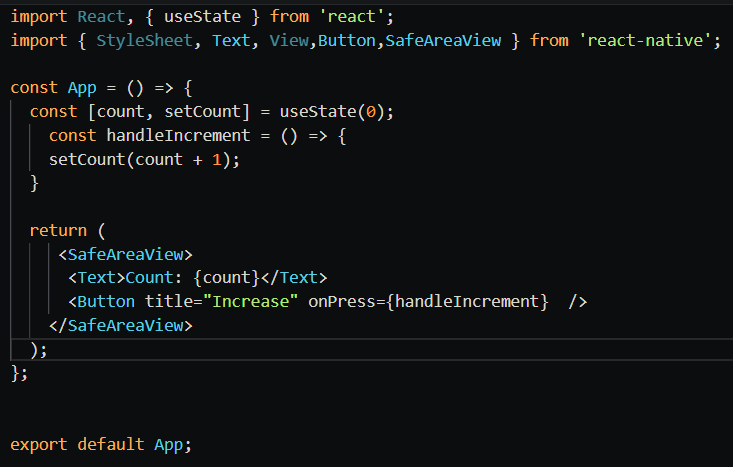
state là giá trị trạng thái hiện tại.

setState là một hàm để cập nhật giá trị trạng thái.

initialState là giá trị ban đầu cho trạng thái.

IMG_258

Sử dụng state để hiển thị giá trị trạng thái trong JSX và sử dụng setState để cập nhật giá trị trạng thái khi có sự kiện hoặc hành động xảy ra.



**[Hình minh hoạ]**

**Giải thích code**Trong ví dụ này, chúng ta sử dụng useState để tạo một biến trạng thái count với giá trị ban đầu là 0. Khi người dùng nhấn vào nút "Increase", chúng ta gọi setCount để cập nhật giá trị trạng thái, và sau đó giao diện người dùng được cập nhật để hiển thị giá trị mới của count.

**[3. Cách hoạt động]**

Khởi tạo trạng thái ban đầu:

Khi bạn sử dụng useState, bạn truyền giá trị ban đầu của trạng thái vào nó. Ví dụ: const [count, setCount] = useState(0); sẽ khởi tạo trạng thái count với giá trị ban đầu là 0.

Trả về một cặp giá trị:

useState trả về một cặp giá trị: giá trị trạng thái hiện tại và hàm để cập nhật trạng thái đó. Trong ví dụ trên, count là giá trị trạng thái hiện tại, và setCount là hàm để cập nhật count.

Sử dụng giá trị trạng thái trong JSX:

Bạn có thể sử dụng giá trị trạng thái (ví dụ: count) trong JSX để hiển thị nó cho người dùng. Ví dụ: <Text>Count: {count}</Text> sẽ hiển thị giá trị trạng thái count lên màn hình.

Sử dụng hàm cập nhật để thay đổi trạng thái:

Khi bạn muốn cập nhật giá trị trạng thái, bạn gọi hàm cập nhật (ví dụ: setCount) và truyền giá trị mới vào nó. Ví dụ: setCount(count + 1); sẽ tăng giá trị trạng thái count lên 1.

React tự động cập nhật giao diện người dùng:

Khi bạn gọi hàm cập nhật (setCount), React sẽ tự động cập nhật giao diện người dùng để hiển thị giá trị trạng thái mới. Điều này xảy ra bởi vì React theo dõi các thay đổi trong trạng thái và tự động render lại thành phần khi có sự thay đổi.

**[Khái niệm về useEffect (hook)]**

**useEffect** là một trong những hook quan trọng của React, được sử dụng để thực hiện các tác động phụ (side effects) trong thành phần chức năng. Tác động phụ bao gồm các hoạt động như gửi yêu cầu mạng, đăng ký sự kiện, tương tác với DOM, và nhiều công việc không đồng bộ khác.

**[Cách hoạt động]**

**useEffect trong React Native tương tự như trong Reactjs, bao gồm các phần sau:**

Khởi tạo và định nghĩa **useEffect**:

**useEffect** là một hook được sử dụng bên trong một thành phần chức năng trong React Native.

Bạn gọi **useEffect** và truyền một hàm làm tham số đầu tiên. Hàm này sẽ chứa logic của các tác động phụ bạn muốn thực hiện.

