



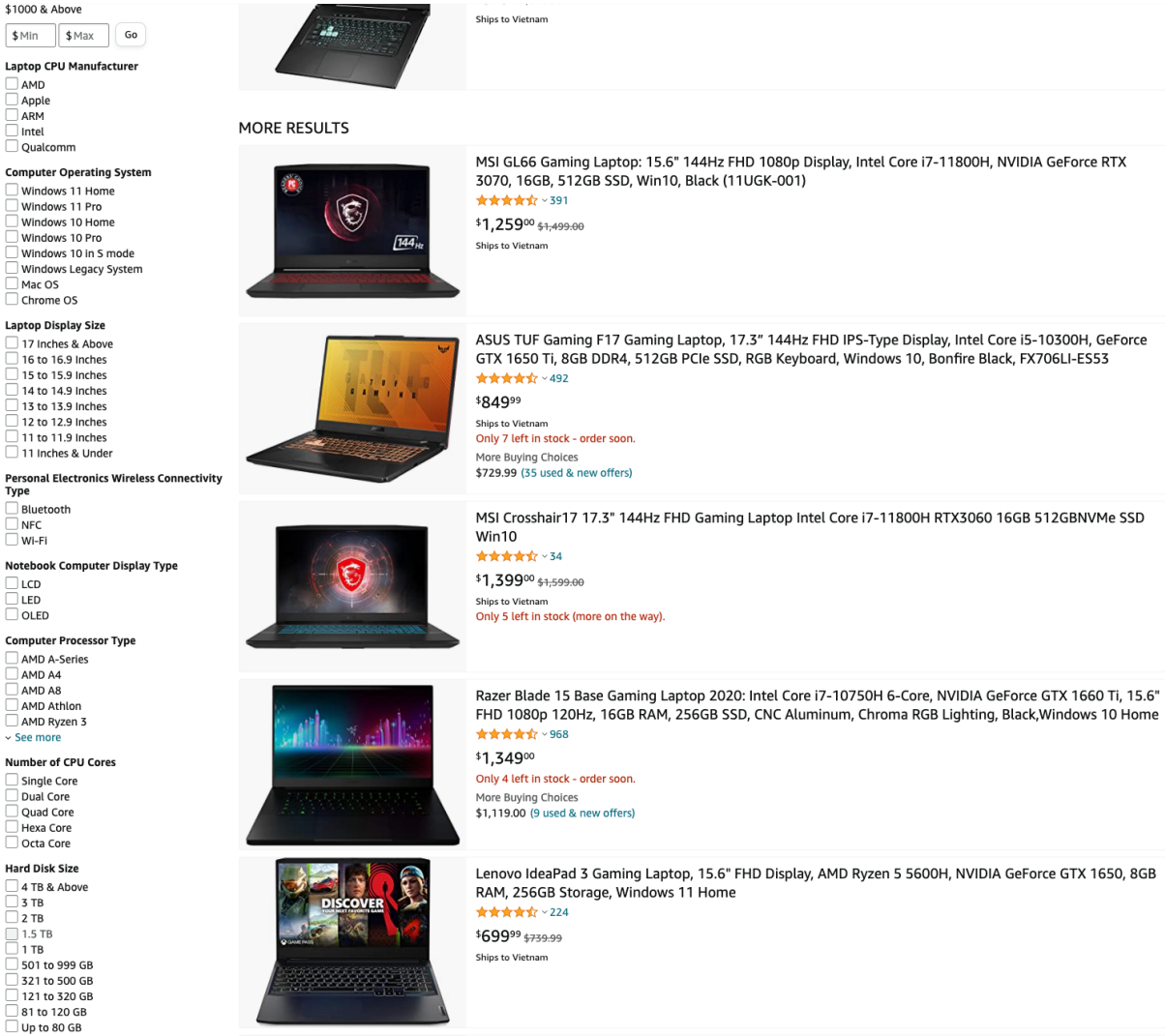
Lesson 7: List & conditional rendering

🕒 Created March 22, 2022 9:55 AM

🏷️ Tags Empty

📌 Related to Untitle... Empty

💡 Xử lý với mảng dữ liệu, render danh sách động với React và JSX



1. Nhắc lại về các function trong array

Để làm việc với dữ liệu dạng mảng trong React, trước tiên chúng ta cần nhắc lại một chút về các function trong array.

```
arr.map((item) => { return newItem })
```

map là một function được sử dụng với mục đích nhằm biến đổi giá trị của các phần tử trong mảng thành các giá trị tương ứng theo một công thức định sẵn. Function **map** trả về một giá trị mảng sau khi các phần tử đã được biến đổi.

```
const arr = [1, 2, 3] const newArr = arr.map(item => {return item * 2})
console.log(newArr) // [2, 4, 6]
```

```
arr.filter((item) => { return boolean })
```

