React.dev

Ngày 5/5/2023

Tóm tắt những gì sẽ học (cơ bản):

Cách tạo và lồng components

Thêm đánh dấu và styles

Thể hiện dữ liệu

Render conditions và lists

Phản ứng lại với các sự kiện và cập nhật lại trên màn hình

Cách chia sẻ data giữa các components

Tạo và lòng components

React được tạo nên từ các components. Một components là một mảnh của UI (User interface) có logic và diện mạo riêng. Một components có thể là nút, hoặc là cả một trang.

React components là Javascript function chả về tags HTML:

Ví dụ:

|  |
| --- |
| function MyButton() {  return (  <button>I'm a button</button>  );  } |

Cách lồng nó trong một components khác:

|  |
| --- |
| export default function MyApp() {  return (  <div>  <h1>Welcome to my app</h1>  <MyButton />  </div>  );  } |

Lưu ý: <MyButton /> bắt đầu với chữ in hoa. Đó là cách nhận biết đó là một React component. React component PHẢI bắt đầu bằng chữ cái in hoa, trong khi HTML tags sẽ là chữ thường

Ví dụ chỉnh chu, ta có:

|  |
| --- |
| function MyButton() {  return (  <button>  I'm a button  </button>  );  }  export default function MyApp() {  return (  <div>  <h1>Welcome to my app</h1>  <MyButton />  </div>  );  } |

Từ khóa `export default` chỉ đỉnh main component trong file.

VIẾT VỚI JSX

JSX mang tính chặt chẽ hơn HTML. Bắt buộc phải đóng tags với ví dụ tương tự như sau: <br />. Một component không thể trả về nhiều JSX tags. Mà phải bọc nó trong một tag cha, như <div> … </div> hoặc 1 tags rỗng <> … </>

Ví dụ:

|  |
| --- |
| function AboutPage() {  return (  <>  <h1>About</h1>  <p>Hello there.<br />How do you do?</p>  </>  );  } |

THÊM CSS

Trong React, để miêu tả một class CSS với từ khóa là className. Nó hoạt động tương từ với thuộc tính class trong HTML:

Ví dụ:

|  |
| --- |
| <img className="avatar" /> |

Sau đó viết CSS trong một file CSS riêng biệt:

|  |
| --- |
| .avatar {  border-radius: 50%;  } |

React không quy định việc thêm CSS files làm sao. Cách đơn giản nhất để làm điều này là thêm vào một tag <link> trong file HTML.

THỂ HIỆN DỮ LIỆU:

JSX cho chúng ta cho HTML vào trong JavaScript. Ngoặc nhọn cho phép ta “trở về” JavaScript để ta có thể nhúng các biến vào trong code của mình và thể hiện nó cho người dùng. Ví dụ:

|  |
| --- |
| return (  <h1>  {user.name}  </h1>  ); |

Hoặc cũng có thể gán nó cho các thuộc tính, nhưng phải sử dụng dấu ngoặc nhọn thay vì dấu hai chấm. Ví dụ, `className = “avatar” ` sử dụng class CSS .avatar, nhưng src={user.imageUrl} đọc giá trị user.imageUrl, sau đó chuyền giá trị đó cho thuộc tính src:

|  |
| --- |
| return (  <img  className="avatar"  src={user.imageUrl}  />  ); |

Ta cũng có thể đặt các biểu thức phức tạp trong dấu ngoặc nhọn của JSX ví dụ:

|  |
| --- |
| const user = {  name: 'Hedy Lamarr',  imageUrl: 'https://i.imgur.com/yXOvdOSs.jpg',  imageSize: 90,  };  export default function Profile() {  return (  <>  <h1>{user.name}</h1>  <img  className="avatar"  src={user.imageUrl}  alt={'Photo of ' + user.name}  style={{  width: user.imageSize,  height: user.imageSize  }}  />  </>  );  } |

Trong ví dụ trên, style ={{}}không phải syntax gì đặc biệt, nó chỉ là một {} object trong một style={}. Ta có thể sử dụng thuộc tính style khi các quy ước về styles phụ thuộc vào các biến của JavaScript.

CONDITIONAL RENDERING

trong react, không có một syntax đặc biệt nào để viết điều kiện (conditions). Thay vào đó, chúng ta sẽ sử dụng một kỹ thuật như khi chúng ta viết JavaScript bình thường. Ví dụ, chúng ta có thể dụng câu lệnh if để làm điều kiện trong JSX:

Ví dụ:

|  |
| --- |
| let content;  if (isLoggedIn) {  content = <AdminPanel />;  } else {  content = <LoginForm />;  }  return (  <div>  {content}  </div>  ); |

Nếu không thích thì ta có thể sử dụng một kỹ thuật khác là Ternary Operator không như if, nó hoạt động như sau:

|  |
| --- |
| <div>  {isLoggedIn ? (<AdminPanel />) : (<LoginForm />) }  </div> |

Khi mà ta công cần nhánh else branch, ta có thể sử dụng viết ngắn gọn bằng logical && syntax:

|  |
| --- |
| <div>  {isLoggedIn && <AdminPanel />}  </div> |

RENDERING LISTS

Có thể sử dụng các tính năng của JavaScript như **for loop** hoặc là **array map() function** để render danh sách của components.  
  