Các tác động phụ nằm trong hàm:

Hàm được truyền vào **useEffect** chứa logic của các tác động phụ bạn muốn thực hiện. Ví dụ: gửi yêu cầu mạng, đăng ký sự kiện, cập nhật trạng thái, và nhiều tác động khác.

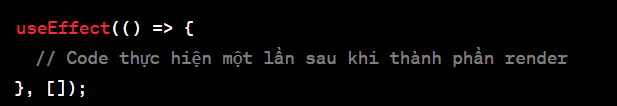
Quản lý dependencies (tham số thứ hai):

Bạn có thể truyền một mảng dependencies (tham số thứ hai của **useEffect**). Mảng này giúp bạn kiểm soát khi **useEffect** sẽ được gọi lại. Nếu mảng dependencies thay đổi giữa các lần render, **useEffect** sẽ được gọi lại.

**[Cách sử dụng]**

**Ví dụ sử dụng useEffect trong React Native:**

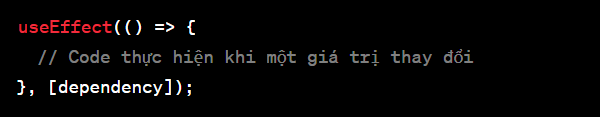
**[Ví dụ 1]: Thực hiện một lần sau khi component mount (componentDidMount):**

****

**[Hình ảnh minh hoạ]**

Trong ví dụ này, useEffect được gọi một lần sau khi thành phần render lần đầu. Tham số thứ hai là một mảng dependencies rỗng, cho biết useEffect không phụ thuộc vào bất kỳ giá trị nào và chỉ cần gọi một lần.

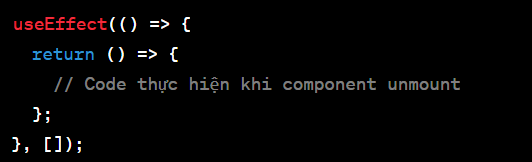
**[Ví dụ 2]: Thực hiện mỗi khi một giá trị thay đổi (componentDidUpdate):**

****

**[Hình minh hoạ ví dụ khi có phụ thuộc]**

Trong ví dụ này, useEffect sẽ được gọi mỗi khi dependency thay đổi. Bất kỳ thay đổi nào của dependency sẽ kích hoạt useEffect.

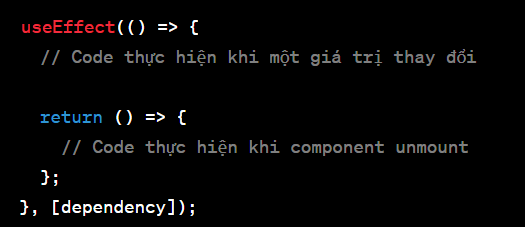
**[Ví dụ 3]:Thực hiện khi component unmount (componentWillUnmount):**

****

**[Hình ảnh minh hoạ cleanup effect]**

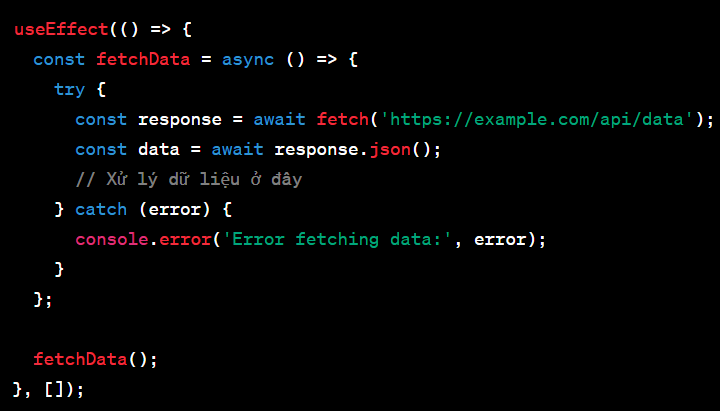
Trong ví dụ này, useEffect sẽ trả về một hàm cleanup. Hàm này sẽ được gọi khi component unmount, tương tự như componentWillUnmount.

