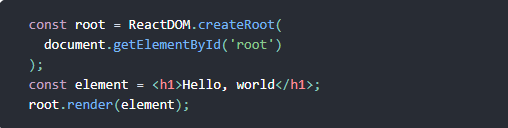
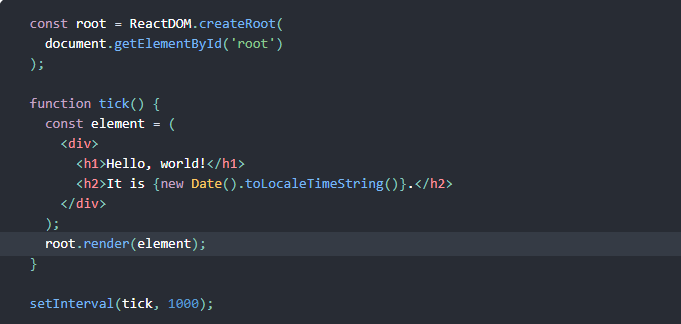
**RENDERING ELEMENTS**

* Để render một React element vào bên trong một “nốt” (node) DOM gốc

****

* Cập nhật element được render

Các React element là [bất biến](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%91i_t%C6%B0%E1%BB%A3ng_b%E1%BA%A5t_bi%E1%BA%BFn). Một khi bạn tạo ra một element, bạn không thể thay đổi các “con” (children) hoặc các “thuộc tính” (attribute) của nó. Một element giống như một khung hình duy nhất



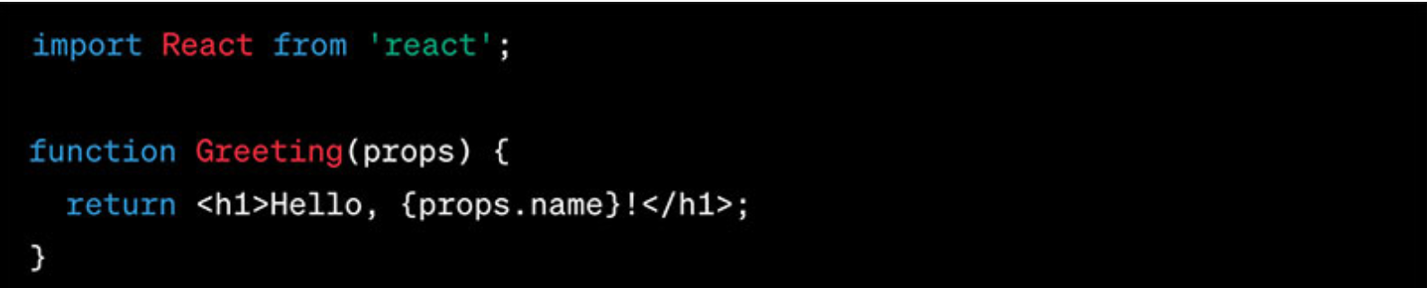
**COMPONENTS VÀ PROPS**

**Component là gì ?**

1 component là một block code độc lập, có thể tái sử dụng, nó chia UI thành nhiều phần nhỏ.

**Function components**

* Functional Component là một hàm JavaScript đơn giản, **nhận các đối số (props) và trả về các phần tử React**.
* Functional components không thể làm nhiều thứ phức tạp như quản lý React State (data) hoặc phương thức life-cycle trong functional components.

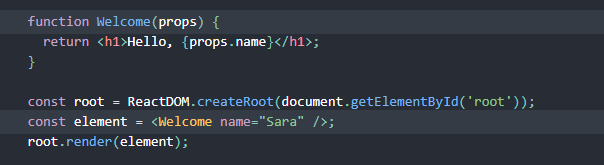


**Class components**

* Class Component là **một lớp JavaScript kế thừa từ React.Component**. Nó có thể **duy trì trạng thái (state) và có nhiều phương thức mạnh mẽ** hơn.
* phương thức khởi tạo, life-cycle, hàm render() và quản lý state (data).

****

**Rendering component**

****

1. Chúng ta gọi root.render() với <Welcome name="Sara" /> element.
2. React gọi đến Welcome component với {name: 'Sara'} là props.
3. Welcome component của chúng ta trả về kết quả là <h1>Hello, Sara</h1> element.
4. React DOM sẽ cập nhật DOM để hiển thị <h1>Hello, Sara</h1>.

**Props**

* Props là viết tắt của **thuộc tính** và chúng được sử dụng để truyền dữ liệu giữa các thành phần React. Luồng dữ liệu của React giữa các thành phần là đơn hướng (từ cha sang con).
* Truyền data với props

***class ParentComponent extends Component {***

***render() {***

***return (***

***<ChildComponent name="First Child" />***

***);***

***}***

***}***

***const ChildComponent = (props) => {***

***return <p>{props.name}</p>;***

***};***

* xác định lấy một số dữ liệu từ thành phần cha và gán nó cho thuộc tính “prop” của thành phần con.

***<ChildComponent name="First Child" />***

* “Name” sẽ xác định prop ở đây và chứa data văn bản. Sau đó chúng ta có thể truyền data cùng với props giống như chúng ta đang đưa ra một đối số cho function:

***const ChildComponent = (props) => {***

***// statements***

***};***

* chúng ta sử dụng dấu chấm để truy cập prop data và render nó

***return <p>{props.nzame}</p>;***

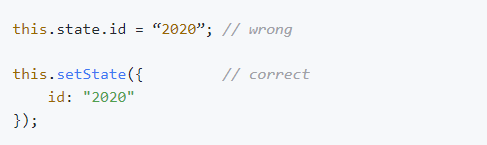
**STATE VÀ LIFECYCLE**

**State**

* State cho phép các thành phần tạo và quản lý dữ liệu của riêng chúng. ác thành phần không thể truyền dữ liệu với state, nhưng chúng có thể tạo và quản lý nó trong nội bộ.
* ví dụ cho thấy cách sử dụng state:



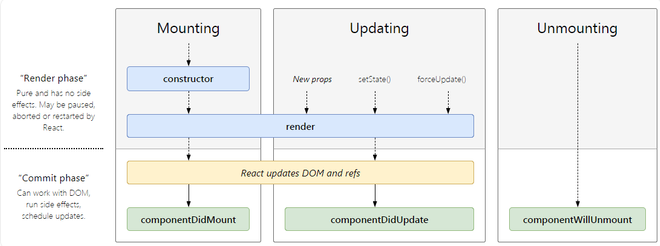
* Không nên trực tiếp sửa đổi state, nhưng nó có thể sửa đổi cùng với một cách đặc biệt được gọi là setState.