Ví dụ ta có chuổi các sản phẩm:

|  |
| --- |
| const products = [  { title: ‘Cabbage’, id: 1 },  { title: ‘Garlic’, id: 2 },  { title: ‘Apple’, id: 3 },  ]; |

Bên trong component, sử dụng function map() để đổi mảng products sang <li>:

|  |
| --- |
| const listItems = products.map(product =>  <li key={product.id}>  {product.title}  </li>  );  return (  <ul>{listItems}</ul>  ); |

Lưu ý cách mà tag <li> có thuộc tính khóa. Mỗi item trong một list, ta nên truyền cho nó một chuỗi string hoặc 1 số để phân biệt nó với các item chị em. Thông thường, một khóa key thường được lấy ra từ database, như ID trong database của ta. React sử dụng khóa đó để biết chuyện gì sảy ra sau khi nhập, xóa, sắp xếp item.

Ví dụ:

|  |
| --- |
| const products =[  { title: ‘Cabbage’ , isFruit: false, id: 1 },  { title: ‘Garlic’ , isFruit: false, id: 2 },  { title: ‘Apple’ , isFruit: true, id: 3 }  ];  export default function ShopppingList() {  const listItems = products.map( product =>  <li  key={product.id}  style={{  color: product.isFruit ? ‘magenta’ : ‘darkgreen’  }}  >  {product.title}  </li>  );    return(  <ul>{listItems}</ul>  );  } |

PHẢN ỨNG VỚI SỰ KIỆN

Có thể phản ứng lại sự bằng cách miêu tả event handler function bên trong một components bình thường

|  |
| --- |
| function MyButton() {  function handleClick() {  alert('You clicked me!');  }  return (  <button onClick={handleClick}>  Click me  </button>  );  } |

Chú ý cách mà onClick={handleClick} không có ngoặc đơn ở cuối. Không cần phải gọi enet handler function: ta chỉ cần truyền nó xuống thôi. React sẽ gọi event handler khi mà người dùng nhấn vào nút.

CẬP NHẬT LÊN MÀN HÌNH.

Thỉnh thoảng, bạn sẽ muốn component nhớ một số thông tin và hiển thị nó. Ví dụ, bạn muốn đếm số lần mà nút được ấn. Để làm điều này , sử dụng useState().  
  
Đầu tiên, ta cần thêm useState từ React :

|  |
| --- |
| import { useState } from 'react'; |

Giờ cần khai báo state variable bên trong component:

|  |
| --- |
| function MyButton() {  const [count, setCount] = useState(0);  // ... |

Khi sử dụng useState: state hiện tại (count), và function để cập nhật nó (setCount). Ta có thể đặt cho nó bất cứ tên nào, nhưng quy ước thì cứ viết là [something, setSomething].

Lần đầu tiên button được hiển thị, count sẽ là 0 bởi vì ta chuyền 0 vào trong useState(). Khi mà muốn đổi state đi, gọi setCount() và truyền dữ liệu mới cho nó. Ấn nút thì bộ đếm sẽ tăng lên:

|  |
| --- |
| function MyButton() {  const [count, setCount] = useState(0);  function handleClick() {  setCount(count + 1);  }  return (  <button onClick={handleClick}>  Clicked {count} times  </button>  );  } |

React sẽ gọi component function một lần nữa. Lần này, count sẽ là 1. Lặp lại lần nữa ta sẽ được hai và 3, rồi cứ như vậy. Nếu bạn render một component nhiều lần, thì từng cái như vậy sẽ có state riêng của nó.

Ví dụ cụ thể như sau:

|  |
| --- |
| import { useState } from 'react';  export default function MyApp() {  return (  <div>  <h1>Counters that update separately</h1>  <MyButton />  <MyButton />  </div>  );  }  function MyButton() {  const [count, setCount] = useState(0);  function handleClick() {  setCount(count + 1);  }  return (  <button onClick={handleClick}>  Clicked {count} times  </button>  );  } |

Trong ví dụ này, thì mỗi nút được hiển thị trong MyApp() sẽ khi nhớ count state riêng của nó và chỉ nó mà thôi, và không ảnh hưởng đến các nút khác.

**Giới thiệu về Hooks**

Các functions bắt đầu bằng use thường được gọi là Hooks. useState là một trong số đó. Có thể tự viết riêng cho mình một hooks bằng cách kết hợp những cái có sẵn.  
  
Hooks có nhiều hạn chế hơn các functions khác. Ta chỉ có thể gọi Hooks ở trên cùng của components (hoặc từ Hooks khác). nếu muốn sử dụng useState trong một lệnh điều kiện hoặc một vòng lặp, ttrích xuất một component mới và đặ nó vào.

