ReactJS

# Giới thiệu

## Giới thiệu

React Js là một thư viện viết bằng javascript, dùng để xây dựng giao diện người dùng (UI). React được sử dụng rộng rãi và có hệ sinh thái đa dạng phong phú. UI tất nhiên là quan trọng, nhưng không phải là tất cả. Để phát triển ứng dụng hoàn chỉnh, một mình React Js là không làm được tất cả, bạn sẽ cần thêm:

* Server side language: để xử lý logic và lưu trữ dữ liệu trên server.
* HTML/CSS nếu bạn làm ứng dụng web.
* Flux/Redux?: là một kiến trúc giúp bạn tổ chức code rõ ràng và sạch sẽ.
* Objective C: nếu bạn sử dụng React để xây dựng app cho iOS

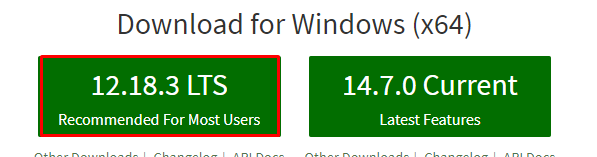
## Cài đặt môi trường

### Các công cụ

* **React react detector**: Giúp ta xác định được trang web đang sử dụng React



* **NodeJS**: Cài đặt NodeJS



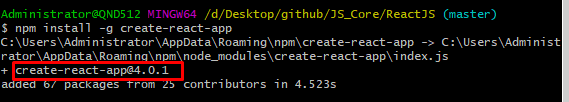
* **Yarn**: Là lựa chọn thay thế cho npm. (Có thể sử dụng thêm)
* **IDE:** Công cụ viết code (Subline Text, Visual Studio Code)

### Khởi tạo Project

* Document React: <https://reactjs.org/docs/getting-started.html>

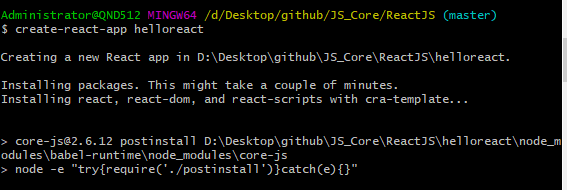
**Bước 1**: Cài đặt create react app

Cài đặt bằng câu lệnh: npm install -g create-react-app



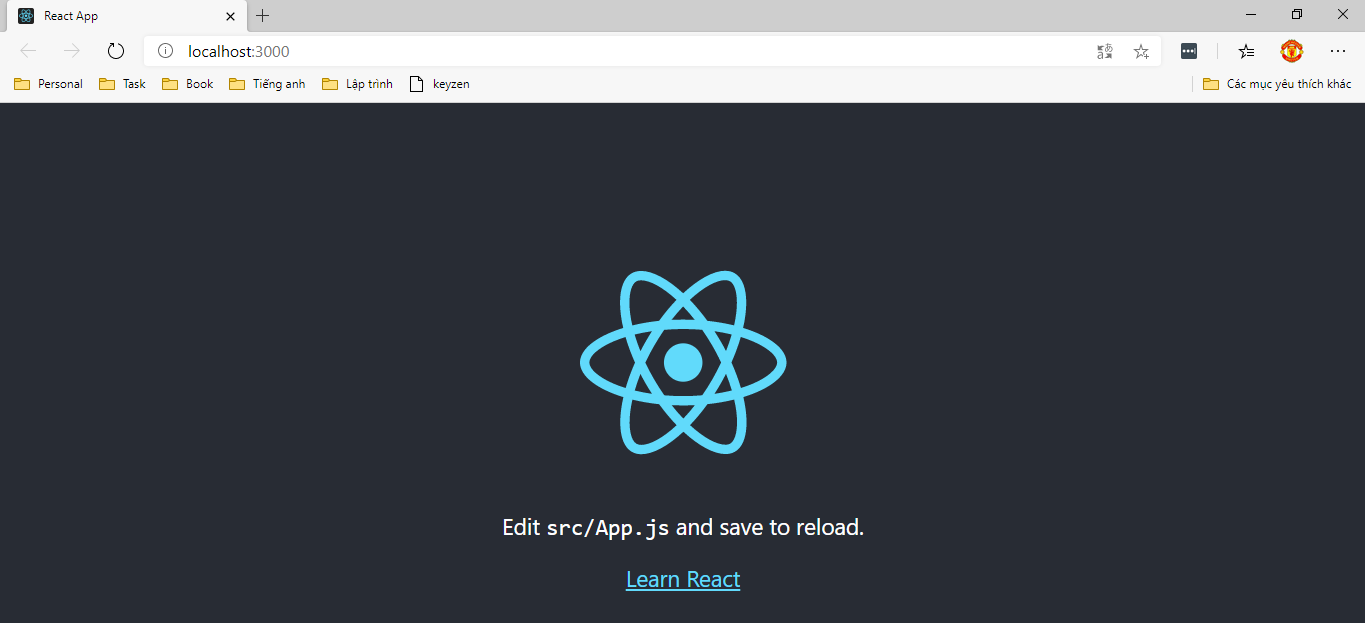
**Bước 2:** Tạo mới một project

* Tạo một thư mục sau đó vào thư mục đó bật cmd lên mà nhập lệnh tạo project
* Câu lệnh tạo: create-react-app ten\_project