`filter` là một function được sử dụng với mục đích lọc các phần tử từ mảng ban đầu theo một điều kiện định sẵn. Function `filter` cũng trả về một mảng với các giá trị thoả mãn điều kiện lọc. Function filter không làm ảnh hưởng tới các phần tử trong mảng ban đầu

```
const arr = [1, 2, 3] const newArr = arr.filter(item => { return item % 2 === 0 } ) console.log(newArr) // [2] console.log(arr) // [1, 2, 3]
```

2. Render danh sách với React

Danh sách cố định các thẻ `li`

Trong rất nhiều trường hợp thực tế, chúng ta cần làm việc với mảng. Xét ví dụ sau:

```
import {useState} from 'react' const TodoList = () => { const [todoItems, setTodoItems] = useState(["homework", "shopping"]) return ( <div> <button>Add</button> <ul> <li>{todoItems[0]}</li> <li>{todoItems[1]}</li> </ul> </div> ) }
```

Có 2 phần tử trong mảng `todoItem` ở ví dụ trên. Tuy nhiên trong thực tế, chúng ta có thể có nhiều hơn hai phần tử. Người dùng có thể thêm, sửa, xoá các phần tử nằm bên trong `todoItems`. Vì vậy, cần cách khác để có thể render toàn bộ danh sách nằm bên trong `todoItems`

Danh sách các thẻ `li` tương ứng với `todoItems`

JSX có thể render giá trị là một mảng. Ví dụ trên có thể sửa thành như sau:

```
... return ( <div> <button>Add</button> <ul> { [<li>{todoItems[0]}</li>, <li>{todoItems[1]}</li>] } </ul> </div> ) ...
```

React vẫn có thể render cho chúng ta 2 thẻ `` tương tự như ở trên

Lúc này, thay vì hard-coded danh sách 2 phần tử có thể được render như ở trên, chúng ta có thể tạo ra một mảng các phần tử `` tương ứng với các giá trị bên trong `todoItems`.

```
import {useState} from 'react' const TodoList = () => { const [todoItems, setTodoItems] = useState(["homework", "shopping"]) const addItem = () => { setTodoItems(prev => { return [...prev, "new todo item"] }) } const todoItemLi = [] todoItems.forEach(item => { todoItemLi.push(<li>{item}</li>) }) return ( <div> <button onClick={addItem}>Add</button> <ul> {todoItemLi} </ul> </div> ) }
```

Lúc này, giá trị nằm của `todoItemLi` chính là một mảng các thẻ `li`. Nếu chúng ta thay đổi giá trị của `todoItems`, giá trị `todoItemLi` cũng sẽ thay đổi tương ứng, dẫn tới giao diện cũng sẽ được cập nhật.

Sử dụng `map` để đơn giản hoá

Trong ví dụ trên, có thể thấy từ mỗi một phần tử bên trong `todoItems`, ta biến chúng thành một phần tử tương ứng nằm trong `todoItemLi`. Như vậy ta có thể sử dụng hàm `map` trong array, thay vì phải sử dụng vòng lặp `for` truyền thống. `todoItemLi` sẽ trở thành:

```
const todoItemLi = todoItems.map(item => { return <li>{item}</li> })
```

Như vậy, ứng dụng trên có thể được sửa thành như sau:

```
import {useState} from 'react' const TodoList = () => { const [todoItems, setTodoItems] = useState(["homework", "shopping"]) const addItem = () => { setTodoItems(prev => { return [...prev, "new todo item"] }) } const todoItemLi = todoItems.map(item => { return <li>{item}</li> }) return ( <div> <button onClick={addItem}>Add</button> <ul> {todoItemLi} </ul> </div> ) }
```



Trong thực tế thì người ta không thường đặt một biến giống như `todoItemLi` ở trên.

Thay vào đó, để ngắn gọn, chúng ta có thể sử dụng `todoItems.map(...)` ngay trực tiếp bên trong JSX.

3. Key là gì

`key` là một giá trị props đặc biệt trong React. Key được sử dụng làm định danh cho các phần tử trong một mảng các phần tử component. `key` cho phép React dễ dàng nhận biết được phần tử nào bị thay đổi, phần tử nào được thêm hoặc xóa đi trong mảng các component của React thông qua quá trình **reconciliation**.

Các component của React nên nhận một giá trị `key` là duy nhất, so sánh với các phần tử trong cùng mảng đó. `key` phải được set cho thẻ HTML ngoài cùng của các phần tử component.

```
return ( <div> <button onClick={addItem}>Add</button> <ul> {todoItems.map((item, idx) => { return ( <li key={idx}> <span>{item}</span> </li> )}} </ul> </div> )
```

Với ví dụ trên, `key` cần phải được đặt vào thẻ `` thay vì `` , vì thẻ `` là thẻ ngoài cùng của mỗi phần tử JSX trong `map` .