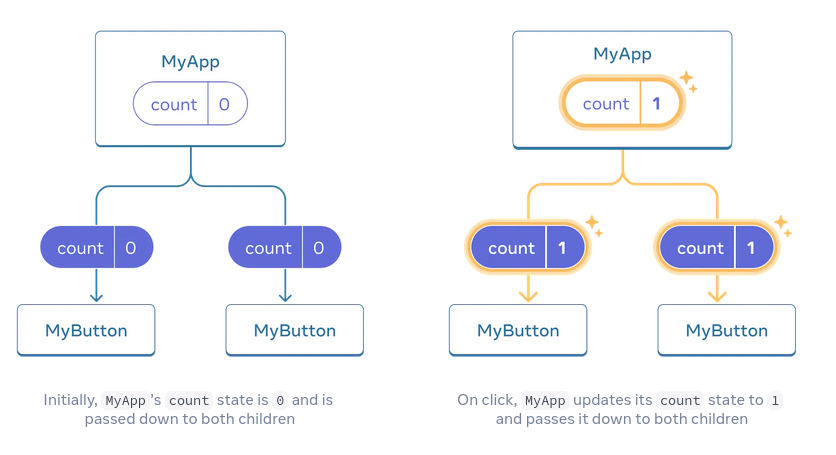
**Chia sẻ data giữa các components**

Trong ví dụ trước, mỗi MyButton có `count` state riêng, và mỗi khi được nhấn, chỉ có `count` của nút nhấn là thay đổi thôi:

Tuy nhên, thỉnh thoảng ta vẫn cần chia sẻ data và cập nhật dữ liệu chung với nhau.

Để cả hai MyButton components hiển thị cùng `count` và cùng tăng hay cập nhật dữ liệu cùng với nhau, ta cần chuyển state từ riêng thành chung, tức chuyển vào nơi chứa tất cả bọn chúng.

Để hiểu rõ hơn thì hình sau sẽ giải thích cặn kẽ:

Và giờ đây mỗi khi nhấn vào bất cứ một nút nào, thì `count` trong MyApp sẽ thay đổi, thứ sẽ thay đổi số lần đếm trong MyButton.

GIải thích bằng code:

Đầu tiên, di chuyên state từ MyButton vô MyApp:

|  |
| --- |
| export default function MyApp() {  const [count, setCount] = useState(0);  function handleClick() {  setCount(count + 1);  }  return (  <div>  <h1>Counters that update separately</h1>  <MyButton />  <MyButton />  </div>  );  }  function MyButton() {  // ... we're moving code from here ...  } |

Sau đó, chuyền state từ MyApp đến mỗi nút. Ta có thể truyền thông tin cho MyButton sử dụng ngoặc nhọn, như cách làm với tag có sẵn như <img>:

|  |
| --- |
| export default function MyApp() {  const [count, setCount] = useState(0);  function handleClick() {  setCount(count + 1);  }  return (  <div>  <h1>Counters that update together</h1>  <MyButton count={count} onClick={handleClick} />  <MyButton count={count} onClick={handleClick} />  </div>  );  } |

Thông tin ta chuyền xuống như thế này gọi là props. Bay giờ MyApp component đã chứa `count` state và `handleClick` thành phần xử lý sự kiện, và truyền cả hai xuống như là props với mổi nút.

Cuối cùng, đổi MyButton để đọc props ta đã truyền xuống từ component cha:

|  |
| --- |
| function MyButton({ count, onClick }) {  return (  <button onClick={onClick}>  Clicked {count} times  </button>  );  } |

Khi ta nhấn nút, trình xử lý onClick kích hoat. Mỗi nút mà có prop onClick được đặt là handleClick function bên trong MyApp, thì code bên trong handleClick sẽ chạy. Code gọi `setCount(count + 1)`, sẽ tăng `count` state variable. một giá trị `count` mới sẽ được truyền xuống gọi là prop truyền xuống mỗi nút, nên tất cả đều hiển thì cùng một giá trị mới. Đây được gọi là “lifting state up” (tạm dịch: Nâng trạng thái.). Bằng cách này, ta có thể chia sẽ dữ liệu giữa các components với nhau.

Ví dụ đầy đủ:

|  |
| --- |
| import { useState } from 'react';  export default function MyApp() {  const [count, setCount] = useState(0);  function handleClick() {  setCount(count + 1);  }  return (  <div>  <h1>Counters that update together</h1>  <MyButton count={count} onClick={handleClick} />  <MyButton count={count} onClick={handleClick} />  </div>  );  }  function MyButton({ count, onClick }) {  return (  <button onClick={onClick}>  Clicked {count} times  </button>  );  } |

**MỘT BÀI TẬP NHỎ ĐỂ HIỂU RÕ HƠN VỀ REACT**

**LẬP TRÌNH TIC-TAC-TOE**

Qua được phần này ta sẽ hiểu được những thành phần cơ bản nhất của React, phần này sẽ chia ra các phần như sau:

+**Setup**

**+Tổng quan:** sơ lươc qua những thứ cơ bản nhất của React là: components, props, và state.

+**Hoàn thành game:** sẽ cho ta biết những kỹ năng thường xuyên được sử dụng trong phát triển ứng dụng React.

**+ Mở rộng:** cho ta nhữnghiểu biết sâu hơn về React.

**Tổng quan:**

Cấu trúc thư muc của một React thông thường sẽ gồm 3 phần trọng yếu sau là:

+App.js: Code trong App.jsx hay App.js tạo ra các component như đã nói trước. Trong React, một component là một mảnh code có khả năng tái sử dụng, có khả năng thể hiện lại một phần của UI(User Interface). Những components được sử dụng để render, quản lý(manage), hoặc cập nhập lại các UI elements trong app. Trong component này ta có:

|  |
| --- |
| export default function Square() {  return <button className="square">X</button>;  } |

Dòng đầu tiên định nghĩa một function tên là Square. Từ khóa export nói rằng function này có thể được truy cập từ bên ngoài file. Từ khóa default nói vói các file khác sử dụng code trong file này biết rằng đây là main function.