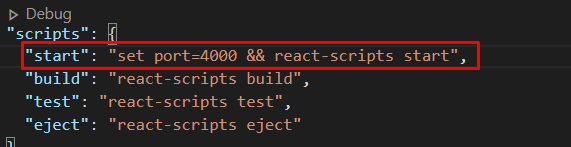
**Bước 3:** Chạy ứng dụng

* Vào trong project vừa tạo: cd helloreact
* Khởi chạy ứng dụng : npm start



**Bước 4:** Đổi port React

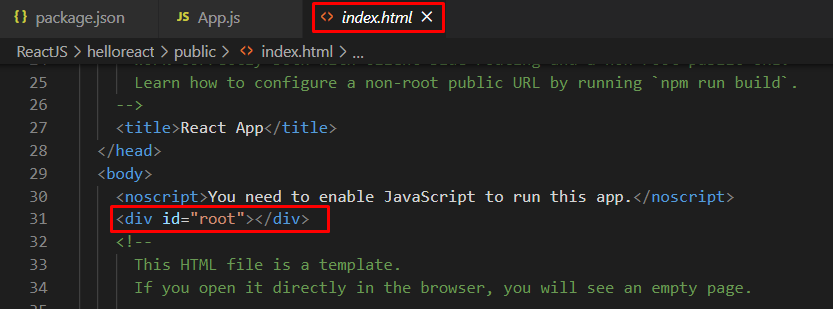
* Đổi port tại file package.json trong project
* Câu lệnh set port: “start” : ”set port=port\_muon\_thay\_doi && react-scripts start”

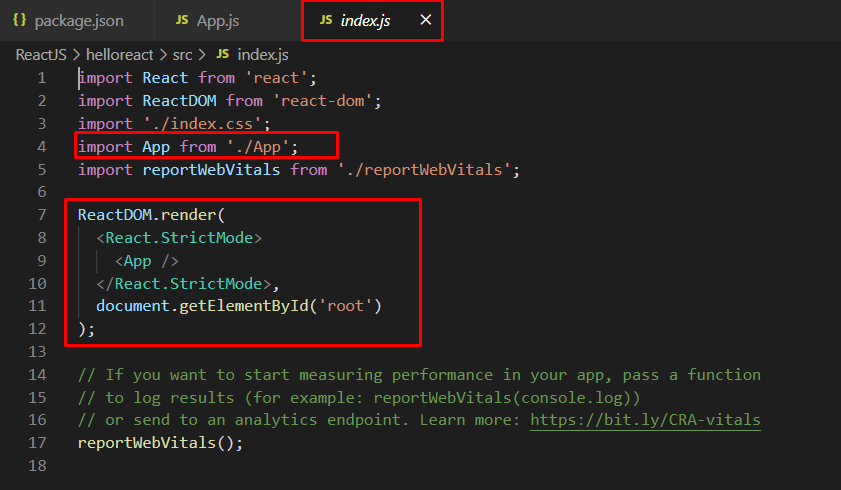


## Cấu trúc thư mục

* **Package.json**: Nơi khai báo thông tin của project
* **Node\_modules:** Nơi chứa các thư viện của ReactJS
* **Public/index.html:** Là file chạy đầu tiên. Trong index.html có:  Đây là nơi react được đổ vào.
* **Src:** Là nơi làm việc của lập trình viên. Ta có thể thay đổi và mở rộng code ở thư mục này.

## Đường đi của các file

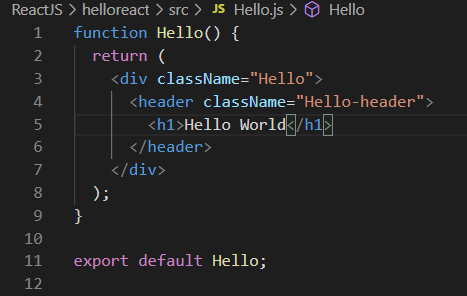




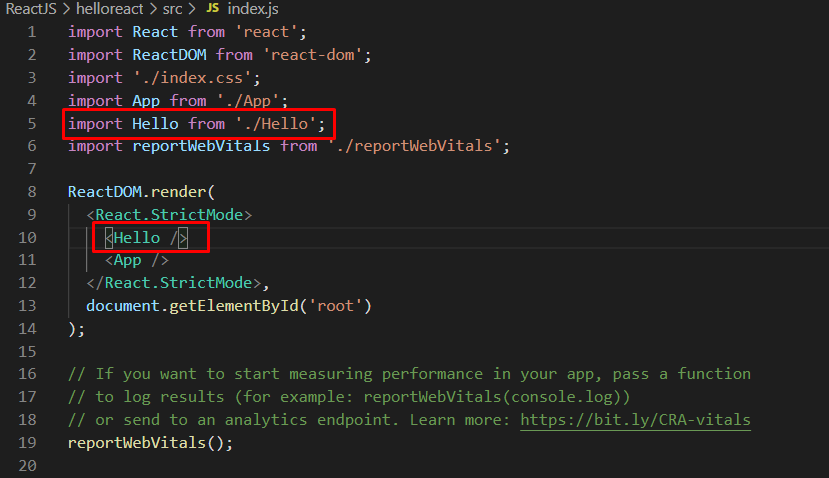
* Trong file index.js có phần render App vào component có Id là root trong index.html bằng câu lệnh document.getElementById(‘root)
* Và file App được import từ ./App và chính là file App.js trong thư mục src

## Hello World

**Bước 1**: Tạo Hello.js trong thư mục src tương tự như App.js



**Bước 2**: Import component Hello vừa mới tạo vào file index.js



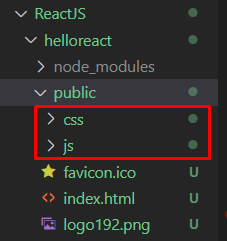
* Trong React.StrictMode sẽ là thứ tự các component mà chúng ta muốn sắp xếp

# Component

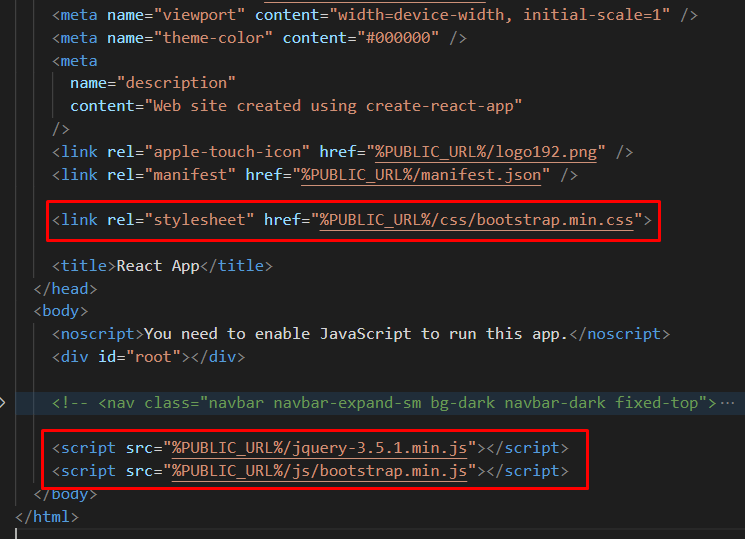
## Nhúng boostrap và các thư viện bên ngoài vào React