**LIFECYCLE**

Life cycle của component trong reactjs là quá trình từ khi tạo ra, thay đổi và hủy bỏ component. Gồm 3 giai đoạn:

* Tạo ra (**Mounting**)
* Thay đổi (**Updating**)
* Hủy bỏ (**UnMounting**)



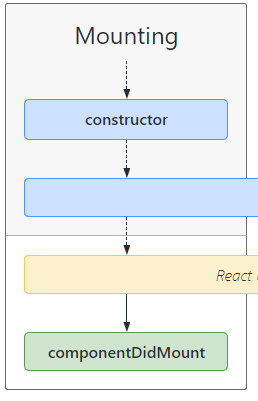
Khi Component **Mounting**(tạo ra) sẽ gọi những hàm nào: (constructor, render, DidMount). **Updating**(thay đổi) sẽ gọi những hàm nào(render, DidUpdate). **Unmounting**(hủy bỏ) sẽ gọi hàm WillUnmount.

**MỘT SỐ HÀM**

**1. Constructor**

* Khi component **mouting**(tạo ra) sẽ đi qua hàm **Constructor**
* Trong constructor sẽ khai báo các **state**, các **properties**(thuộc tính) của component.

**Lưu ý:** super(props) để gọi hàm khởi tạo của component cha React.Component mà Component con kế thừa.



class Clock extends React.Component {

constructor(props) {

super(props);

this.state = {date: new Date()};

}

}

**2. DidMount()**

* Khi component được **Mounting**(tạo ra) sau khi đi qua hàm Constructor và render lần đầu thì sẽ gọi hàm **DidMount()**.
* Thường được dùng để gọi api để lấy dữ liệu, setState để cập nhật state

Như ví dụ ở dưới: gọi api để get nội dung comment và sau đó setState để rerender lại 1 lần nữa. (lúc này đã có nội dung comment lấy từ api)

\*\*Lưu ý: \*\*hàm **DidMount()** chỉ chạy 1 lần duy nhất khi component được tạo ra.

componentDidMount() {

fetchPosts().then(response => {

this.setState({

posts: response.posts

});

});

fetchComments().then(response => {

this.setState({

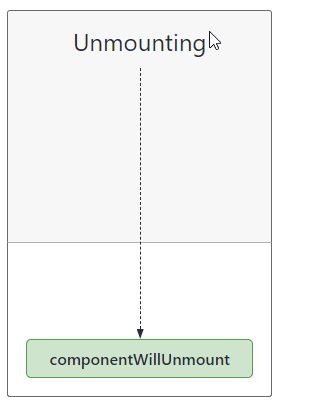
comments: response.comments

});

});

}

**3. WillUnMount()**



* Khi component **unmounting** (hủy bỏ) ta sẽ gọi hàm **WillUnMount**().
* Khi ta không render component hoặc chuyển trang thì component sẽ bị hủy bỏ để render nội dung mới lên.
* Dùng để hủy timeout, clearInterval. (nếu không hủy bỏ thì sẽ bị chạy hoài liên tục), reset dữ liệu nếu cần thiết.

**Lưu ý:** hàm **WillUnMount()** chỉ chạy 1 lần duy nhất khi component trong vòng đời của component. Tương tự **Mount()** chỉ chạy 1 lần duy nhất. Còn **DidUpdate()** có thể gọi nhiều lần nếu có nhiều update

componentWillUnmount() {

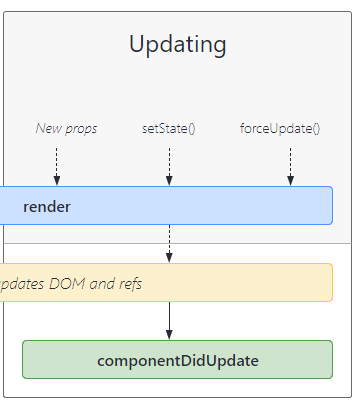
clearInterval(this.timerID);