`key` cần là duy nhất giữa các Node Sibling và nên ổn định

Các thuộc tính key cần phải duy nhất để React có thể cập nhật chính xác các component tương ứng. Nếu `key` props bị thiếu hoặc bị trùng, ứng dụng có thể sẽ gặp các lỗi khó hiểu. Xét ví dụ sau khi không sử dụng `key`

```
import React from "react"; const App = () => { const [hobbies, setHobbies] = React.useState([ { id: 1, hobby: "🎮" }, { id: 2, hobby: "📺" } ]); const deleteHobby = hobbyId => { const updatedhobbies = hobbies.filter(item => item.id !== hobbyId); setHobbies(updatedhobbies); }; return ( <div> <h1>Rate your hobbies !</h1> {hobbies.map((item, i) => ( <li> I <select> <option>likes</option> <option>loves</option> </select> {item.hobby} <button onClick={() => deleteHobby(item.id)}>X</button> </li> ))} </div> ); }
```


Ban đầu, chúng ta thay đổi giá trị `<select>` tương ứng với hobby có id = 1 thành `loves` . Sau đó, chúng ta xóa hobby có id = 1 đó đi, ta sẽ thấy giá trị `<select>` bên trong hobby có id = 2 sẽ chuyển thành “loves”, mặc dù chưa hề set nó.

Không nên sử dụng giá trị index của vòng lặp để làm `key` . Vì các giá trị index có thể bị thay đổi khi chúng ta thực hiện các thao tác trên mảng. Xét ví dụ sau:

```
import { useState } from "react"; const generateId = () =>
Math.floor(Math.random() * 10000); export default function App() { const
[inputs, setInputs] = useState([]); const append = () => { setInputs((prev) =>
{ return [...prev, generateId()]; }); }; const prepend = () => {
setInputs((prev) => { return [generateId(), ...prev]; }); }; return ( <div
className="App"> <button onClick={append}>Append</button> <button onClick=
{prepend}>Prepend</button> {inputs.map((input, idx) => { return ( <li key=
{idx}> <label> {input} <input /> </label> </li> ); })} </div> ); }
```

Giá trị của các input hoàn toàn không có vấn đề gì nếu chúng ta sử dụng “append”. Tuy nhiên, khi “prepend”, các giá trị input không còn đúng vị trí của nó nữa

Khi chúng ta thêm phần tử vào trước danh sách, nó sẽ làm index từ vị trí đó trở đi bị thay đổi. Và React không còn cập nhật giao diện một cách chính xác được nữa

 Trong thực tế, các phần tử của mảng nên có một trường duy nhất làm định danh cho chúng, thường là các trường ID được xử lý bởi database. Nếu trong trường hợp cần tự tạo một ID cho các phần tử, nên sử dụng các thư viện bên ngoài để đảm bảo giá trị định danh là duy nhất

Tham khảo: <https://www.npmjs.com/package/uuid>

4. Render theo điều kiện

Trong ứng dụng React, chúng ta cần tùy biến giao diện tùy vào giá trị của các state hoặc props ở thời điểm đó. React cho phép chúng ta thử hiện các lệnh rẽ nhánh để đạt được kết quả mong muốn.

Cấu trúc `if – else` thông thường

```
import {useState} from 'react' const App = () => { const [isLoggedIn,
setIsLoggedIn] = useState(false) let button = null if (isLoggedIn) { button =
<button onClick={() => {setIsLoggedIn(false)}}>Logout</button> } else { button
= <button onClick={() => {setIsLoggedIn(true)}}>Login</button> } return (
<div> {button} </div> ) }
```

Cấu trúc với ternary operator

Ternary operator (toán tử 3 ngôi) là một toán tử đặc biệt, cho phép chúng ta thực hiện phép toán logic và trả ra kết quả với một câu lệnh

```
import {useState} from 'react' const App = () => { const [isLoggedIn,
setIsLoggedIn] = useState(false) return ( <div> {isLoggedIn ? <button onClick=
{() => {setIsLoggedIn(false)}}>Logout</button> : <button onClick={() =>
{setIsLoggedIn(true)}}>Login</button> } </div> ) }
```

Ternary operator có thể được áp dụng với cả các giá trị props hoặc styling bên trong React.

```
import {useState} from 'react' const App = () => { const [isLoggedIn,
setIsLoggedIn] = useState(false) const toggleLogin = () => {
setIsLoggedIn(prev => {return !prev}) } return ( <div> <button style=
{{backgroundColor: isLoggedIn ? "yellow" : "red"}} onClick={toggleLogin} >
{isLoggedIn ? "Logout" : "Login"} </button> </div> ) }
```

Màu sắc của button sẽ thay đổi tùy thuộc vào trạng thái của `isLoggedIn`

Inline với toán tử `&&`

```
import {useState} from 'react' const App = () => { const [isLoggedIn,
setIsLoggedIn] = useState(false) return ( <div> {!isLoggedIn &&
<button>Login</button> } </div> ) }
```