Dòng tiếp theo `return` lại một nút. Từ khóa return nghĩa là bất cứ thứ gì đi sau sẽ được trả về là một giá trị cho bất cứ nơi nào gọi nó. <button> là một JSX elements, một JSX element là một tổ hợp của JavaScript code và HTMl tags miêu tả những gì ta sẽ thể hiện. className=”square” là một nút button property hay props, dùng để CSS style của button. X là chử sẽ được thể hiện bên trong button và </button> là tag đóng của JSX element.

+style.css

File css của react. Ví dụ như nó chứa CSS của square trong App.js

+index.js

Trong này có các dòng đáng chú ý sau:

|  |
| --- |
| import { StrictMode } from 'react';  import { createRoot } from 'react-dom/client';  import './styles.css';  import App from './App'; |

Từ dòng 1 - dòng 5:

+React  
+React’s library hay React DOM

+thêm file CSS vào trong React, ở đây có tên style.css

+Components được tạo trong App.js

Các dòng còn lại đem mọi thứ lại với nhau và đưa vào file ở trong index.html/

**Xây dưng bàn cờ.**

Giờ vào file App,jsx.

Hiện tại ta có thể thấy rằng bảng ở đây chỉ có một ô thôi, nhưng ta cần 9 ô. Nếu ta copy paste hết 9 ô vào trong chỗ return để nó trả về 9 ô thì nó sẽ gây lỗi.

|  |
| --- |
| /src/App.js: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? |

React components sẽ chỉ trả về 1 JSX element thôi, vì thế, để trả về nhiều JSX elements thì ta cần đặt nó trong một tag rỗng hay gọi là fragments (<> </>) để bọc lại.

|  |
| --- |
| export default function Square() {  return (  <>  <button className="square">X</button>  <button className="square">X</button>  </>  );  } |

Giờ để làm ra 9 ô ta cần bọc nó trong một div và CSS nó, để nó hiển thị như là lưới trong này.

|  |
| --- |
| export default function Square() {  return (  <>  <div className="board-row">  <button className="square">1</button>  <button className="square">2</button>  <button className="square">3</button>  </div>  <div className="board-row">  <button className="square">4</button>  <button className="square">5</button>  <button className="square">6</button>  </div>  <div className="board-row">  <button className="square">7</button>  <button className="square">8</button>  <button className="square">9</button>  </div>  </>  );  } |

Sau khí đã định nghĩa trong CSS và đặt các div với className là board-row. Giờ coi như là xong bàn cờ.

Nhưng,điều này gây ra một vấn đề rất lớn. Ta đặt tên cái function này là Square chứ có phải là Board đầu, như thế này thì việc tổ chức code sẽ lộn xộn mất. Nên ta cần thay đổi:

Đổi tên function lại. Từ Square thành Board:

|  |
| --- |
| export default function Board() {  return (  <>  <div className="board-row">  <button className="square">1</button>  <button className="square">2</button>  <button className="square">3</button>  </div>  <div className="board-row">  <button className="square">4</button>  <button className="square">5</button>  <button className="square">6</button>  </div>  <div className="board-row">  <button className="square">7</button>  <button className="square">8</button>  <button className="square">9</button>  </div>  </>  );  } |

**Truyền data qua props.**

Tiếp theo ta sẽ cần thay đổi value của một ô từ empty sang “X” hoặc là “O” khi user click vào ô.

Ta đã xây dựng bàn cờ những việc và ta sẽ phải copy-paste tới tận 9 lần và nó khá là lặp lại code. React cho phép ta tạo các component có khả năng tái sử dụng, vì vậy ta cần định nghĩa lại. Ta thêm một function component khác có tên là Square()

|  |
| --- |
| function Square() {  return <button className="square">1</button>;  }  export default function Board() {  // ...  } |

rồi trong function component board ta sẽ update lại như sau:

|  |
| --- |
| export default function Board() {  return (  <>  <div className="board-row">  <Square />  <Square />  <Square />  </div>  <div className="board-row">  <Square />  <Square />  <Square />  </div>  <div className="board-row">  <Square />  <Square />  <Square />  </div>  </>  );  } |

Nhưng nó lại làm điều bất tiện là giờ tất cả những thứ này đều hiển thị số một giống nhau hết cả. Để fix lại, ta sẽ cần chuyền dữ liệu vào mỗi lần gọi thay vì là một trong mỗi square.

|  |
| --- |
| function Square({ value }) {  return <button className="square">{value}</button>;  } |

Lưu ý đặt trong dấu {} để bảo đây là một biến không nó sẽ chuyển thành biến string value bình thường.