}

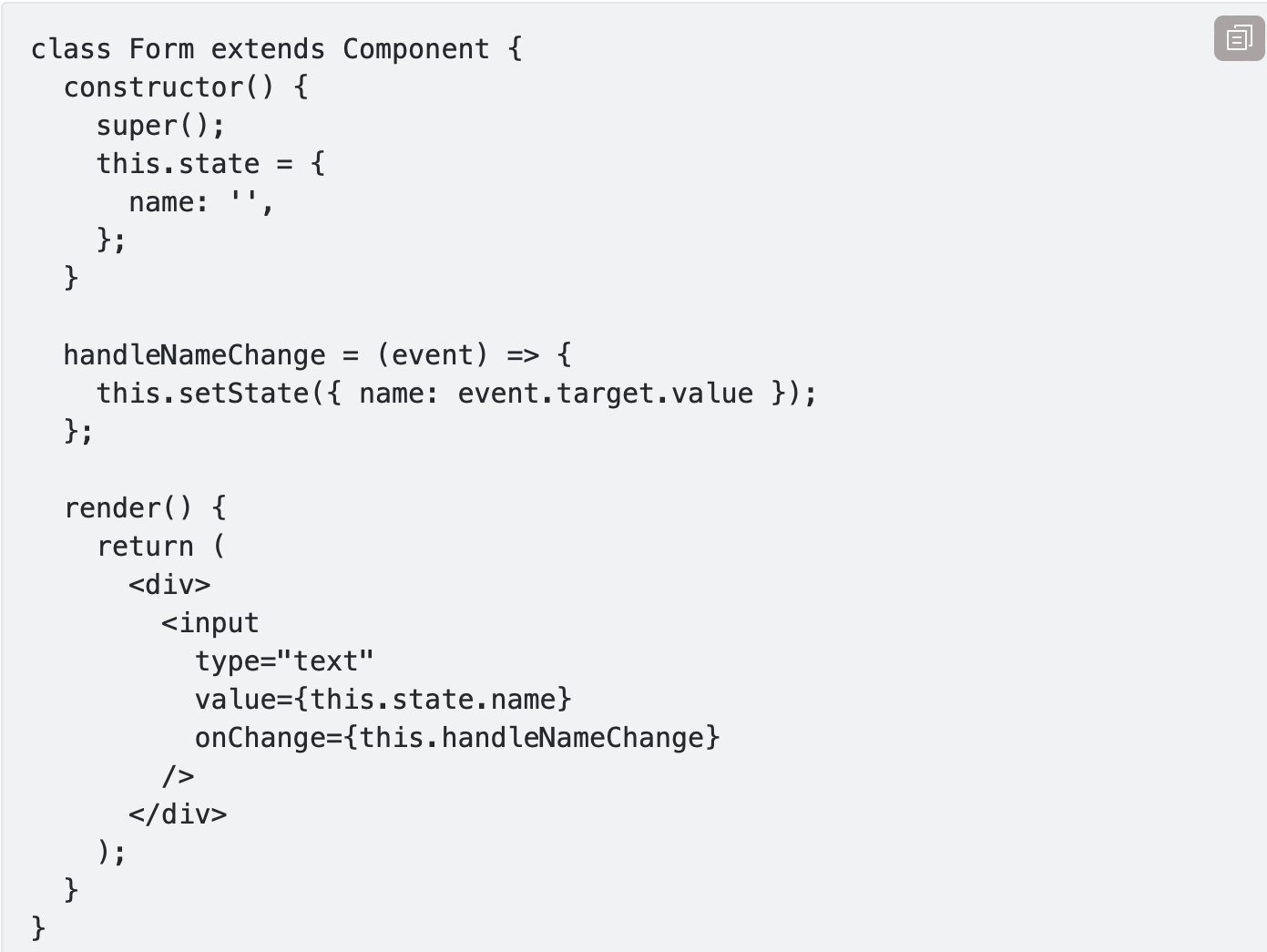
**4. DidUpdate()**

* Được dùng khi component **Updating** (thay đổi ) sẽ gọi hàm **DidUpdate()**
* **DidUpdate()** có thể không được gọi hoặc gọi nhiều lần nếu có update component.(khi có props thay đổi, state thay đổi hoặc bắt buộc update (forceUpdate)
* Khi render sẽ trigger gọi hàm **DidUpdate()**

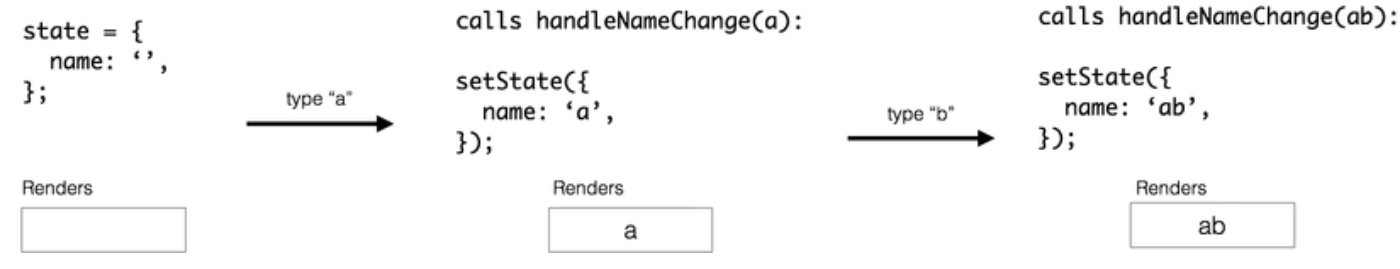
**Lưu ý:** Cẩn thận khi xài DidUpdate() cần quản lý chặt chẽ các thay đổi của props, state. Nếu không cẩn thận thì có khả năng bị render nhiều lần gây chậm chương trình. Thậm chí có trường hợp lặp vô tận do thay đổi state bên trong hàm **DidUpdate()**



**CONTROLLED COMPONENT**



Mỗi khi bạn thay đổi input thì handleNameChange được gọi và nó sẽ cập nhật giá trị mới cho state. Sau khi state thay đổi thì nó sẽ render lại form với giá trị tương ứng với giá trị của state.



. Giá trị khởi tạo của [state.name](http://state.name) sẽ là một string rỗng ''. . Khi bạn gõ 'a' vào ô input thì handleNameChange sẽ được gọi và cập nhật lại giá trị mới cho state là a. Tiếp đó là ô input sẽ được render lại với giá trị là state.name . Khi bạn gõ tiếp 'b' thì logic cũng tương tự. Khi đó [state.name](http://state.name) sẽ có giá trị là 'ab' và giá trị này sẽ là giá trị của input khi render lại.

Ta có thể hiểu là ô input trong ví dụ trên luôn luôn hiển thị giá trị của state.name