**[Ví dụ 4]:Thực hiện khi một giá trị thay đổi và khi component unmount (componentDidUpdate và componentWillUnmount):**

****

Trong ví dụ này, useEffect sẽ được gọi mỗi khi dependency thay đổi và hàm cleanup sẽ được gọi khi component unmount.

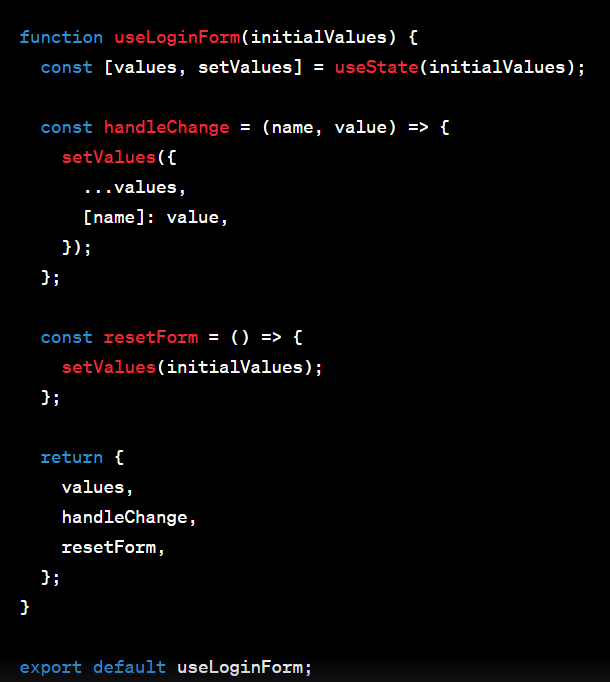
**[Ví dụ 5]: Sử dụng async/await (bất đồng bộ ) trong useEffect:**

****

**[2. Khái niệm về Custom Hook(hook)]**

Trong React Native, custom hook là một khái niệm tương tự như trong React.js. Một custom hook là một hàm JavaScript mà bạn có thể viết và tái sử dụng để chia sẻ logic và trạng thái giữa các thành phần chức năng khác nhau.

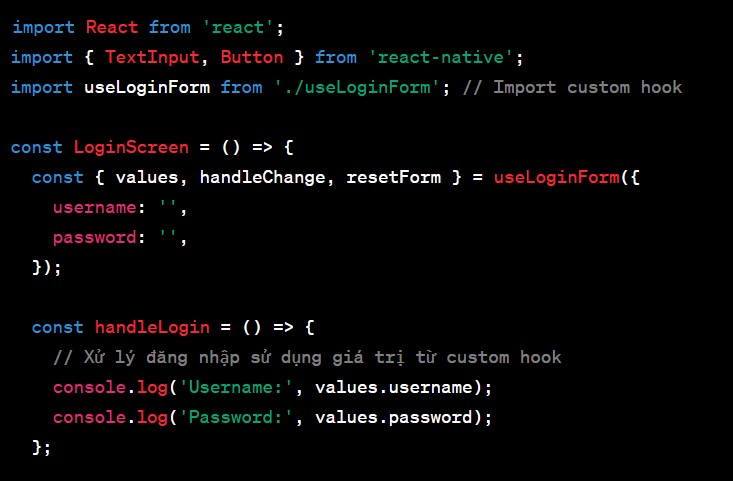
Custom hook thường bắt đầu bằng tiền tố "use" và thường được sử dụng để trừu tượng hóa và quản lý các khía cạnh của ứng dụng như gọi API, quản lý trạng thái, xử lý sự kiện, và nhiều công việc khác.

