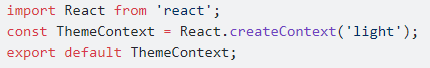
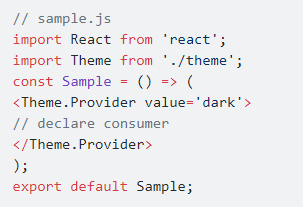
1. Context API:

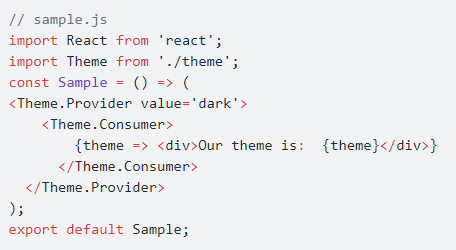
* Tồn tại để không phải truyền dữ liệu một cách thủ công bằng việc dùng props ở tất cả các cấp của component
* Nó chia dữ liệu cho các component khác nhau
* Những thứ thuộc về context là dữ liệu được coi là global như thông tin người dùng, thông tin giỏ hàng, …Vậy lý do dùng context là:
* Dữ liêụ là cần thiết ở nhiều nơi: dữ liệu cần được sử dụng bởi nhiều component như chủ đề, người dùng, giỏ hàng...
* Truyền dữ liệu qua nhiều component: sử dụng context trong trường hợp này là tốt hơn khi bạn muốn chuyển 1 giá trị props thông qua nhiều component.
* Context API bao gồm các khối xây dựng:
* context: đối tượng context là một đối tượng lưu giữ giá trị context hiện tại và có thể được đăng ký.
* provider: là một component của React cung cấp giá trị, nó lấy từ đối tượng context.
* consumer: là một component có thể sử dụng giá trị của provider và có thể hiển thị giá trị.
* Ví dụ:
* Tạo đối tượng context: Dùng React.createContext() để tạo đối tượng context. Truyền cho nó một tham số là giá trị mặc định của context, export để có thể dùng nhiều nơi



* Khai báo provider: Dùng đối tượng context tham chiếu đến Provider



* Khai báo Consumer:



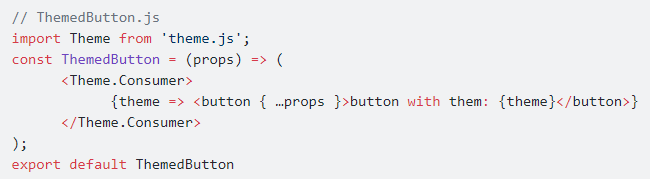
Ở trên, ta thêm consumer dưới dạng đối tượng Theme.Consumer và bên trong định nghĩa một hàm có tham số truyền vào là giá trị của theme.

Vì sao đặt giá trị mặc định trong component Theme?

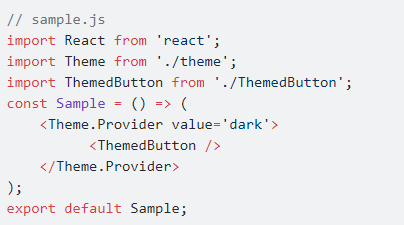
* Giá trị mặc định này sẽ không được dùng nếu khai báo Provider, nhưng nếu thiếu component Provider thì nó sẽ dùng giá trị mặc định, vì vậy đoạn mã trên xuất ra giá trị “dark”, còn đoạn mã dưới thì ra giá trị “light”



* Sử dụng:
* Có thể đưa thành phần Consumer vào component như sau:



* Ở sample.js có thể viết gọn như sau:



Giá trị từ provider đang được truyền qua các props và có thể truy cập thuộc tính theme thông qua Consumer

* Dynamic context:
* Nếu muốn thay đổi giá trị provider thì dùng dynamic context.
* Đặt provider cào bên trong một component và để giá trị của nó phụ thuộc vào state của component như sau:



Nhờ đó có thể thay đổi giá trị provider thông qua thay đổi giá trị state

Ví dụ:



1. Redux

* Là công cụ quản lý trạng thái
* Có thể dùng với bất kỳ khung hoặc thư viện nào của JS
* Khi dùng redux, trạng thái ứng dụng được giữ trong một “store” và mỗi thành phần có thể truy cập bất kỳ trạng thái nào mà nó cần từ store này
* Redux hoạt động rất đơn giản: Có một store trung tâm chứa toàn bộ trạng thái của ứng dụng, mỗi thành phần có thể truy cập trạng thái được lưu trữ mà không phải gửi từ thành phần này sang thành phần khác
* Có 3 phần xây dựng: actions, store, reducers
* Actions:

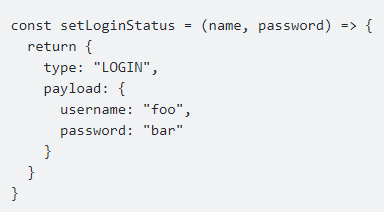
+ Là sự kiện, là cách duy nhất để gửi dữ liệu từ ứng dụng của mình đến store Redux

+ Các hành động được gửi bằng phương thức store.dispatch()

+ Các hành động là các đối tượng Js đơn giản và chúng phải có thuộc tính loại để chỉ ra loại hành động sẽ được thực hiện

+ Phải có “payload” có chứa thông tin cần được xử lý bằng hành động được tạo thông qua Action Creator

+ Ví dụ về Action Creator:



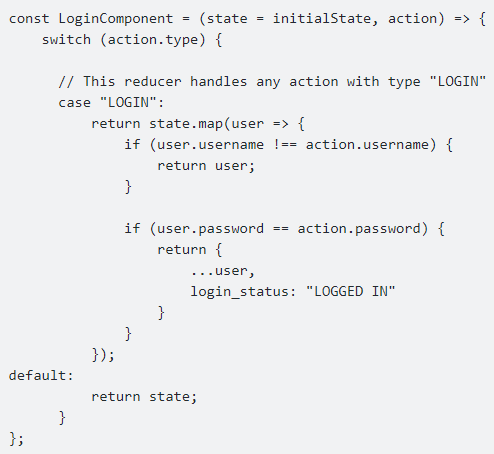
* Reducers:

+ Là hàm thuần túy lấy trạng thái hiện tại cả ứng dụng, thực hiện một hành động và trả về trạng thái mới

+ Lưu trữ dưới dạng đối tượng, xác định trạng thái của ứng dụng thay đổi như thế nào để đáp ứng với hành động được gửi đến “store”

+ Dựa trên hàm “reduce” trong Js, trong đó giá trị được tính từ nhiều giá trị sau khi thực hiện chức năng gọi lại

+ Ví dụ:



+ Không thay đổi dữ liệu trong đối tượng được truyền hoặc thực hiện bất kỳ tác dụng phụ nào trong ứng dụng

+ Cùng một đối tượng, luôn tạo ra cùng một kết quả

* Store:

+ Các store giữ trạng thái ứng dụng

+ Chỉ có một store trong bất kỳ ứng dụng redux nào

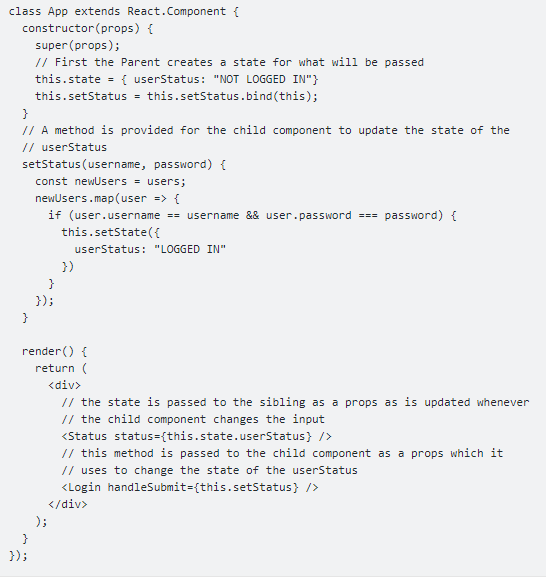
+ Có thể truy cập trạng thái được lưu trữ, cập nhật trạng thái và đăng ký hoặc hủy đăng ký “listeners” thông qua các phương thức trợ giúp

+ Tạo store cho việc đăng nhập

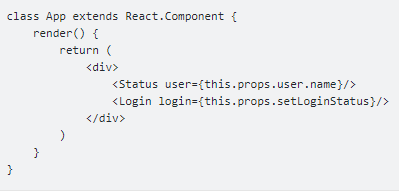


+ Các hành động được thực hiện trên trạng thái luôn trả về một trạng thái mới bên quản lý trạng thái dễ dàng và có thể dự đoán

+ Ví dụ Login trong React:



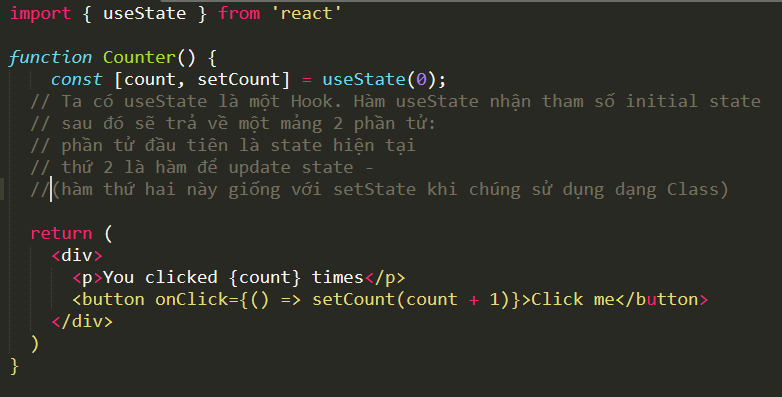
Sau khi dùng Redux:



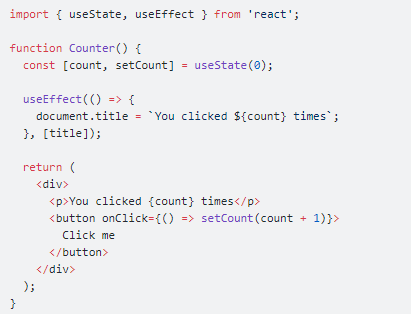
* Với redux, có một trạng thái chung trong store và mỗi thành phần có quyền truy cập vào trạng thái, giúp loại bỏ sự cần thiết phải liên tục chuyển trạng thái từ thành phần này sang thành phần khác

1. Hook

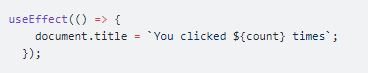
* Là một bổ sung mới trong react 16.8
* Là những hàm cho phép “kết nối” React state và lifecycle vào các components sử dụng hàm
* Có thể dùng state và lifecycle mà không cần dùng es6 class
* React hook giải quyết các vấn đề sau:
* Wrapper hell các component được lồng vào nhau nhiều tạo ra một DOM tree phức tạp
* Component quá lớn
* Sự rắc rối của Lifecycle trong class
* Lợi ích:
* Khiến component trở nên gọn nhẹ hơn
* Giảm số lượng code, dễ tiếp cận
* Cho phép dùng state ngay trong function component
* Sử dụng:
* Ví dụ:



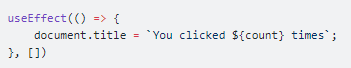
* useEffect – hook này có tác dụng tương tự như componentDidMount, componentDidUpdate và componentWillUnmount nhưng nó chỉ có một hàm duy nhất
* Ví dụ:



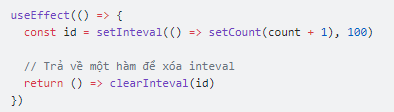
* Effect này được khai báo ngay bên trong component và có quyền truy cập được các state và props. Mặc định, effect sẽ được gọi sau mỗi lần render DOM bao gồm cả lần đầu tiên
* useEffect nhận 2 parameter, đầu tiên là một function nơi chúng ta xử lý các side effect, thứ hai là một array [title], là nơi chứa biến, khi biến này thay đổi thì hook useEffect sẽ được kích hoạt
* Nếu không bỏ array vào hook useEffect thì nó sẽ chạy cùng với mọi lần component chạy function render



* Nếu muốn nó chạy 1 lần sau lần render đầu tiên thì truyền vào tham số thứ 2 của useEffect là một mảng rỗng



* Effect hook có thể trả về một hàm, và hàm này sẽ được gọi để thực hiện tác vụ dọn dẹp, tương tự componentWillUnmount()



* Ngoài ra còn các hook như:

+ useContext

+ useReducer

+ useCallback

+ …

* Một số lưu ý khi làm việc với Hook:
* Trong cùng một component, có thể dùng bao nhiêu useState và useEffect tùy ý nhưng các hook này phải gọi ở trên cùng của function, không được nằm trong vòng lặp, khu vực điều kiện, hay các function con
* Chỉ dùng trong functional component
* Khi dùng useEffect để lấy dữ liệu, cần kiểm tra dữ liệu đã tồn tại hay chưa. Nếu không hàm sẽ gửi request liên tục