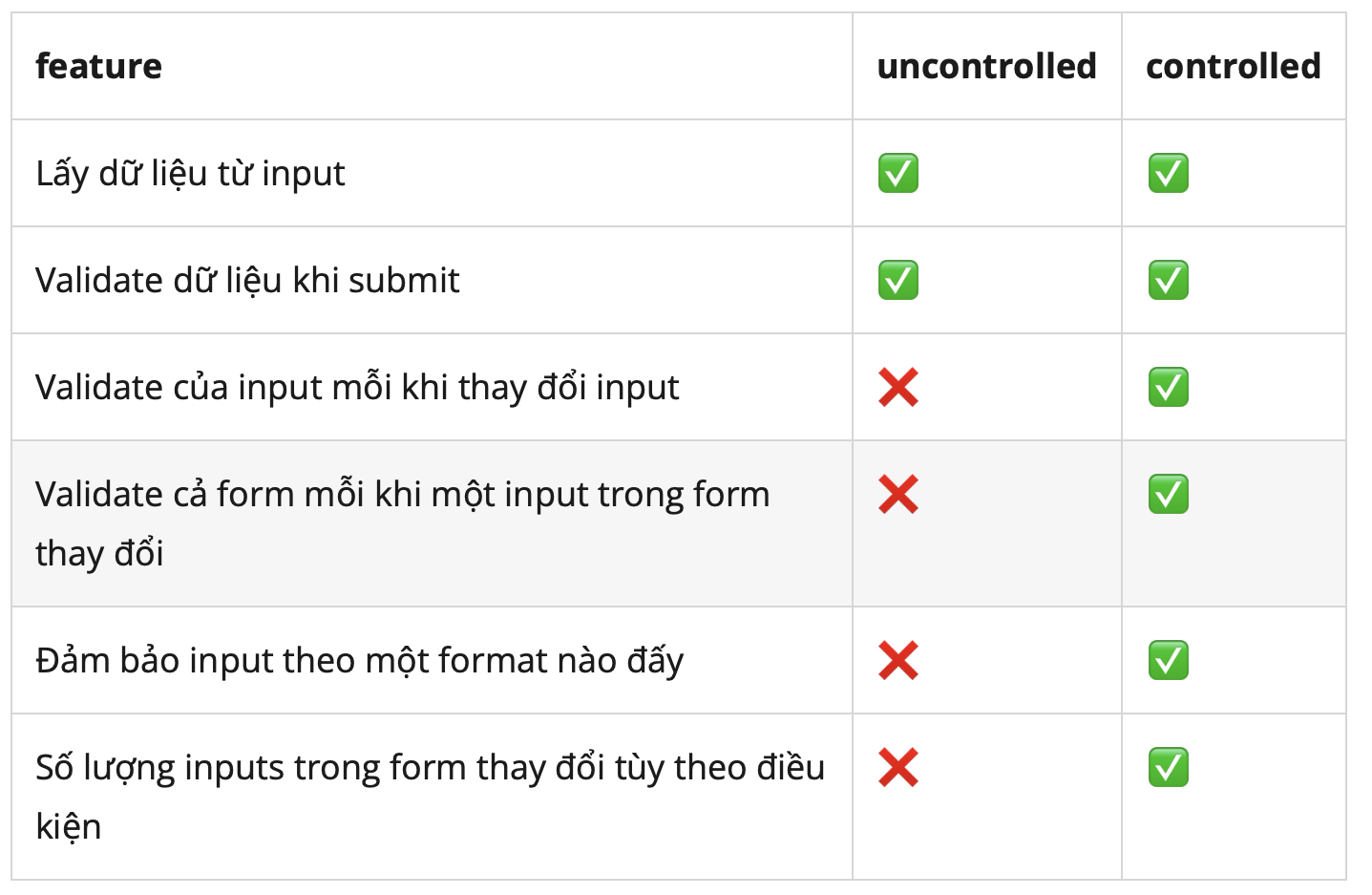
**UNCONTROLLED COMPONENT**

Để viết một uncontrolled component  bạn sẽ ko viết các event bắt sự kiện thay đổi input trong form mà bạn sẽ sử dụng một tham chiếu trực tiếp đến DOM.



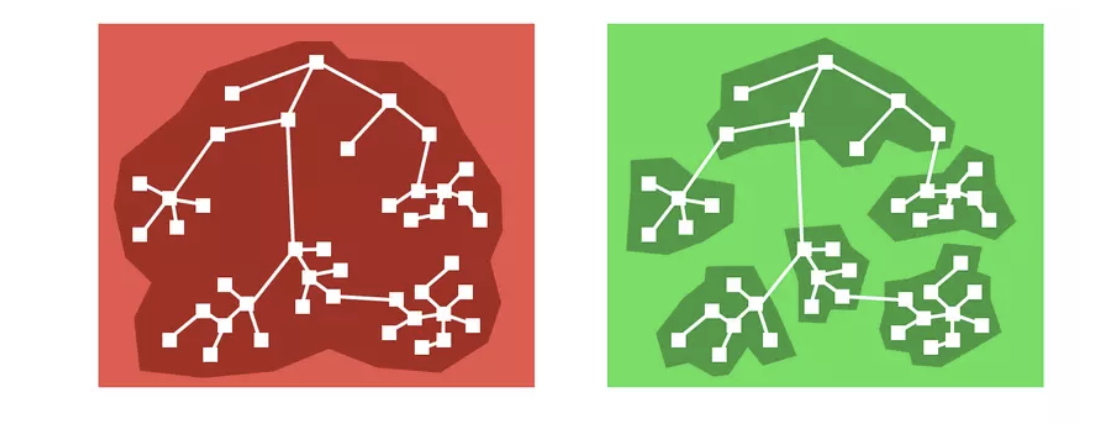
đặt một biến tham chiếu là input tham chiếu đến DOM object. Khi cần lấy giá trị ta sẽ truy xuất giá trị thông qua tham chiếu này.

Chú ý: Việc khởi tạo giá trị cho input trong trường hợp dùng uncontrolled component khác với controlled component. Bạn ko thể dùng thuộc tính value mà dùng thuộc tính defaultValue bởi vì nếu bạn dùng thuộc tính value thì mỗi khi form render lại nó sẽ luôn set giá trị này cho input, còn defaultValue chỉ set duy nhất lần đầu tiên khi khởi tạo form.



CODE-SPLITTING

* + Code-Splitting là một trong những kỹ thuật giúp tăng tốc thời gian load Javascript của React App.
  + code-splitting có nghĩa là chúng ta chia nhỏ code hiệu quả hơn và load những phần cần thiết cho trang web. Kỹ thuật này cũng được hiểu như là lazy load javascript.



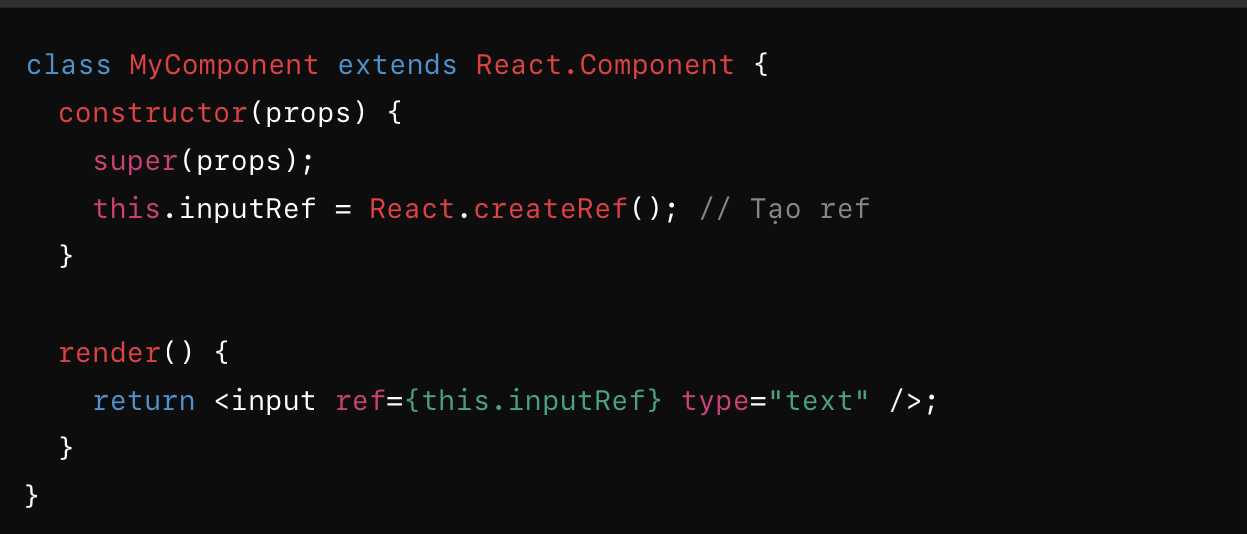
* + *Code splitting looks like* Với phần màu đỏ, code chưa được splitting, chúng ta có thể thấy rằng tất cả đều gom về một cục bundle.js. Ngược lại, với phần màu xanh, code được chia thành nhiều phần nhỏ. Có thể được chia nhỏ theo module và một khi load một trang thuộc về một module nào đó thì phần code nhỏ sẽ được load lên.