Và trong function component Board ta cần truyền dữ liệu vào.

|  |
| --- |
| export default function Board() {  return (  <>  <div className="board-row">  <Square value="1" />  <Square value="2" />  <Square value="3" />  </div>  <div className="board-row">  <Square value="4" />  <Square value="5" />  <Square value="6" />  </div>  <div className="board-row">  <Square value="7" />  <Square value="8" />  <Square value="9" />  </div>  </>  );  } |

**Tạo ra function có thể tương tác.**

Bây giờ thay vì hiển thị 1,2,3,4,…. thì ta cần nó hiển thị X hoặc là O mỗi lần ta click. Ta cần khai báo một biến tên là handleClick bên trong Square. Sau đó, thêm onClick vào torng props của button JSX element:

|  |
| --- |
| function Square({ value }) {  function handleClick() {  console.log('clicked!');  }  return (  <button  className="square"  onClick={handleClick}  >  {value}  </button>  );  } |

Hiện tại thì khi ta click vào ta sẽ thấy 1 log hiên ra với nội dung là “clicked!” bên trong browse’s console. Để mở trong đây ta sẽ cần ấn tổ hợp Ctrl+Shift+J.

Bước tiếp theo, ta sẽ cần Square component nhớ là nó đã được click rồi. Để component nhớ được thì ta cần sử dụng state.

React cung cấp cho ta một function đặc biệt là useState() cho phép các component khi nhớ trạng thái của nó.

Để sử dụng useState thì ta cần, import useState vào đầu file. Xóa prop value trong Square component đi. Thay vào đó, ta cần thêm 1 dòng mới vào Suare gọi là useState. Cho nó trả về một state variable gọi là value:

|  |
| --- |
| import { useState } from 'react';  function Square() {  const [value, setValue] = useState(null);  function handleClick() {  //... |

value thì chứa value và setValue là function được sử dụng để thay đổi value. Giá trị null được chuyền vào trong useState thì là một giá trị ban đầu được gán vào trong state variable, nói đơn giản thì value hiện tại đang là null.

Khi mà Square component không nhận prop value nữa thì, ta xóa value đi:

|  |
| --- |
| export default function Board() {  return (  <>  <div className="board-row">  <Square />  <Square />  <Square />  </div>  <div className="board-row">  <Square />  <Square />  <Square />  </div>  <div className="board-row">  <Square />  <Square />  <Square />  </div>  </>  );  } |

Bây giờ ta có thể thay đổi nó thành giá trị X khi click. thay thế console.log(“clicked!”); bằng setValue(‘X’); . Bầy giờ Suare function component Square thế này:

|  |
| --- |
| function Square() {  const [value, setValue] = useState(null);  function handleClick() {  setValue('X');  }  return (  <button  className="square"  onClick={handleClick}  >  {value}  </button>  ); |

Bằng cách gọi set function khi mỗi lần click, React sẽ re-render lại square mỗi lần <button> được clicked. Sau khí update, Square value sẽ là X, mỗi lần click thì X sẽ hiện lên.

Mỗi Square có state riêng của chính nó: cho nên value được chứa trong mỗi Square cũng là riêng biệt với những nơi khác. Khi ta gọi một set function trong một component, thì React sẽ tự động update các components con bên trong nó luôn.

Bây giờ, mỗi một Square component sẽ chứa một phần của game. để kiểm tra xem ai win game này ta cần biết trạng thái của từng ô một, hay gọi ngôn ngữ trong React là biết state của 9 Square components.

Ta có thể làm bằng cách chứa game’s state trong component cha là Board thay vì ở trong Square. Board component có thể nói Square biết cái gì sẽ được hiển thị bằng cách chuyền prop, giống như cách ta chuyền số lúc nãy cho từng ô <Square />

Để thu data từ nhiều con, hoặc là ta có 2 component con giao tiếp với nhau, khai báo một shared state trong component cha của cả hai thì tốt nhất. Component cha có thể truyền state xuống cho các con bằng props. Như thế sẽ giữ cho components con đồng bộ với nhau và với components cha nữa.

|  |
| --- |
| export default function Board() {  const [squares, setSquares] = useState(Array(9).fill(null));  return (  // ...  );  } |

Array(9).fill(null) tạo ra một array với 9 phần tử và đặt mỗi cái là null.

|  |
| --- |
| export default function Board() {  const [squares, setSquares] = useState(Array(9).fill(null));  return (  <>  <div className="board-row">  <Square value={squares[0]} />  <Square value={squares[1]} />  <Square value={squares[2]} />  </div>  <div className="board-row">  <Square value={squares[3]} />  <Square value={squares[4]} />  <Square value={squares[5]} />  </div>  <div className="board-row">  <Square value={squares[6]} />  <Square value={squares[7]} />  <Square value={squares[8]} />  </div>  </>  );  } |

Ta sẽ sửa hết component và nhận value prop từ Board component. Vì thế ta cũng không cần mấy thứ bên trên của Square function nữa, nên là xóa hết đi và Square() chỉ còn là như vầy thôi:

|  |
| --- |
| function Square({value}) {  return <button className="square">{value}</button>;  } |

Để mỗi ô giờ có thế nhận là X, O, hay là null.

Tiếp theo, ta sẽ cần nó hiện X hoặc O khi được nhấn. Board component giờ chứa các ô, ta sẽ cần 1 cách để Square có thể update Board state. Khí mà, state thì nó private với component khai báo nó, ta không thể update Board state trực tiếp qua Square.