**Bước 1**: Download boostrap và Jquery

**Bước 2**: Giải nén và copy các thư mục css, js, và jquery vào thư mục public trong project

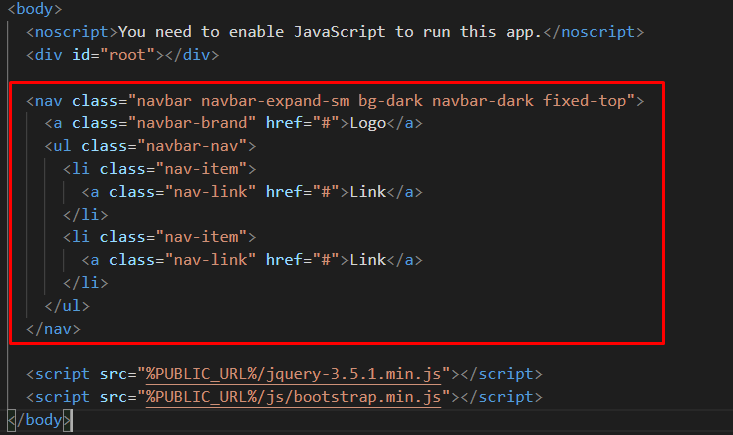


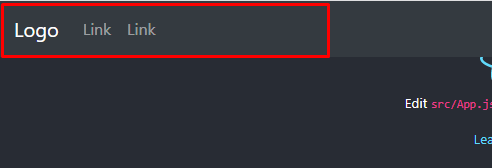
**Bước 3**: Import vào file index.html trong thư mục public



* Với %PUBLIC\_URL% là đường dẫn tới thư mục public trong project

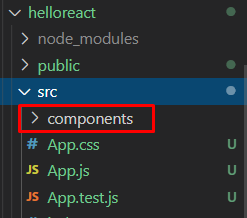
**Bước 4**: Kiểm tra xem project đã nhận boostrap chưa



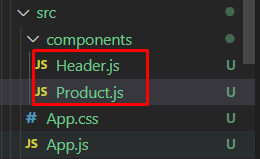


## Tạo một components

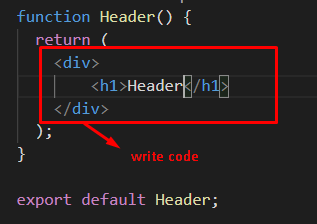
**Bước 1**: Tạo folder chứa các components



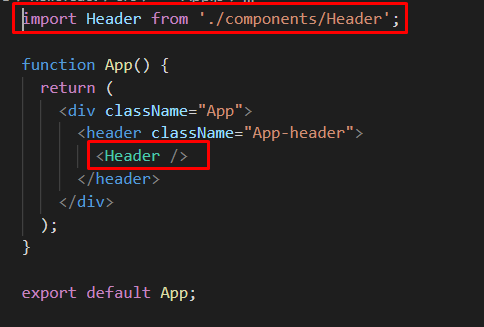
**Bước 2**: Tạo component trong folder vừa tạo



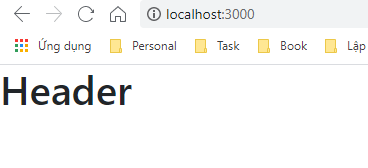
Với Header.js



Lồng Header.js vào App.js: Trong App.js



KQ:

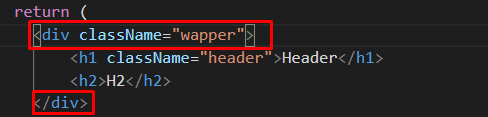


Chú ý:

* Trong component thì class sẽ được viết thành className

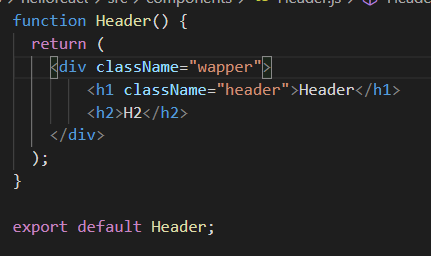


* Trong một component thì luôn luôn phải có một thẻ ngoài cùng bao bọc toàn bộ các thẻ con khác

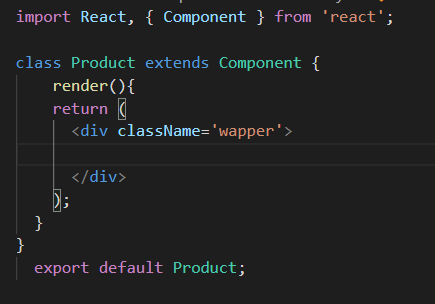


## Class component và Function component

* Function component



* Class component (ES6)

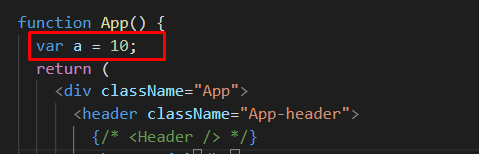


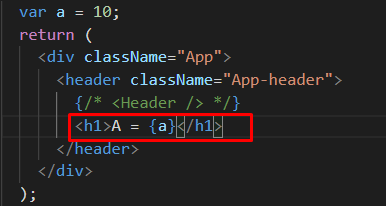
|  |  |
| --- | --- |
| Class component | Function component |
|  |  |
|  |  |
|  |  |