REFS

Refs trong React là một cách để truy cập và tương tác với các DOM elements hoặc các instance của components mà không cần phải sử dụng state để quản lý giá trị. Refs rất hữu ích trong các trường hợp như:

* Truy cập vào các element DOM để thực hiện các thao tác như lấy giá trị, thiết lập focus, hoặc thực hiện animations.
* Gửi các phương thức đến các components con để điều khiển hành vi của chúng.

Cách sử dụng:





**Khi nào nên sử dụng Refs**

* **Truy cập DOM**: Khi bạn cần tương tác trực tiếp với một element DOM, ví dụ như khi thiết lập focus.
* **Quản lý animations**: Khi bạn cần điều khiển các animations trực tiếp.
* **Tương tác với third-party libraries**: Khi bạn làm việc với các thư viện bên ngoài không sử dụng React.

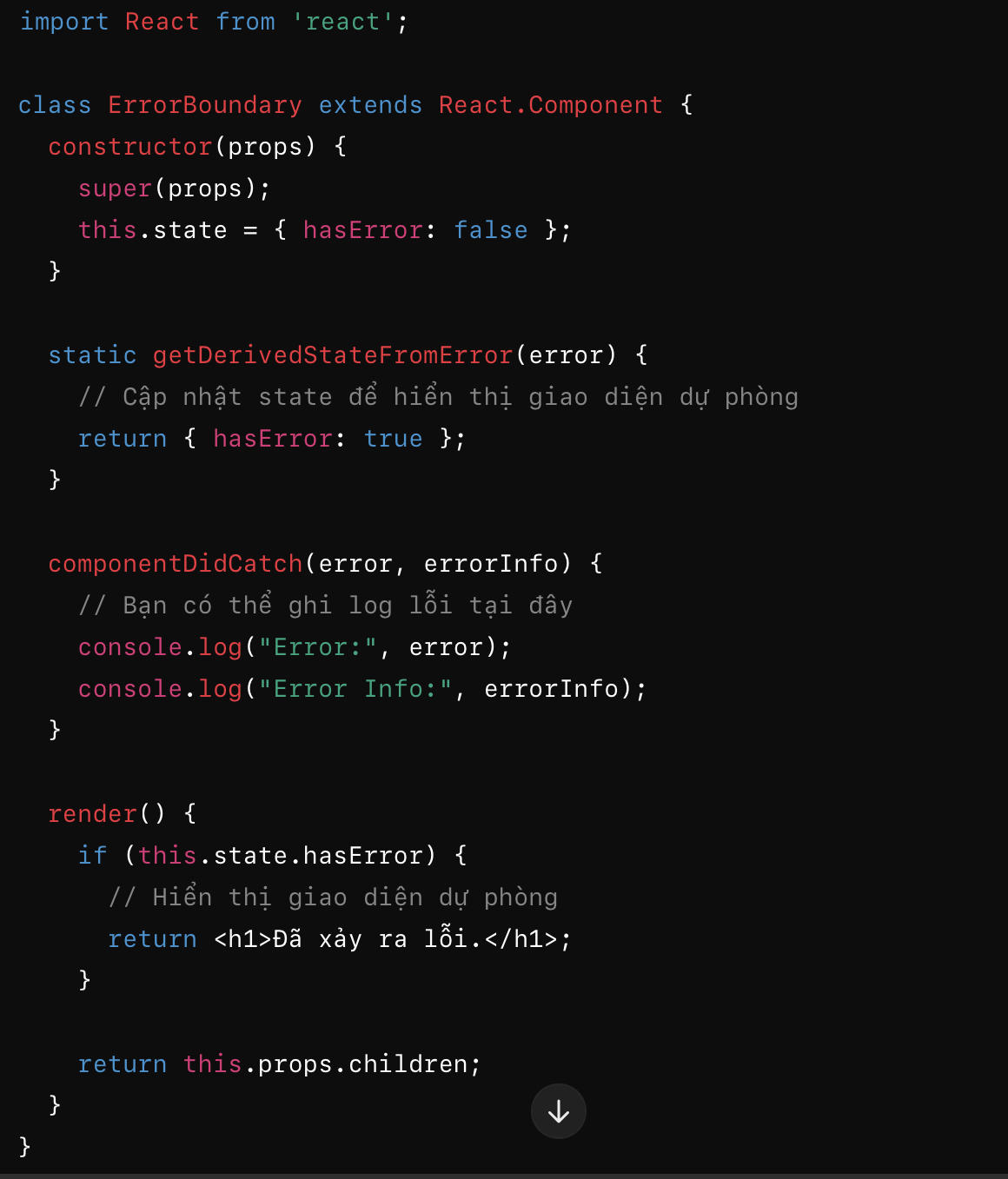
ERROR BOUNDARIES

Là một tính năng cho phép bạn xử lý lỗi trong các component con, ngăn không cho ứng dụng bị "sập" và cung cấp một cách để hiển thị giao diện thay thế khi có lỗi xảy ra.

**Cách hoạt động của Error Boundaries**

* Error Boundaries chỉ có thể được sử dụng trong các class components, không phải trong functional components.
* Bạn có thể tạo một Error Boundary bằng cách tạo một class component và định nghĩa ít nhất một trong hai phương thức: static getDerivedStateFromError() hoặc componentDidCatch().

1. Tạo một Error Boundary



2. Sử dụng Error Boundary



HIGHER-ODER COMPONENT

Higher-Order Components (HOCs) trong React là một pattern cho phép bạn tái sử dụng logic giữa các components. Một HOC là một function nhận vào một component và trả về một component mới, thường được sử dụng để thêm chức năng hoặc chia sẻ logic giữa các components khác nhau.

**Đặc điểm của Higher-Order Components**

* **Tái sử dụng logic**: HOCs giúp bạn tái sử dụng logic mà không cần phải sao chép mã trong nhiều components.
* **Nhận và trả về components**: HOCs luôn nhận vào một component và trả về một component mới.
* **Không làm thay đổi cấu trúc của components**: HOCs chỉ bổ sung chức năng, không thay đổi cách thức hoạt động của component gốc.