Thay vào đó, ta sẽ cần truyền 1 function xuống từ Board component đến Square component, và Square sẽ gọi function đó khi square được nhấn.

|  |
| --- |
| function Square({ value }) {  return (  <button className="square" onClick={onSquareClick}>  {value}  </button>  );  } |

Tiếp theo, ta sẽ cần thêm onSquareClick function vào Square component props:

|  |
| --- |
| function Square({ value, onSquareClick }) {  return (  <button className="square" onClick={onSquareClick}>  {value}  </button>  );  } |

Giờ ta có thể truyền onSquareClick prop đến Board component, đặt tên nó là handleClick. Để sử dụng handleClick ta cần truyền 1 function vào onSquareClick prop.

|  |
| --- |
| export default function Board() {  const [squares, setSquares] = useState(Array(9).fill(null));  return (  <>  <div className="board-row">  <Square value={squares[0]} onSquareClick={handleClick} />  //...  );  } |

Cuối cùng ta khai báo handleClick function bên trong Board component để cập nhất lại squares array ở trong board’s state:

|  |
| --- |
| export default function Board() {  const [squares, setSquares] = useState(Array(9).fill(null));    function handleClick() {  const nextSquares = squares.slice();  nextSquares[0] = "X";  setSquares(nextSquares);  }  return (  // ...  )  } |

handleClick function tạo một copy của squares array (nextSquares) với JavaScript slice(). Sau đó thì, handleClickupdate lại nextSquares array và thêm X vào Square (ví dụ ở đây là [0] index). Gọi setSquares function để cho React biết là state của component đã bị thay đổi. Nó sẽ kích hoạt việc React re-render lại component đã sử dụng square state (Board) cũng như là component con nó.

Giờ ta có thế thêm X vào rồi đó nhưng tiếc cái là chỉ thêm vào mỗi cái đầu tiên. Việc này là do handleCick function là được “bắt buộc” update index [0] thôi. Để update những ô khác. Thêm đối số i vào handleClick function, đổi index từ 0 thành i.

|  |
| --- |
| export default function Board() {  const [squares, setSquares] = useState(Array(9).fill(null));    function handleClick(i) {  const nextSquares = squares.slice();  nextSquares[i] = "X";  setSquares(nextSquares);  }  return (  // ...  )  } |

Tiếp theo, ta thay đổi onSquareClick prop thành handleClick(0):

|  |
| --- |
| <Square value={squares[0]} onSquareClick={handleClick(0)} /> |

Thì nó sẽ xảy ra lỗi.

Khi handleClick(0) được gọi trong khi nó đang là một phần của Board() component, mà handleClick(0) làm thay đổi đi state của Board() component bằng cách gọi setSquares, dẫn đến toàn bộ board component lại bị re-render. Rồi khi làm vậy nó lại chạy tiếp handleClick(0) lần nữa, dẫn đến là bị infinite loop.  
  
Khi truyền nó xuống như là prop thì nó lại khiến cho function này chạy ngay lập tức, sau đó nó liên tục re-render lại khiến cho bị infinite loop.   
  
Có 2 cách sửa:  
+Tạo một function handleFirstSuareClick gọi handleClick(0), một function handleSecondSquareClick gọi handleClick(1), và cứ vậy đến 9 cái thì thôi. Sau đó truyền nó xuống như là một prop onSquareClick={handleFirstSquareClick}.

+Cách 2 thì ta dùng arrow function bằng cách:

|  |
| --- |
| export default function Board() {  // ...  return (  <>  <div className="board-row">  <Square value={squares[0]} onSquareClick={() => handleClick(0)} />  // ...  );  } |

Làm vậy liên tục với 9 cái:

|  |
| --- |
| export default function Board() {  // ...  return (  <>  <div className="board-row">  <Square value={squares[0]} onSquareClick={() => handleClick(0)} />  <Square value={squares[1]} onSquareClick={() => handleClick(1)} />  <Square value={squares[2]} onSquareClick={() => handleClick(2)} />  </div>  <div className="board-row">  <Square value={squares[3]} onSquareClick={() => handleClick(3)} />  <Square value={squares[4]} onSquareClick={() => handleClick(4)} />  <Square value={squares[5]} onSquareClick={() => handleClick(5)} />  </div>  <div className="board-row">  <Square value={squares[6]} onSquareClick={() => handleClick(6)} />  <Square value={squares[7]} onSquareClick={() => handleClick(7)} />  <Square value={squares[8]} onSquareClick={() => handleClick(8)} />  </div>  </>  );  }; |

Code đầy đủ hiện tại:

|  |
| --- |
| import { useState } from 'react';  function Square({ value, onSquareClick }) {  return (  <button className="square" onClick={onSquareClick}>  {value}  </button>  );  }  export default function Board() {  const [squares, setSquares] = useState(Array(9).fill(null));  function handleClick(i) {  const nextSquares = squares.slice();  nextSquares[i] = 'X';  setSquares(nextSquares);  }  return (  <>  <div className="board-row">  <Square value={squares[0]} onSquareClick={() => handleClick(0)} />  <Square value={squares[1]} onSquareClick={() => handleClick(1)} />  <Square value={squares[2]} onSquareClick={() => handleClick(2)} />  </div>  <div className="board-row">  <Square value={squares[3]} onSquareClick={() => handleClick(3)} />  <Square value={squares[4]} onSquareClick={() => handleClick(4)} />  <Square value={squares[5]} onSquareClick={() => handleClick(5)} />  </div>  <div className="board-row">  <Square value={squares[6]} onSquareClick={() => handleClick(6)} />  <Square value={squares[7]} onSquareClick={() => handleClick(7)} />  <Square value={squares[8]} onSquareClick={() => handleClick(8)} />  </div>  </>  );  } |

Giờ thì xử lý trạng thái đã ở trong Board() component, component cha là Board truyền prop xuống cho component con là Square. Khi click vào Square, thì component con Square sẽ hỏi conponent cha update lại state của bàn cờ. Khi mà Board’state thay đổi, thì cả component cha và tất cả các con đều sẽ re-render lại tự động. Để state ở trong component Board sẽ giúp component quyết định được ai là người thắng cuộc.