## JSX

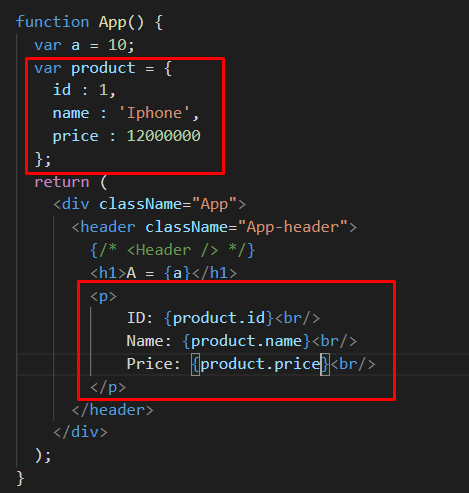
* JSX (viết tắt của cụm từ JavaScript XML) là một loại cú pháp mở rộng dành cho ngôn ngữ JavaScript viết theo kiểu XML. JSX cung cấp cú pháp ngọt (syntactic sugar) để thay cho câu lệnh React.createElement() trong React.
* Chuyển đổi từ HTLM sang JSX: <https://magic.reactjs.net/htmltojsx.htm>

### Biến và hiển thị biến trong JSX

* Khai báo biến: 
* Hiển thị biến ra ngoài màn hình:



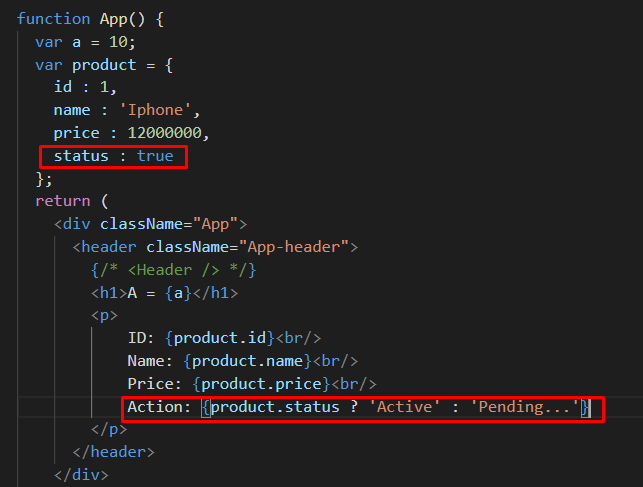
* Với biến là một project



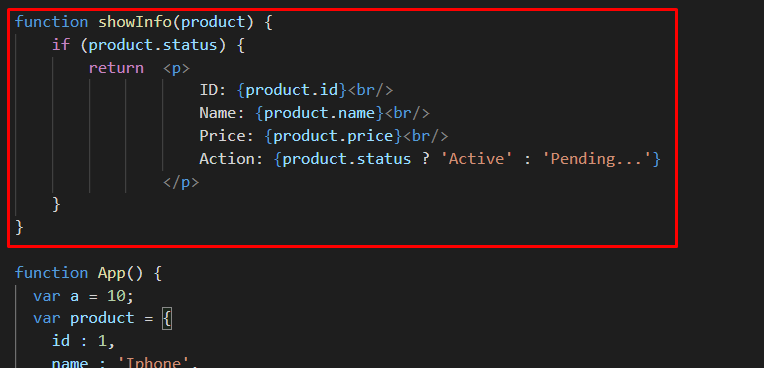
### Điều kiện

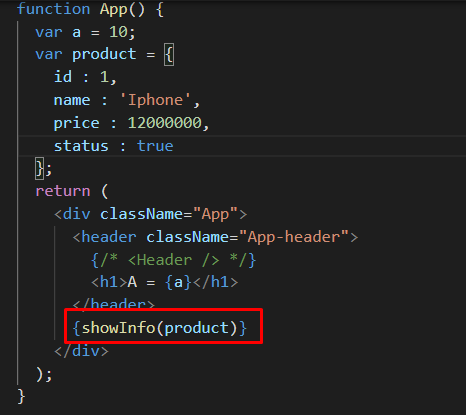
#### Toán tử 3 ngôi

* Cú pháp: {Cau\_lenh ? ’Dk\_dung’ : ‘Dk\_sai’}



#### Gọi thông qua thuộc tính



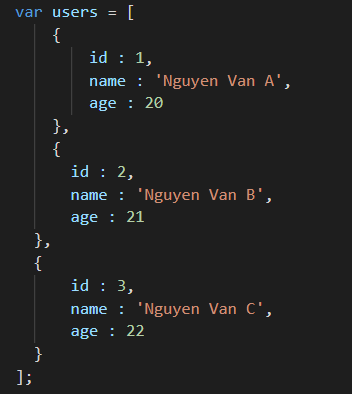


### Vòng lặp

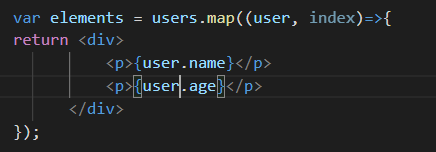
* Dùng hàm map() để sử dụng vòng lặp

VD:

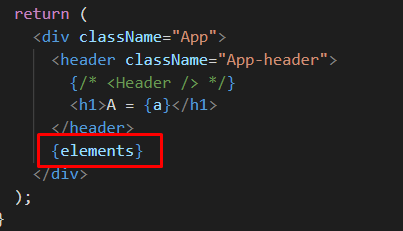
1. Ta có một mảng sau:



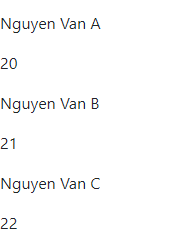
1. Khai báo vòng lặp với hàm map()