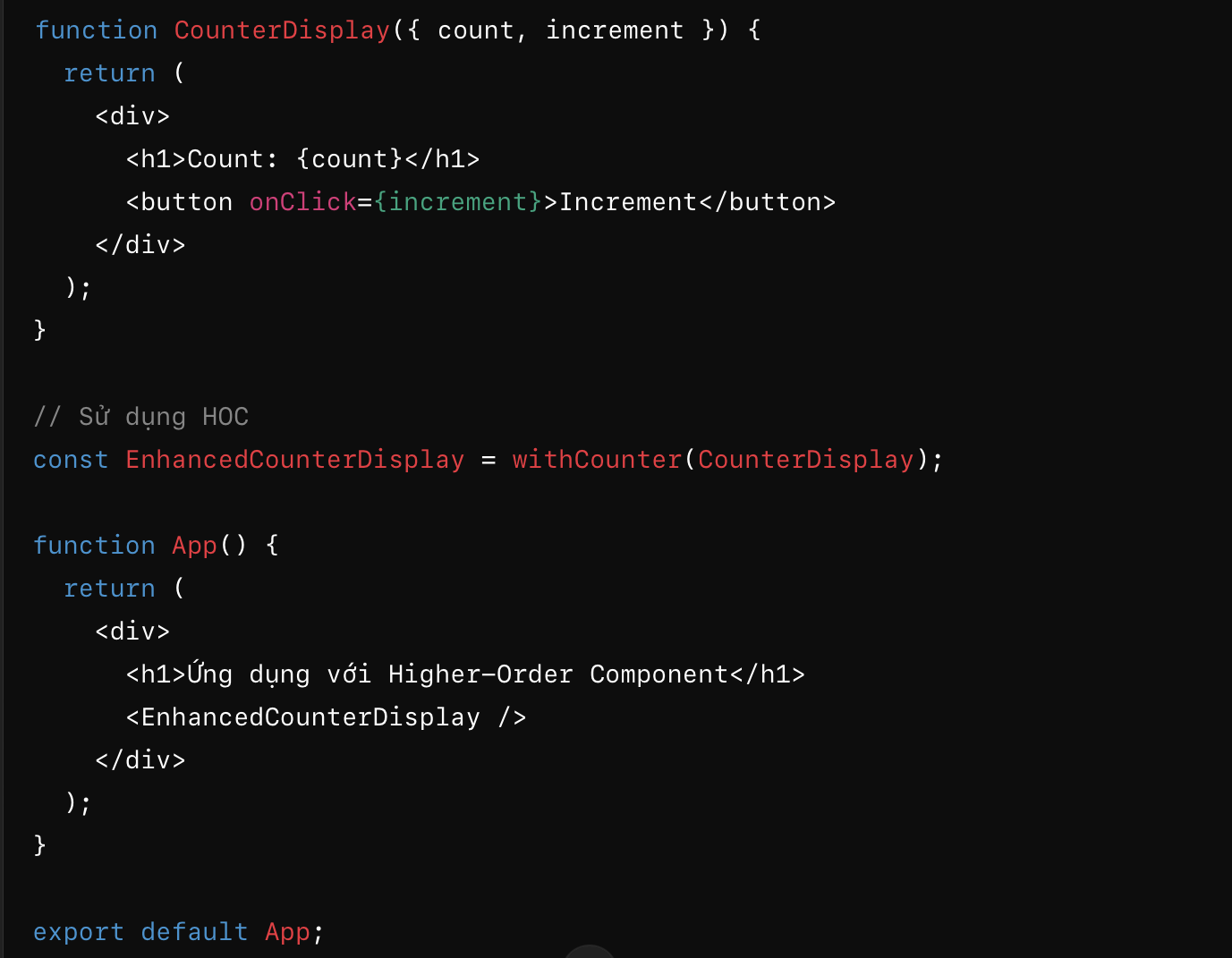
**1. Tạo một Higher-Order Component**

Dưới đây là một ví dụ về một HOC đơn giản, withCounter, để thêm chức năng đếm cho một component.



**2. Sử dụng Higher-Order Component**

Bây giờ, bạn có thể sử dụng HOC withCounter để tạo một component mới có chức năng đếm.



**Các điểm cần lưu ý**

* **Tên Components**: Để dễ dàng theo dõi, bạn nên đặt tên cho HOCs với tiền tố with. Ví dụ: withAuth, withLogging, etc.
* **Tránh vấn đề với props**: Nếu HOC không xử lý props đúng cách, bạn có thể gặp phải vấn đề với props mà component gốc mong muốn.
* **Composition**: Bạn có thể kết hợp nhiều HOCs để tạo ra các components phức tạp hơn. Hãy chắc chắn rằng bạn quản lý các props một cách cẩn thận.

**Một số ví dụ phổ biến về Higher-Order Components**

* **withRouter**: HOC từ React Router để thêm thông tin về router vào component.
* **connect**: HOC từ Redux để kết nối components với state của Redux store.

