1. **JSX :**

- Cú pháp mở rộng cho JavaScript, sử dụng cùng react để mô tả giao diện .

<div>

<h1>Hello {name}!</h1>

<h2>Good to see you here.</h2>

</div>

Đặc điểm của JSX:

* JSX cho phép bạn viết HTML trong JavaScript, nhưng thực chất nó được chuyển đổi thành các lệnh JavaScript React.createElement khi biên dịch.
* Có thể sử dụng biểu thức JavaScript trong JSX bằng cách đặt chúng trong dấu ngoặc nhọn {}
* Có thể viết thuộc tính cho thẻ giống như html , nhưng sẽ khác một chút cú pháp

JSX children: là nội dung nằm giữa thẻ mở và thẻ đóng của một component trong JSX. Đây có thể là văn bản, các phần tử JSX khác, hoặc thậm chí là các biểu thức JavaScript

1. **Component**:

- Cho phép chia giao diện người dùng thành các phần độc lập, có thể tái sử dụng và xem xét từng phần một cách riêng biệt

- Nó giống như các hàm JavaScript, cho phép các đối số đầu vào : goi là props

- Có 2 loại component: class component và function component.

- Nên chia code thành các component, việc có được các thành phần nhỏ có thể tái sử dụng sẽ mang lại các ứng dụng lớn. nếu một phần giao diện người dùng được sử dụng nhiều lần (Button, Panel, Avatar) hoặc bản thân nó đủ phức tạp (App, FeedStory, Comment)thì nên chia thành các component

- Quy tắt viết tên component : Bắt đầu bằng một chữ viết hoa

1. **Props**

Các dạng props có thể truyền vào một component:

* Số, boolean, mảng.
* Có thể chuyển bất kỳ biểu thức JavaScript nào dưới dạng prop, bằng cách bao quanh nó bằng {}:  
  <MyComponent foo={1 + 2 + 3 + 4} />
* Truyền dưới dạng chuỗi ký tự :

<MyComponent message="hello world" />

<MyComponent message={'hello world'} />

* Truyền props là một object , có thể sử dụng Spread Operator để truyền

const props = {firstName: 'Ben', lastName: 'Hector'};

return <Greeting {...props} />;

* Hàm

<SubmitButton onClick={handleSubmit} />

* Component khác:

<WrapperComponent>

<ChildComponent />

</WrapperComponent>

* Phần tử React:

<CustomComponent title={<h1>Hello World</h1>} />

1. **State**

State trong React là một đối tượng quản lý dữ liệu động trong component, giúp phản ánh trạng thái hiện tại của giao diện. Khi state thay đổi, React sẽ re-render lại component để hiển thị dữ liệu mới.

Lifeciycle của một class component gồm 3 giai đoạn:

* Mounting: giai đoạn khi component được khởi tạo và thêm vào DOM lần đầu tiên
* Update: Updating xảy ra khi props hoặc state của component thay đổi, dẫn đến việc re-render component
* Inmounting: Unmounting xảy ra khi component bị loại bỏ khỏi DOM

1. **Xử lý sự kiện**

* Xử lý sự kiện với Reactjs khá giống js , một số khác biệt là : cách đặt tên hàm camelCase thay vì chữ thường , khi viết JSX, truyền vào một hàm để xử lý sự kiện thay vì một chuỗi

UseMemo :

* Cho phép lưu trữ các kết quả tính toán giữa các lần re-render
* Nhận vào 2 đối số : Một hàm tính toán ko đối số , để trả về kết quả muốn tính toàn ; danh sách các thành phần phụ thuộc
* Trong lần render đầu tiên : giá trị bạn nhận được từ useMemo sẽ là kết quả của phép tính của bạn.
* Trong lần render tiếp theo , React sẽ so sánh các phần phụ thuộc với các phần phụ thuộc mà bạn đã chuyển trong lần kết xuất gần đây nhất. Nếu không có phần phụ thuộc nào thay đổi, useMemo sẽ trả về giá trị bạn đã tính toán trước đó. Nếu không, React sẽ chạy lại phép tính của bạn và trả về giá trị mới.

UseCallback :

* Sử dụng để lưu vào bộ nhớ đệm các hàm mà bạn chuyển cho các thành phần con. Nếu ko sử dụng nó có thể khiến rerender lại các phần con một cách không mong muốn
* Hai đối số
  + Hàm muốn ghi nhớ
  + Mảng các phụ thuộc , nếu bất kkif giá trị nào trong mảng thay đổi thì hàm sẽ được tạo lại ; nếu mảng là rỗng thì hàm sẽ ko bị thay đổi sau render

Use Context:

* giúp bạn dễ dàng chia sẻ dữ liệu giữa các component mà không cần phải truyền props qua từng cấp bậc
* tạo context và cung cấp giá trị ở com cha -> sử dụng usecontext trong com con

UseReducer

* cho phép quản lý state phức tạp .useReducer cung cấp một cách để tổ chức logic cập nhật state của bạn trong một function (reducer) và tách biệt các hành động (actions).
* Khi bạn có một cấu trúc component phức tạp với nhiều cấp độ và cần truyền dữ liệu sâu từ cha đến con, useReducer giúp bạn quản lý state một cách có tổ chức và hiệu quả. Thay vì phải truyền props qua nhiều cấp độ, bạn có thể lưu trữ state ở component cha và sử dụng reducer để cập nhật nó.

React-Router:

* Cần làm lại :
* Viết logic redux ở action thay vì reducer .