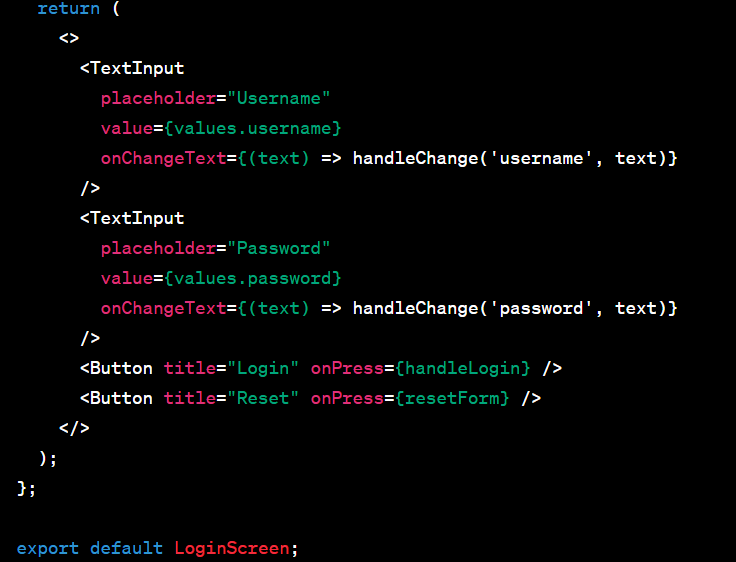
****

**Custom Hook**

Custom hook này (useLoginForm) quản lý trạng thái của biểu mẫu đăng nhập và cung cấp các hàm để thay đổi giá trị của các trường trong biểu mẫu và đặt lại biểu mẫu khi cần thiết. Bạn có thể sử dụng nó trong bất kỳ thành phần chức năng nào trong ứng dụng của bạn để quản lý trạng thái của biểu mẫu đăng nhập một cách dễ dàng.

**[Áp dụng custom hook vào component]**

****

****

Custom hook useLoginForm ở trên giúp bạn quản lý trạng thái của biểu mẫu đăng nhập và cho phép tái sử dụng nó trong bất kỳ thành phần nào cần đăng nhập.

**[6.1 Giới thiệu useRef]**

**[ Giới thiệu Lý thuyết]**

**useRef**() là một hook trong React Native được sử dụng để tạo và truy cập tham chiếu (ref) đến các DOM elements hoặc lưu trạng thái không gây re-render.

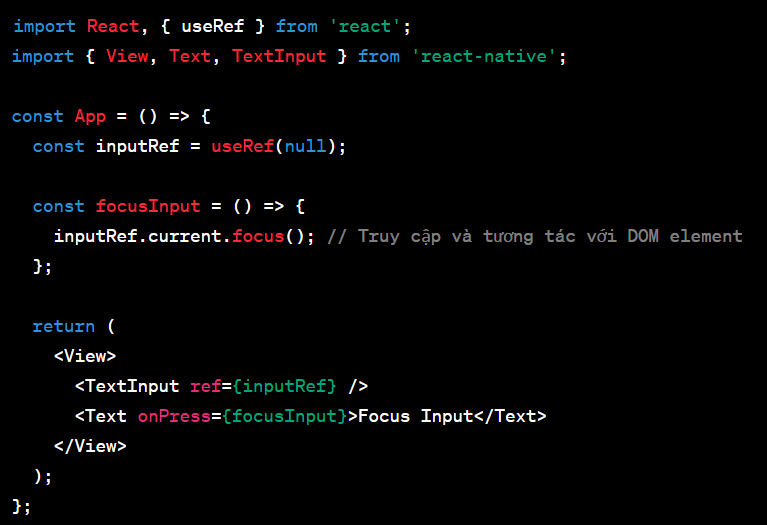
**[Cách hoạt động]**

Khi bạn sử dụng useRef(), nó sẽ tạo một đối tượng ref và gán nó vào một biến.

Bạn có thể truy cập đối tượng DOM element bằng cách sử dụng .current trên đối tượng ref.

Thay đổi giá trị của đối tượng ref không gây re-render cho thành phần chứa nó.

**[Cách sử dụng]**

****

**[Giải thích về đoạn mã]**

Trong đoạn mã này, chúng ta sử dụng hook useRef() để tạo một tham chiếu (inputRef) đến một DOM element <TextInput />.

Khi bạn nhấn vào phần tử Text, hàm focusInput được gọi, và nó sử dụng inputRef.current.focus() để tập trung vào phần tử <TextInput />, bất kể nó ở đâu trong cây component.