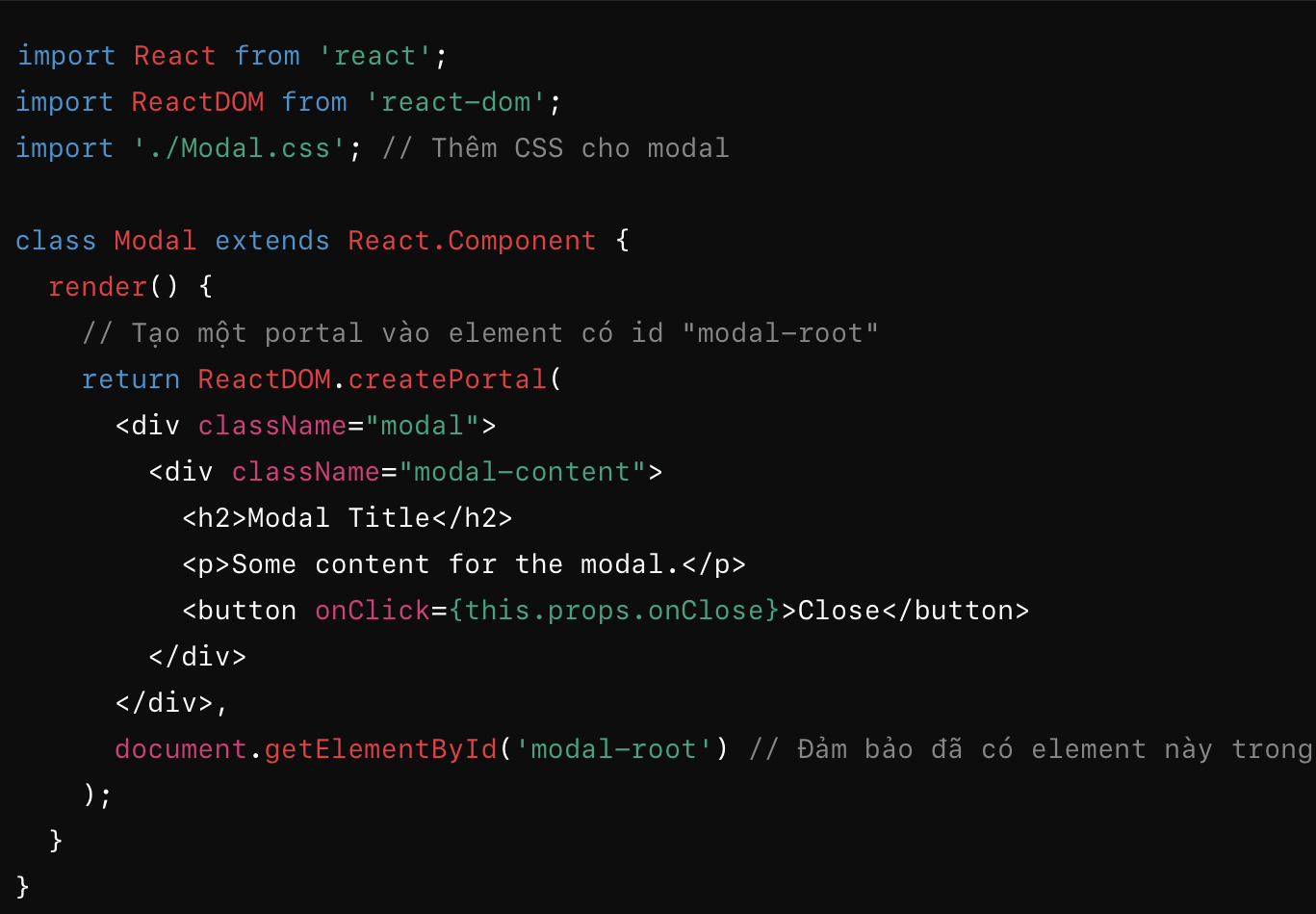
PORTALS

Là một tính năng cho phép bạn render một component con vào một DOM node khác, bên ngoài cấu trúc DOM của component cha.

Portals sử dụng hàm ReactDOM.createPortal để chuyển một component vào một node khác trong DOM. Khi bạn tạo một portal, component đó vẫn sẽ giữ nguyên context và lifecycle của component

cha.

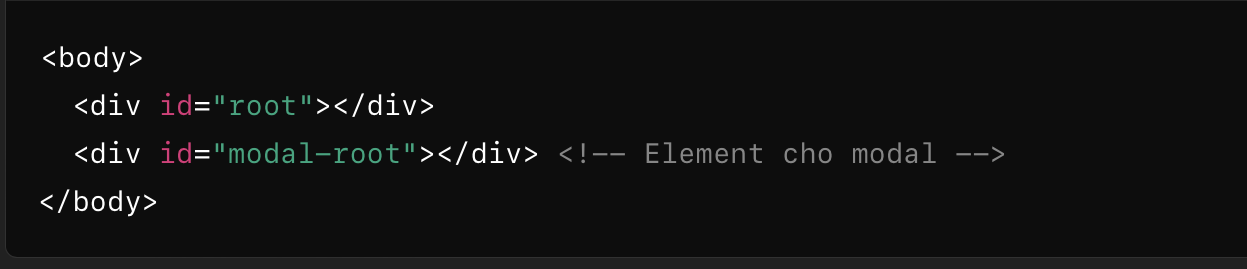
//Tạo portal



//Sử dụng portal



//HTML



RECONCILIATION

**Reconciliation** là quá trình React sử dụng để xác định những thay đổi cần thực hiện khi một component được cập nhật. Nó giúp tối ưu hóa hiệu suất bằng cách chỉ cập nhật những phần tử cần thiết trong DOM.