Recap:

1.Click vào ô trái trên cùng sẽ chạy function mà button nhận được từ onClick prop từ Square. Square component nhận tiếp prop onSquareClick từ Board. Board thì khai báo và chạy trức tiếp trong JSX và gọi handClick với đối số là 0.

2. handleClick sử dụng đối số (0) upate lại element đầu tiên trong squares arrray lại từ null thành X.

3. squares state của Board component được update, và Board và toàn bộ con của nó dươc re-render. từ đó thay đổi value prop của Square component với index là 0 đổi tự null thành X.

Note:

DOM <button> onclick aattribute là component built-in. Custom components như là Square, thì tên gì ta tự đặt. Ta cũng có thể đặt bất cứ tên gì cho các prop của Square và Board, code sẽ vẫn hoặt động. Trong React, khuyến khích nên đặt tên onSomething names cho prop thể hiện cho các sự kiện và handleSomething cho các function định nghĩa cho các function handle sự kiện.

Tại sao tính bất biến (immutability) quan trọng.

Nhớ lại trong handleClick, ta gọi .slice(0 để tạo một copy của square array thay vì chỉnh sửa trực tiếp array hiện có. Để giải thích tại sao thì ta cần thảo luận vì tính bất biến và tại sao tính bất biến lại quan trọng để học.

Thông thường ta có 2 cách để tiếp cận , thay đổi dữ liệu. Cách đầu tiên là tthay đổi trực tiếp giá trị của data. Hai là thay thế bằng một bản copy của data nhưng có chứa dữ liệu mà ta muốn nó được thay đổi.

ví dụ:

|  |
| --- |
| const squares = [null, null, null, null, null, null, null, null, null];  squares[0] = 'X';  // Now `squares` is ["X", null, null, null, null, null, null, null, null] |

|  |
| --- |
| const squares = [null, null, null, null, null, null, null, null, null];  const nextSquares = ['X', null, null, null, null, null, null, null, null];  // Now `squares` is unchanged, but `nextSquares` first element is 'X' rather than `null` |

Kết quả là như nhau nhưng việc không thay đổi data trực tiếp có những lỡi ích riêng.

Tính bất biến khiến cho những tính năng (features) phức tạp dễ dàng được áp dụng hơn. Ví dụ như lát nữa sẽ làm một tính năng tua lại ván đấu ở nhưng thời điểm nhất định. Nói ngắn gọn, đây là một tính năng undo và redo một hành động nào đó. Điều này là một yêu cầu bình thường đối với bất kỳ một app nào. Việc tránh sử dụng data trực tiếp sẽ cho ta cơ hội để sử dụng data đó cho các mục đích sau này.

Một lợi ích khác là. Mặc định, tất cả các component con đều được re-render một cách tự động khi mà state của component cha thay đổi. Điều này bao gồm cả việc kể cả các components mà không được tác động cũng bị re-render luôn. Mặc dù việc re-render như vậy sẽ không được những user chú ý, ta vẫn muốn bỏ qua vài công đoạn re-render để đảm bảo cho performance tốt. Tính bất biến khiến chó nó việc so sánh data có thay đổi hay không.

Turn đấu.

Giờ thì ta đã đánh dấu được cho X, nhưng vẫn con O nữa.

Đặt X là người đi đầu tiên. Đặt thêm môt state nữa để theo dõi xem đây có phải lượt của X không bằng cách:

|  |
| --- |
| function Board() {  const [xIsNext, setXIsNext] = useState(true);  const [squares, setSquares] = useState(Array(9).fill(null));  // ...  } |

Mỗi lần 1 người chơi đánh một nước, thì xIsNext (boolean) sẽ bị đảo ngược lại, để kiểm tra xem ai đánh tiếp, game’s state sẽ được saved lại. Ta sẽ update Board’s handleClick function để đảo dữ liệu của xIsNext lại:

|  |
| --- |
| export default function Board() {  const [xIsNext, setXIsNext] = useState(true);  const [squares, setSquares] = useState(Array(9).fill(null));  function handleClick(i) {  const nextSquares = squares.slice();  if (xIsNext) {  nextSquares[i] = "X";  } else {  nextSquares[i] = "O";  }  setSquares(nextSquares);  setXIsNext(!xIsNext);  }  return (  //...  );  } |

Khi ta click vào một ô khác thì nó hoạt động bình thường, nhưng khi nhấn vào cùng một ô thì dữ liệu của ô đó sẽ bị viết đè lên (overwritten)

Để fix lại thì ta chỉ cần kiểm tra xem ô đó đã chứa dữ liệu chưa. Nếu đã có ta return về luôn.

|  |
| --- |
| function handleClick(i) {  if (squares[i]) {  return;  }  const nextSquares = squares.slice();  //...  } |