**[6.2 Giới thiệu useCallback]**

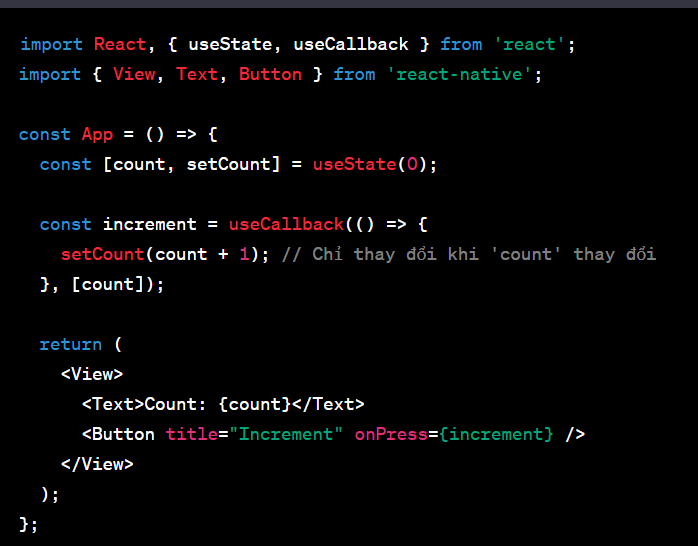
**[Giải thích về hook useCallback]**

**useCallback**() là một hook được sử dụng để tạo và memoize các hàm callback, đảm bảo rằng chúng không bị tạo lại mỗi khi thành phần render.

**[Cách hoạt động]**

**useCallback**() nhận một hàm callback và một mảng dependencies.

Hàm callback sẽ được memoize và chỉ được tạo lại khi bất kỳ dependencies nào trong mảng thay đổi

**[Cách sử dụng]**

**[Giải thích về đoạn mã]**

Trong đoạn mã này, chúng ta sử dụng hook useCallback() để tạo một hàm increment và memoize nó.

Hàm increment chỉ thay đổi khi giá trị của count thay đổi (được xác định bởi mảng dependencies [count]).

Điều này đảm bảo rằng hàm increment không bị tạo lại mỗi khi thành phần re-render, giúp tối ưu hóa hiệu suất.

**[6.3 Giới thiệu useMemo]**

**[Giải thích hook]**

**useMemo**() là một hook được sử dụng để tính toán và memoize giá trị dựa trên các dependencies, giúp tối ưu hóa tính toán không cần thiết.

**[Cách hoạt động]**

**useMemo**() nhận một hàm callback và một mảng dependencies.

Nó tính toán giá trị của hàm callback và memoize nó.

Giá trị mới sẽ được tính toán và memoize lại chỉ khi bất kỳ dependencies nào trong mảng thay đổi.

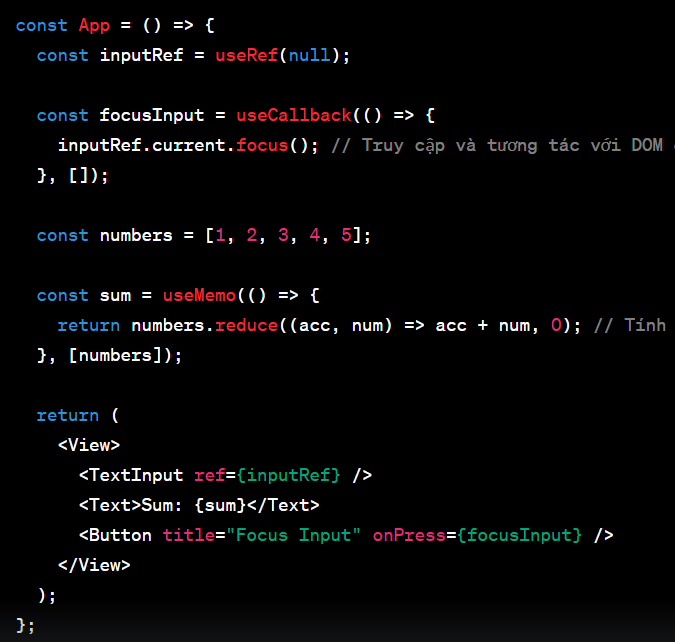
**[Cách sử dụng]**

có thể kết hợp các hook useRef(), useCallback(), và useMemo() để tối ưu hóa hiệu suất và quản lý trạng thái trong ứng dụng React Native.