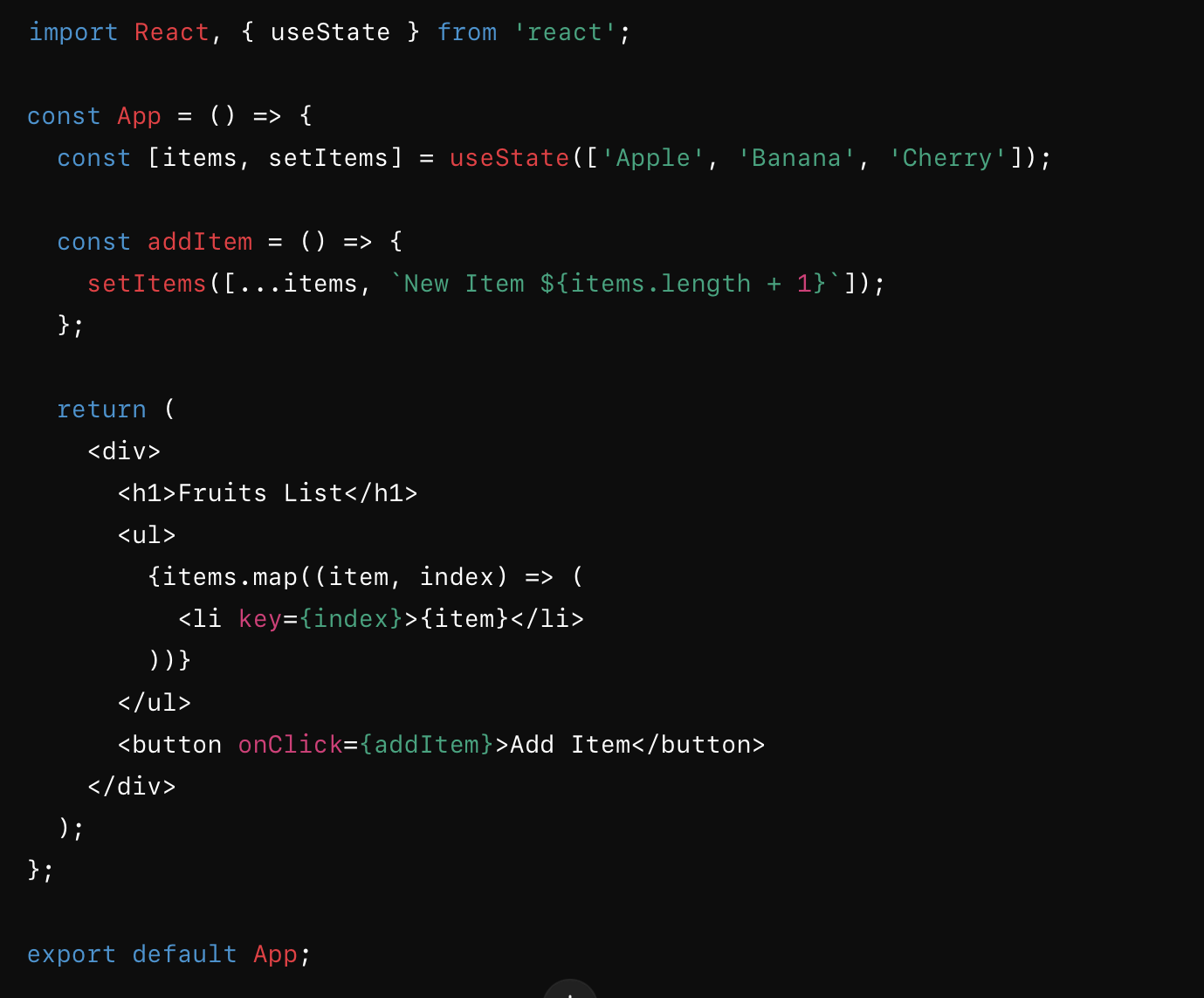
**Các Khái Niệm Chính**

1. **Virtual DOM**: Cấu trúc dữ liệu mà React sử dụng để quản lý cây DOM. Khi có thay đổi, React tạo một cây Virtual DOM mới trước khi cập nhật DOM thực tế.
2. **Diffing Algorithm**: Thuật toán so sánh hai cây Virtual DOM để tìm ra sự khác biệt và xác định các phần tử cần thêm, sửa hoặc xóa.
3. **Keys**: Các thuộc tính duy nhất dùng trong danh sách các component, giúp React theo dõi và tối ưu hóa quá trình diffing.

**Quy trình Reconciliation**

1. **State/Props Thay đổi**: Khi state hoặc props thay đổi, React gọi lại render() để tạo cây Virtual DOM mới.
2. **So sánh với Virtual DOM Hiện tại**: React sử dụng thuật toán diffing để xác định các thay đổi.
3. **Cập nhật DOM Thực**: Chỉ cập nhật những phần tử đã thay đổi.

**Ví dụ**



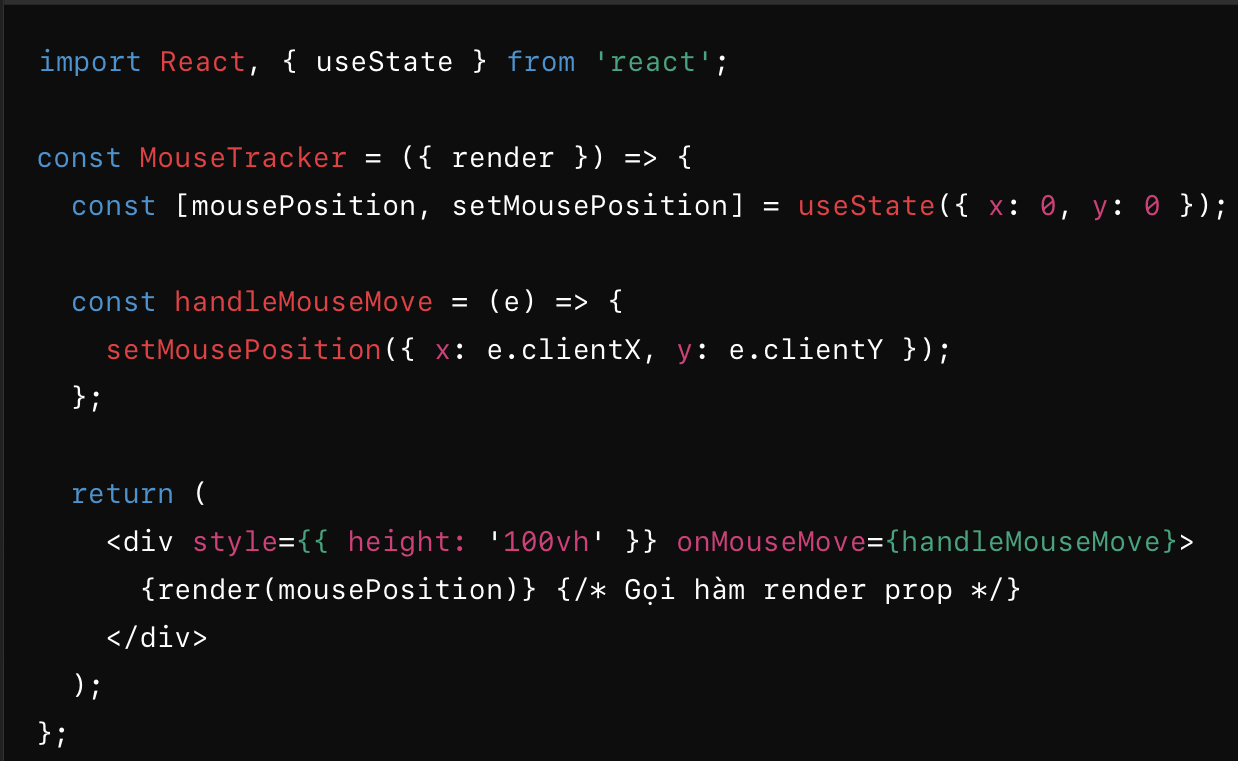
RENDER PROPS

**Render Props** là một pattern trong React cho phép bạn chia sẻ logic giữa các component bằng cách sử dụng một hàm render. Thay vì truyền dữ liệu qua props, bạn truyền một hàm, và hàm đó sẽ trả về component mà bạn muốn render.

**Cách hoạt động**

1. **Component Chứa Logic**: Một component nhận vào một hàm render (render prop) và gọi hàm đó để render nội dung.
2. **Component Sử Dụng Render Prop**: Component này sẽ được truyền logic và có thể sử dụng nó để hiển thị UI.

Tạo Component Chứa Logic



Sử Dụng Component với Render Prop