Giờ thì ta có thể thêm X và O mà không bị viết đè dữ liệu nữa, code đầ đủ sẽ như thế này:

|  |
| --- |
| mport { useState } from 'react';  function Square({value, onSquareClick}) {  return (  <button className="square" onClick={onSquareClick}>  {value}  </button>  );  }  export default function Board() {  const [xIsNext, setXIsNext] = useState(true);  const [squares, setSquares] = useState(Array(9).fill(null));  function handleClick(i) {  if (squares[i]) {  return;  }  const nextSquares = squares.slice();  if (xIsNext) {  nextSquares[i] = 'X';  } else {  nextSquares[i] = 'O';  }  setSquares(nextSquares);  setXIsNext(!xIsNext);  }  return (  <>  <div className="board-row">  <Square value={squares[0]} onSquareClick={() => handleClick(0)} />  <Square value={squares[1]} onSquareClick={() => handleClick(1)} />  <Square value={squares[2]} onSquareClick={() => handleClick(2)} />  </div>  <div className="board-row">  <Square value={squares[3]} onSquareClick={() => handleClick(3)} />  <Square value={squares[4]} onSquareClick={() => handleClick(4)} />  <Square value={squares[5]} onSquareClick={() => handleClick(5)} />  </div>  <div className="board-row">  <Square value={squares[6]} onSquareClick={() => handleClick(6)} />  <Square value={squares[7]} onSquareClick={() => handleClick(7)} />  <Square value={squares[8]} onSquareClick={() => handleClick(8)} />  </div>  </>  );  } |

Tiếp theo ta cần xác định xem ai là người thắng.

Để xác định xem ai là người thắng thì ta cần thêm một function component nữa để xác định điều này.

|  |
| --- |
| export default function Board() {  //...  }  function calculateWinner(squares) {  const lines = [  [0, 1, 2],  [3, 4, 5],  [6, 7, 8],  [0, 3, 6],  [1, 4, 7],  [2, 5, 8],  [0, 4, 8],  [2, 4, 6]  ];  for (let i = 0; i < lines.length; i++) {  const [a, b, c] = lines[i];  if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {  return squares[a];  }  }  return null;  } |

Lưu ý:

Không quan trọng việc ta khai báo function component calculateWinner trước hay sau Board. Nhưng ta nên đặt nó ở cuối vì lúc đó ta không phải lăn lên lăn xuống để chỉnh sửa component nữa.

giờ chỉnh sửa ở function handleClick của Board nếu một player đã thắng rồi thì, thì ta không cần phải đánh thêm nữa, không cần cập nhật state của component mà return function trước.

|  |
| --- |
| function handleClick(i) {  if (squares[i] || calculateWinner(squares)) {  return;  }  const nextSquares = squares.slice();  //...  } |

Tiếp theo thì chỉ cần hiển thị trạng thái xem cho user biết ai là người thắng cuộc.

|  |
| --- |
| export default function Board() {  // ...  const winner = calculateWinner(squares);  let status;  if (winner) {  status = "Winner: " + winner;  } else {  status = "Next player: " + (xIsNext ? "X" : "O");  }  return (  <>  <div className="status">{status}</div>  <div className="board-row">  // ...  )  } |

|  |
| --- |
| import { useState } from 'react';  function Square({value, onSquareClick}) {  return (  <button className="square" onClick={onSquareClick}>  {value}  </button>  );  }  export default function Board() {  const [xIsNext, setXIsNext] = useState(true);  const [squares, setSquares] = useState(Array(9).fill(null));  function handleClick(i) {  if (calculateWinner(squares) || squares[i]) {  return;  }  const nextSquares = squares.slice();  if (xIsNext) {  nextSquares[i] = 'X';  } else {  nextSquares[i] = 'O';  }  setSquares(nextSquares);  setXIsNext(!xIsNext);  }  const winner = calculateWinner(squares);  let status;  if (winner) {  status = 'Winner: ' + winner;  } else {  status = 'Next player: ' + (xIsNext ? 'X' : 'O');  }  return (  <>  <div className="status">{status}</div>  <div className="board-row">  <Square value={squares[0]} onSquareClick={() => handleClick(0)} />  <Square value={squares[1]} onSquareClick={() => handleClick(1)} />  <Square value={squares[2]} onSquareClick={() => handleClick(2)} />  </div>  <div className="board-row">  <Square value={squares[3]} onSquareClick={() => handleClick(3)} />  <Square value={squares[4]} onSquareClick={() => handleClick(4)} />  <Square value={squares[5]} onSquareClick={() => handleClick(5)} />  </div>  <div className="board-row">  <Square value={squares[6]} onSquareClick={() => handleClick(6)} />  <Square value={squares[7]} onSquareClick={() => handleClick(7)} />  <Square value={squares[8]} onSquareClick={() => handleClick(8)} />  </div>  </>  );  }  function calculateWinner(squares) {  const lines = [  [0, 1, 2],  [3, 4, 5],  [6, 7, 8],  [0, 3, 6],  [1, 4, 7],  [2, 5, 8],  [0, 4, 8],  [2, 4, 6],  ];  for (let i = 0; i < lines.length; i++) {  const [a, b, c] = lines[i];  if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {  return squares[a];  }  }  return null;  } |