**[Cách hoạt động]**

Khi kết hợp các hook này, có thể truy cập DOM elements, tạo và memoize hàm callback, và tính toán giá trị dựa trên dependencies.

**[Cách sử dụng]**

****

sử dụng useRef() để tạo một tham chiếu inputRef đến một DOM element <TextInput />. Điều này cho phép chúng ta truy cập và tương tác với phần tử này.

sử dụng useCallback() để tạo hàm focusInput. Hàm này chỉ được tạo lại khi dependencies (trong trường hợp này, mảng rỗng []) thay đổi. Việc này đảm bảo rằng hàm focusInput không bị tạo lại mỗi khi thành phần re-render, tối ưu hóa hiệu suất.

sử dụng useMemo() để tính toán giá trị sum. Hàm callback bên trong useMemo() tính tổng của các số trong mảng numbers. Giá trị sum chỉ tính toán lại khi mảng numbers thay đổi. Điều này giúp tránh tính toán không cần thiết mỗi khi thành phần re-render và tối ưu hóa hiệu suất.

render các phần tử UI, bao gồm một <TextInput /> được tham chiếu bởi inputRef, một phần tử hiển thị tổng sum, và một nút để gọi hàm focusInput để tập trung vào <TextInput />.

Kết hợp các hook này cho phép chúng ta quản lý trạng thái, tối ưu hóa hiệu suất và tạo các hàm callback mà không cần tính toán lại mỗi khi thành phần re-render.

Tổng quan về các khái niệm và hook trong React Native đã được trình bày trong bài viết này:

Hooks trong React Native: Đây là một tính năng mạnh mẽ cho phép quản lý trạng thái và logic của thành phần chức năng trong ứng dụng React Native.

useState: Hook này được sử dụng để quản lý trạng thái của thành phần. Nó cho phép khai báo biến trạng thái và một hàm để cập nhật giá trị trạng thái đó.

useEffect: Hook này được sử dụng để thực hiện các tác vụ sau khi thành phần đã render, chẳng hạn như gọi API, đăng ký sự kiện, hoặc thay đổi trạng thái.

Custom hook: Custom hook là một hàm JavaScript được viết để chia sẻ logic và trạng thái giữa các thành phần chức năng khác nhau. Chúng thường bắt đầu bằng tiền tố "use" và có thể được sử dụng lại trong toàn ứng dụng.

Các hook bổ sung:

useRef(): Được sử dụng để tạo tham chiếu đến DOM elements và lưu trạng thái không gây re-render.

useCallback(): Được sử dụng để memoize hàm callback, giúp tối ưu hóa hiệu suất.

useMemo(): Được sử dụng để tính toán và memoize giá trị dựa trên dependencies, giúp tối ưu hóa tính toán không cần thiết.

Các hook này cùng nhau giúp bạn xây dựng các ứng dụng React Native mạnh mẽ, hiệu quả với quản lý trạng thái, hiệu suất và tái sử dụng code dễ dàng hơn. Chúng là một phần quan trọng trong việc phát triển ứng dụng di động sử dụng React Native.

**Bài tập**

1. **Sử dụng useState:**

Tạo một ứng dụng React Native đơn giản với một nút "Tăng số" và một hiển thị số.

Sử dụng hook useState để thực hiện tính năng tăng số khi nút được nhấn.

Sử dụng useEffect:

Tạo một ứng dụng React Native đơn giản với một nút "Gọi API" và một vùng hiển thị dữ liệu.

Sử dụng hook useEffect để gọi một API giả lập và hiển thị dữ liệu lấy về.

2.**Kết hợp useState và useEffect:**

Tạo một ứng dụng React Native cho việc quản lý danh sách công việc (Todo List).

Sử dụng useState để quản lý danh sách công việc và useEffect để lưu trạng thái danh sách vào bộ nhớ cục bộ khi thay đổi.

Sử dụng useRef:

Mở rộng ứng dụng Todo List bằng cách thêm tính năng chỉnh sửa công việc.

Sử dụng hook useRef để thực hiện việc chỉnh sửa công việc mà không cần tạo lại danh sách.