1. Gọi vòng lặp ra ngoài màn hình



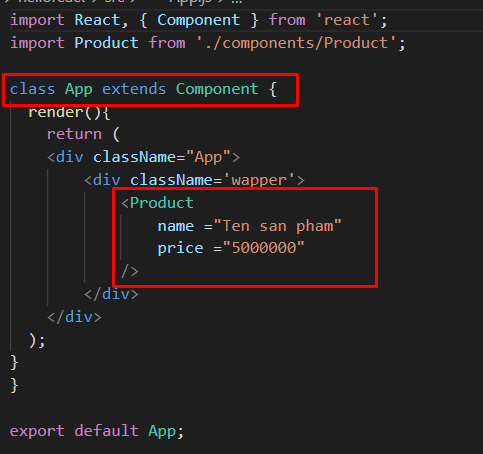
KQ:



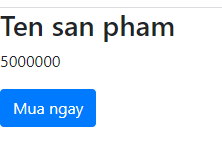
## Props

* Là thuộc tính của một components. Và props là cố định không thay đổi được giá trị của nó
* Truyền dữ liệu từ cha -> con theo dạng key=”value” (Không đặt key=”children”)
* Vlaue nhận vào có kiểu dữ liệu là kiểu chuỗi. Bỏ trong dấu { } để truyền đúng kiểu dữ liệu
* Nhận dữ liệu thông qua từ khóa: this.props.key
* Nhận nội dung bên trong thẻ: this.props.children

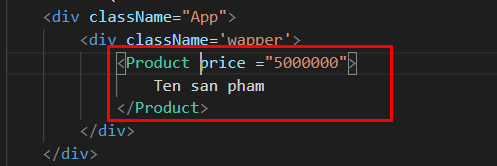
VD: Với React ở dạng ES6

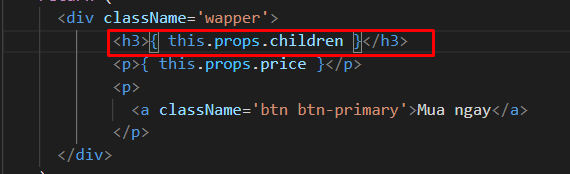


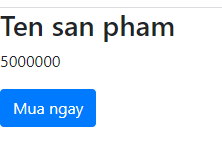




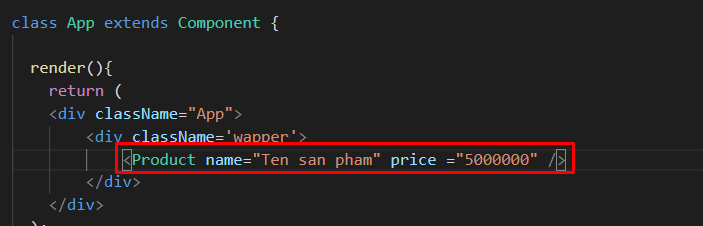
VD2:

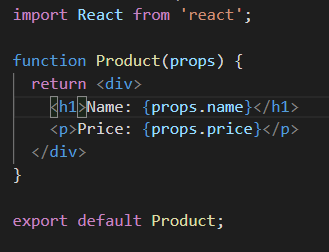






VD3: Với react ở dạng function

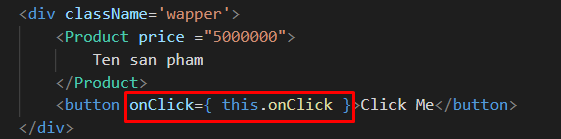


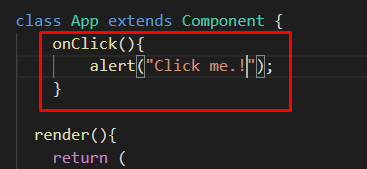


## Bắt sự kiện

* Sử dụng ReactJS được viết dưới dạng ES6

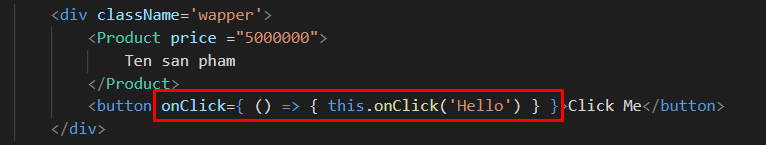
**Cách 1**: Bắt sự kiện không truyền tham số

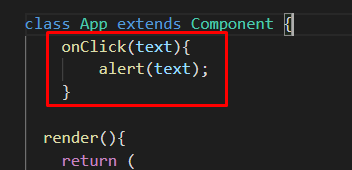




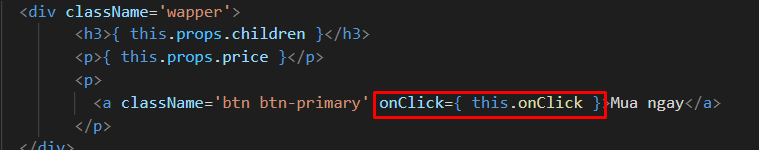
**Cách 2**: Sự kiện có tham số

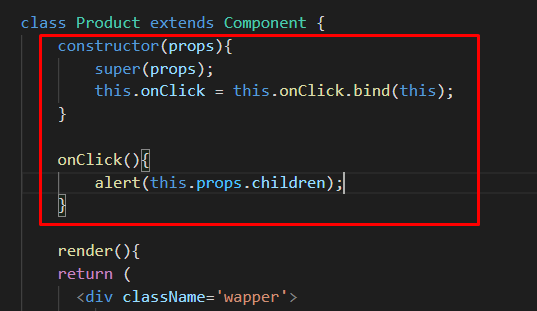
* Sử dụng () => {su\_Kien}



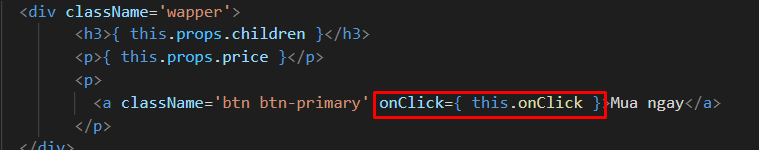


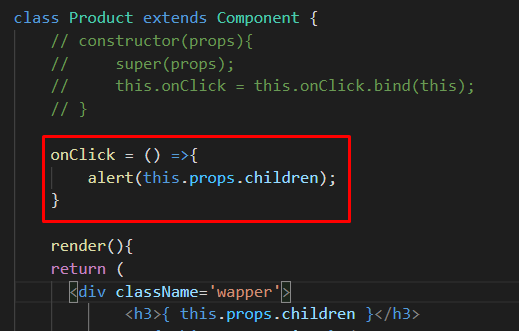
**Cách 3**: Sử dụng Props





**Cách 4**: Sử dụng props không khai báo contructor

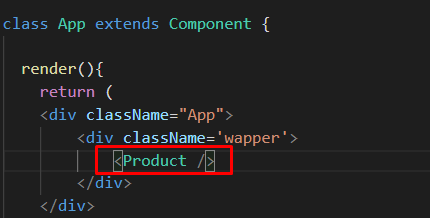




* Có thể tham khảo một số Events khác tại : <https://reactjs.org/docs/events.html#gatsby-focus-wrapper>

## State

* State là trạng thái của component. Khai báo những giá trị cần lưu trữ của riêng component đó. Và ta có thể thay đổi giá trị của state



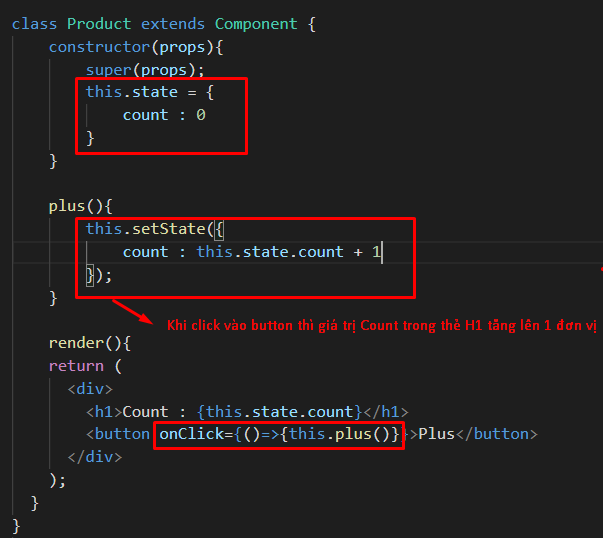


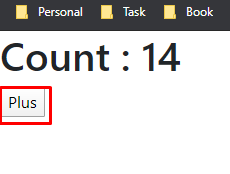
### SetState

* Dùng để thay đổi giá trị của state.

|  |  |
| --- | --- |
| Props | State |
| Nhận dữ liệu từ bên ngoài | Dữ liệu nội bộ |
| Không thể thay đổi giá trị | Có thể thay đổi giá trị |
|  | Phạm vi private trong component |

VD

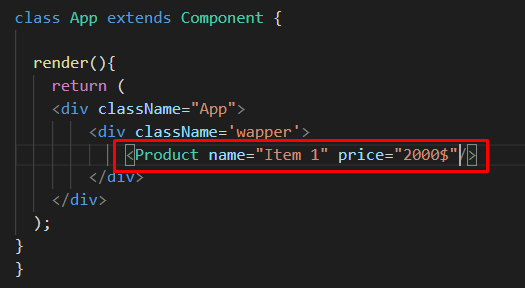




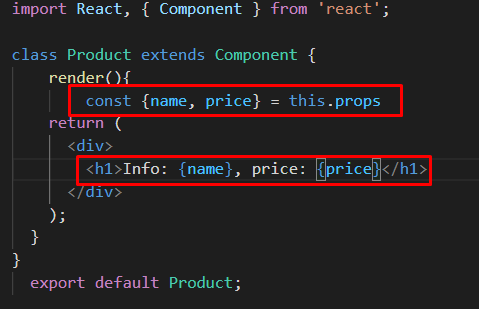
## Destructuring props and state

VD:

App.js



Product.js

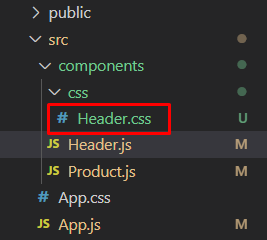




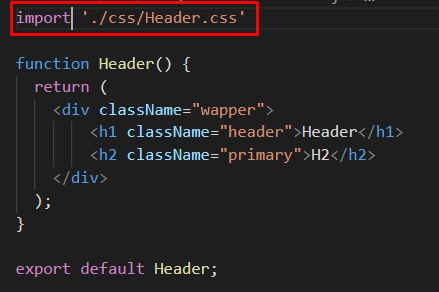
# CSS trong React

## Import css với function component

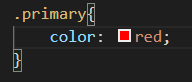
**1**: Cấu trúc thư mục của file css



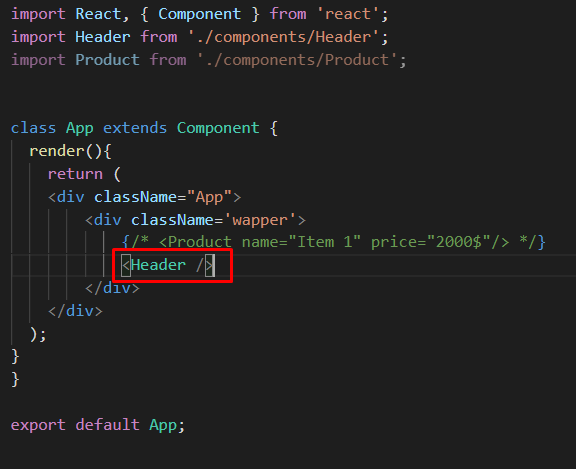
**2**: File Header.js



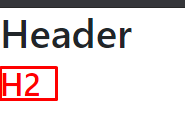
**3**: File Header.css



4: Flie App.js

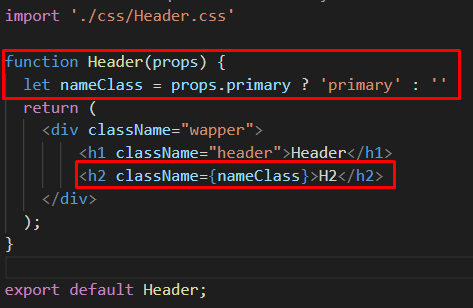


Kết quả:

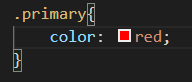


## Import css dưới dạng props

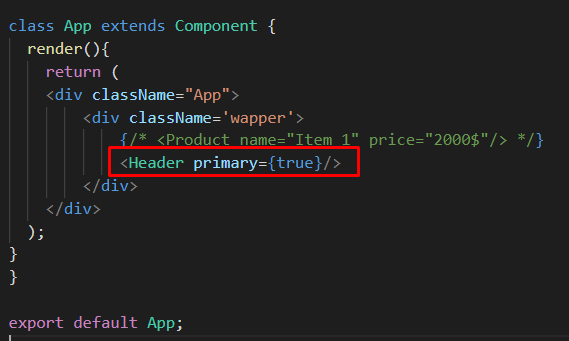
**1**: File Header.js



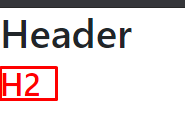
**2**: File Header.css



**3**: Flie App.js



Kết quả:



## Add nhiều class vào element trong ReactJS



Với: `` là phím dấu ~ bên cạnh phím số 1

${nameClass}: là tên class được viết dưới dạng props

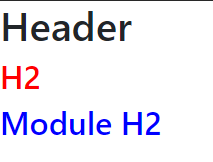
Size-xl: là tên của class khác

## Inline CSS



## Module CSS

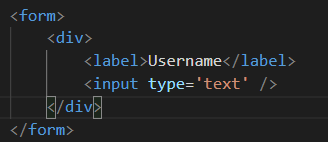




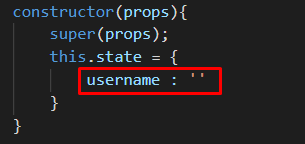
# Form Handling

## Với các text filed

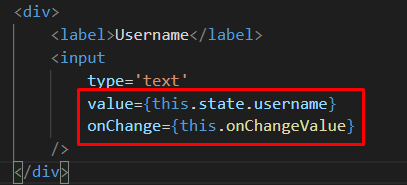
Bước 1:



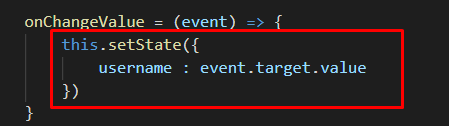
Bước 2:



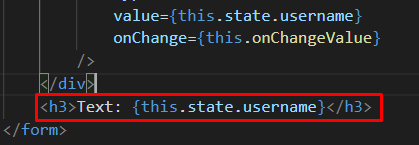
Bước 3:



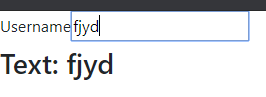
Bước 4:



Bước 5:

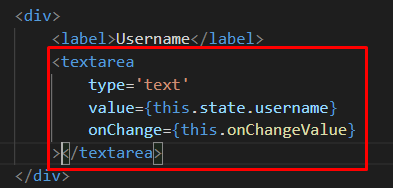


KQ:



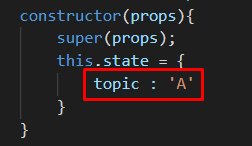
## Với textara

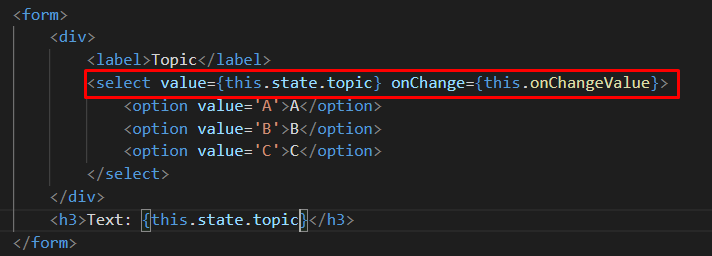
Tương tự như làm với text filed nhưng ở bước 3 thay thế:

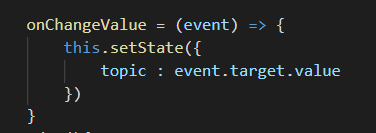


## Với dropdown

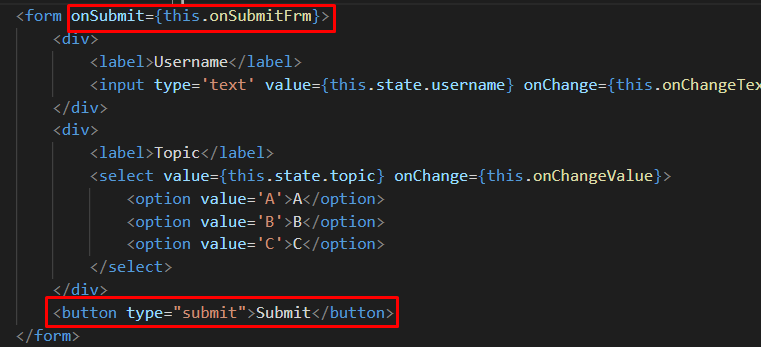
Cũng làm tương tự như text filed:

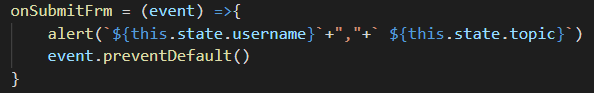






## Với submit





# Vòng đời Component

## Mounting

Một component thực hiện "mount" chỉ khi nó render trong lần đầu tiên. Khi mount nó sẽ thực hiện lần lượt các mounting lifecycle methods sau:

**contructor**: Là lifecycle methods được chạy đầu tiên trong tất cả các ngôn ngữ nó cũng được chạy đầu tiên vì đây là hàm khởi tạo.