3. Hãy tạo một màn hình loading screen, sau khi loading xong sẽ hiện ra chữ “**Dữ liệu đã tải thành công!**”

4. **Tạo custom hook:**

Tạo một custom hook cho phép bạn theo dõi kích thước cửa sổ và cập nhật nó khi cửa sổ thay đổi.

Sử dụng custom hook này trong ứng dụng để hiển thị kích thước cửa sổ thời gian thực.

Tối ưu hóa hiệu suất:

Tối ưu hóa ứng dụng Todo List của bạn bằng cách sử dụng useCallback và useMemo để tránh tính toán không cần thiết và tạo lại hàm callback.

Bài tập tự luyện (Basic)

Sử dụng **useState** và **useEffect**:

Tạo một ứng dụng React Native đơn giản và sử dụng useState để quản lý một biến trạng thái (ví dụ: một biến để đếm số lần người dùng nhấn vào một nút).

Sử dụng useEffect để theo dõi thay đổi trong biến trạng thái và hiển thị thông báo khi giá trị thay đổi.

Sử dụng **useRef**:

Tạo một ứng dụng có một input và một nút.

Sử dụng useRef để thực hiện tập trung vào input khi nút được nhấn.

Sử dụng **useCallback** và **useMemo**:

Tạo một ứng dụng có danh sách các phần tử.

Sử dụng **useCallback** để tối ưu hóa việc truyền các hàm callback vào các thành phần con.

Sử dụng **useMemo** để tối ưu hóa tính toán các giá trị dẫn xuất.

Sử dụng **Hooks** tùy chỉnh:

Tạo một hook tùy chỉnh để quản lý một tính năng cụ thể (ví dụ: xử lý đăng nhập, tạo thông báo, quản lý form).

Sử dụng hook tùy chỉnh trong ứng dụng của bạn để tách logic và làm cho mã dễ đọc hơn.

Sử dụng **useLayoutEffect**:

Tạo một ứng dụng với một số phần tử giao diện người dùng.

Sử dụng useLayoutEffect để thực hiện các tác vụ trước khi giao diện người dùng được vẽ lên màn hình.

Sử dụng **useWindowDimensions**:

Tạo một ứng dụng và sử dụng **useWindowDimensions** để lấy kích thước cửa sổ của thiết bị và tùy chỉnh giao diện người dùng dựa trên kích thước này.

**NÂNG CAO**

**Bài tập 1 : Quản lý Danh Sách Công Việc (To-Do List)**

Mục tiêu: Tạo một ứng dụng React Native sử dụng React Hooks để quản lý danh sách công việc (to-do list).

Gợi Ý:

* Tạo một thành phần chứa một danh sách các công việc (to-do items).
* Sử dụng useState để quản lý danh sách công việc và cập nhật giao diện người dùng khi danh sách thay đổi.
* Thêm nút "Thêm Công Việc" để cho phép người dùng thêm mới công việc vào danh sách.
* Sử dụng useEffect để lưu trạng thái danh sách công việc vào bộ nhớ cục bộ khi danh sách thay đổi.
* Cho phép người dùng đánh dấu công việc là đã hoàn thành hoặc xóa công việc khỏi danh sách bằng cách sử dụng useState.

**Bài tập 2 : Ứng Dụng Ghi Chú Cá Nhân**

Mục tiêu: Xây dựng một ứng dụng ghi chú cá nhân đơn giản bằng React Native, sử dụng các React Hooks theo cách bạn muốn.

Không có gợi ý cụ thể cho bài tập này, nhưng dưới đây là một số yếu tố bạn có thể xem xét:

* Tạo các thành phần cho việc hiển thị danh sách các ghi chú và thêm ghi chú mới.
* Sử dụng các hooks như useState, **useEffect**, **useCallback**, và **useContext** (nếu cần) để quản lý trạng thái ứng dụng và tương tác với dữ liệu.
* Cho phép người dùng thêm, sửa đổi, xóa ghi chú và xem chi tiết từng ghi chú.
* Lưu trạng thái ứng dụng (danh sách ghi chú) vào cơ sở dữ liệu hoặc bộ nhớ cục bộ (**AsyncStorage**) để dữ liệu không bị mất khi ứng dụng được tắt và mở lại.