**componentWillMount**: Là lifecycle methods chạy tiếp sau đó trước khi lifecycle methods render được chạy.

**render**: Là lifecycle methods chính của component.

**componentDidMount**: Luôn chạy sau khi các lifecycle methods đã chạy xong. ComponentDidMount là một nơi thích hợp để kết nối một ứng dụng React với một ứng dụng bên ngoài như một web API hay một javascript framework. componentDidMount cũng là nơi để các method set time như: setTimeout và setInterval.

VD:

class LifeA extends Component {

    constructor(props){

        super(props);

        this.state = {

            name:'Quang'

        }

        console.log('LifeA contructor');

    }

    static getDerivedStateFromProps(props, state){

        console.log('LifeA getDerivedStateFromProps');

        return null;

    }

    componentDidMount(){

        console.log('LifeA DidMount')

    }

  render(){

        console.log('LifeA render')

        return (

            <div>

                <div>Life A</div>,

                <LifeB/>

            </div>

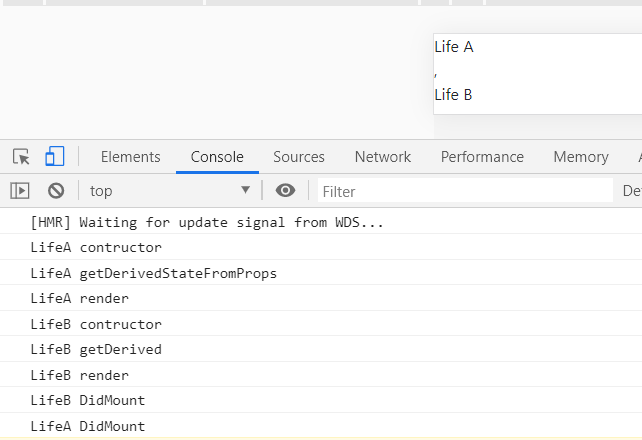
        )

    }

}

export default LifeA;

* Làm tương tự với LifeB



## Updating

Sau khi component render lần đầu tiên, các lifecycle methods của Updating sẽ được gọi bắt đầu với lần render thứ hai. Với cơ chế automatic binding của mình thì chắc chắn các component sẽ được render nhiều lần trong ứng dụng của bạn. Có các method trong lifecycle updating:

**componentWillReceiveProps**: Mỗi khi một instance của component được update, nó sẽ tự động gọi lần lượt 5 methods trên. Updating lifecycle methods đầu tiên là componentWillReceiveProps, khi một instance của component được update, componentWillReceiveProps sẽ được gọi trước khi render. Có một chú ý ở đây là: componentWillReceiveProps chỉ được gọi nếu component được nhận một prop.

**shouldComponentUpdate**: Khi một component update, shouldComponentUpdate sẽ được gọi sau componentWillReceiveProps nhưng vẫn trước render. shouldComponentUpdate sẽ trả về true hoặc false. Nếu trả về true, việc update sẽ diễn ra bình thường. Nhưng nếu trả về false, tất cả method còn lại của updating cycle method sẽ không được gọi nữa, kể cả render và component sẽ không được update.

**componentWillUpdate**: Mục tiêu chính của componentWillUpdate là tương tác những thứ bên ngoài kiến trúc React. Nếu bạn cần set up một cái gì đó ngoài React, như check window size hay tương tác với một API thì componentWillUpdate là nơi thích hợp để thực hiện điều đó.

**render**

**componentDidUpdate**: componentDidUpdate thường được sử dụng để tương tác với một số thứ bên ngoài vào môi trường React như là browser hay APIs. Nó tương tự như componentWillUpdate ngoại trừ việc nó được gọi sau method render

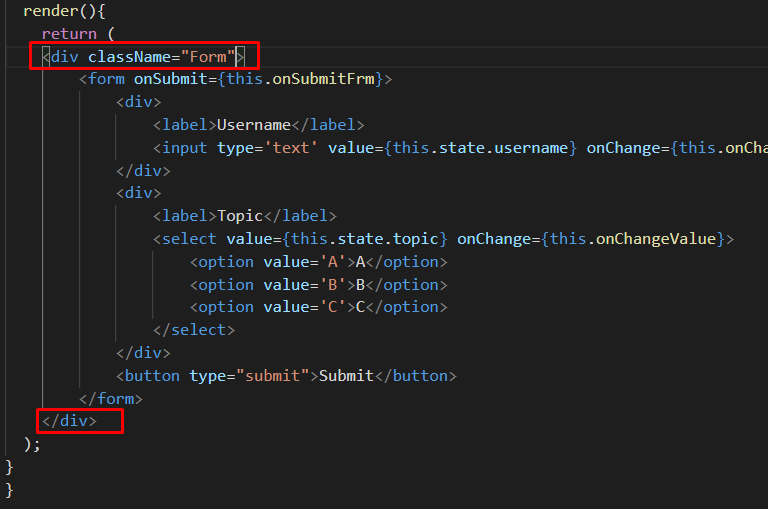
## Unmounting

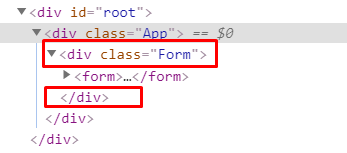
Quá trình unmounting của component xảy ra khi component bị removed từ một DOM. Việc này có thể xảy ra khi một DOM được render mà không có component hoặc nếu user chuyển hướng đến một trang web khác hoặc khi trình duyệt được đóng. Chỉ có duy nhất một method trong quá trình này là: componentWillUnmount, componentWillUnmount sẽ được gọi trước khi một component bị remove khỏi một DOM. Nếu một component khởi tạo bất kì một method nào mà method đó yêu cầu phải clean up thì componentWillUnmount sẽ là nơi bạn nên đặt clean up.

# Fragments

Nó là 1 parttern chung trong React dùng cho component để trả về nhiều elements mà không cần tạo thêm các nodes vào DOM.

Thay vì dùng:





Thì ta nên dùng Fragment để thay thế và loại bỏ element Form:



