Báo cáo Seminar

Môn: Công nghệ Web và ứng dụng

Giảng viên: Trần Anh Dũng

Họ tên sv: Nguyễn Phi Ân

MSSV: 17520217

Đề tài: Tìm hiểu về ReactJS

1. **Lịch sử hình thành và phát triển:**

* Được phát triển bởi Jordan Wake, một kỹ sư phần mềm tại Facebook
* Được triển khai cho ứng dụng Newsfeed của Facebook năm 2011, và sau đó là Instagram.com năm 2012
* Trở thành framework mã nguồn mở vào 29/5/2013 với giấy phép Apache 2.0
* Tháng 10 năm 2014 với phiên bản 0.12.0 được thay bằng giấy phép 3-clause BSD và thêm một tệp văn bản PATENTS riêng biệt cho phép sử dụng bất kỳ bằng sáng chế nào của Facebook liên quan đến phần mềm
* Dựa trên phản hồi của cộng đồng, Facebook đã cập nhật việc cấp bằng sáng chế vào tháng 4 năm 2015 để bớt mơ hồ và dễ dãi hơn
* Sau 1 số vấn đề về giấy phép thì ngày 23/9/2017 Facebook thông báo React sẽ được cấp lại giấy phép bởi MIT. 3 ngày sau đó thì React 16.0.0 đã ra mắt với giấy phép mới

1. **Giới thiệu tổng quát:**
2. Giới thiệu:

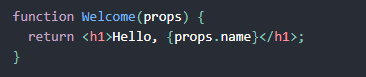
* React (còn được gọi là React.js hay ReactJS) là thư viên mã nguồn mở, là một framework trong Javascript dùng cho front end
* Dùng để xây dựng giao diện người dùng (user interfaces) hoặc các UI components
* Được duy trì và phát triển bởi Facebook và đóng góp từ cộng đồng developers trên thế giới
* Có thể sử dụng làm cơ sở để xây dụng single-page application hoặc mobile applications
* Các phần có liên quan với ReactJS: Virtual DOM, JSX, Components, Props và states

1. Vấn đề mà ReactJs giải quyết:

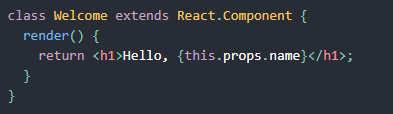
* Giải quyết được vấn đề của tầng View trong MVC (Model-View-Controller)
* Giúp viết mã Javascript dễ dàng hơn với JSX
* “Thành phần hoá” giao diện bằng cách tạo ra các component tương ứng cho các thành phần của giao diện (Một giao diện được ghép từ các giao diện nhỏ hơn)
* Component là tương lai của lập trình Web và là tính năng quan trọng nhất mà React mang lại

1. **Giới thiệu các thành phần trong ReactJS:**
2. **Components và Props:**

* Components giúp ta tách các thành phần trong UI thành các phần nhỏ không phụ thuộc, có thể tái sử dụng
* Về mặt khái niệm thì components giống như functions trong Javascript, chúng nhận các đầu vào tuỳ ý (gọi là “props”) và trả về React elements để mô tả giao diện hiển thị trên screen. VD:
* **Function và Class Components:**
* Cách đơn giản nhất để tạo một component là viết một Javascript function. VD:



* Ta có thể gọi các components là các “function components”
* Ngoài ra ta có thể dùng ES6 class để định nghĩa component. VD:



* Function components và Class component là tương đương trong góc nhìn của ReactJS (Một vài điểm khác nhau như cú pháp, state (function là stateless, class là stateful), lifecycle, 1 bên ít code và dễ test hơn,…)
* **Rendering a component:**
* Các React elements không chỉ đại diện cho DOM tags mà còn có thể là các user-defined components:

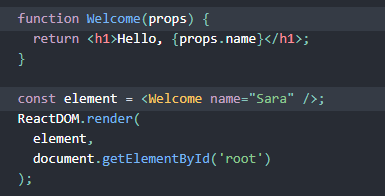
+ DOM tags:

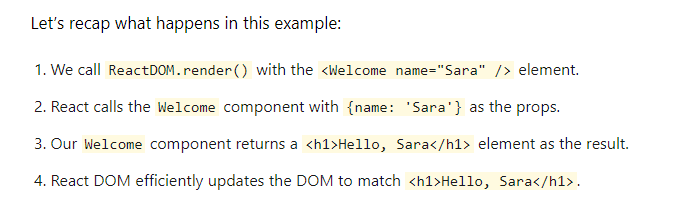


+ User-defined components:

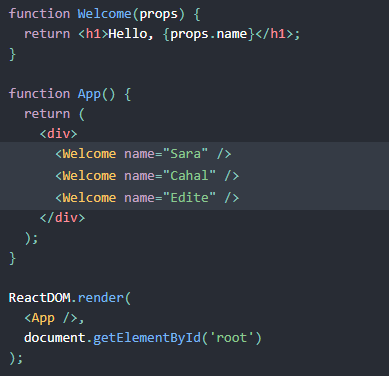


* Khi một element là user-defined component thì React sẽ chèn vào đó các thuộc tính của JSX và con cho component này như một object duy nhất. Ta gọi đây là “props”. VD:





* **Composing components:**
* Các components có thể tham chiếu đến các component khác
* Các thành phần trong React như button, dialog, screen,… đều được coi là các components
* VD cho việc component có thể tham chiếu đến các component khác:

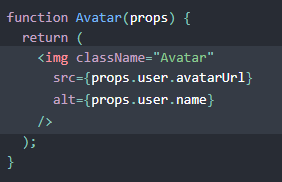


* **Extracting component:**
* Ta có thể tách một component lớn thành các component nhỏ hơn. VD:



+ Component comment nhận một author (1 object), text (1 string) và ngày là các props và mô tả cho comment trên mạng xh. Tuy nhiên như vậy thì các component bên trong không thể tái sử dụng

+ Các component sau khi tách:

\_ Avatar

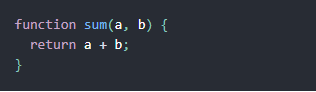
\_ User info



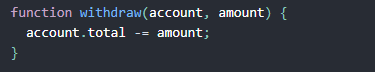
\_ Comment component sau khi tách:



* Đây là cách tốt để giúp tối giản code và tận dụng lại các components được sử dụng nhiều lần
* **Props are Read-only:**
* Cho dù ta có khai báo 1 component là 1 function hay 1 class thì nó không bao giờ được thay đổi các props của nó. VD:



* Các function như trên được gọi là “thuần tuý” vì chúng không thay đổi các input của chúng và luôn trả về cùng 1 kết quả cho các input giống nhau (Các input có giá trị không đổi sau khi function được thực thi)
* Ngược lại một function được gọi là “không thuần tuý” khi nó thay đổi input của nó (Giá trị của input sẽ bị thay đổi sau khi function được thực thi). VD:



* React khá linh hoạt nhưng nó có 1 quy tắc nghiệm ngặt là:

+ Tất cả các React components phải hoạt động như dạng thuần tuý với các props của nó

* Do đó trong React có state để giúp các component phản hồi các actions từ người dùng và thay đổi các output theo thời gian mà không xâm phạm quy ước trên

1. **State and Lifecycle:**

* Phần trước khi muốn render up date của component ra màn hình thì ta dùng ReactDOM.render()
* Phần này ta sẽ tìm hiểu cách để biến một component có thể thực sự tái sử dụng và đóng gói
* Bài toán VD về 1 cái Clock, Clock vẫn update qua ReactDOM.render() và ta cần ad state vào để nó có thể tự update:



* State cũng tương tự như props nhưng nó private và có thể control hoàn toàn bởi component
* Converting a Function to a Class:
* Ta có thể convert 1 function thành 1 class qua 5 bước:

+ Bước 1: Tạo 1 ES6 Class với cùng tên, và class này extends React.component

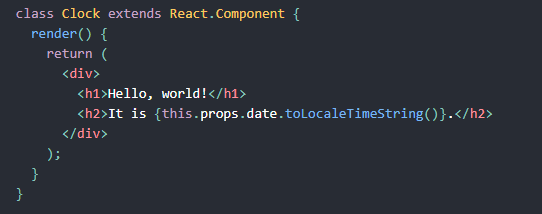
+ Bước 2: Add cho class đó 1 hàm rỗng gọi là render()

+ Bước 3: Chuyển phần thân của function vào trong hàm render() vừa tạo

+ Bước 4: Thay props thành this.props trong phần thân của hàm render()

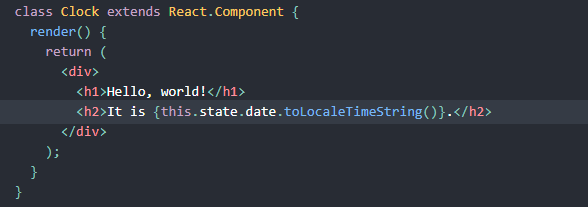
+ Bước 5: Xoá bỏ phần khai báo còn lại của hàm rỗng

+ VD về kết quả sau khi thực thi xong

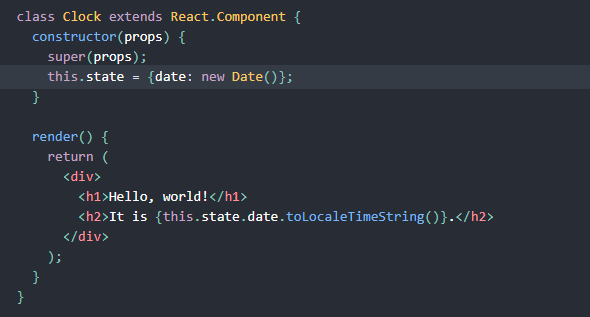


* Hàm render() sẽ được gọi khi có update xảy ra, miễn là ta render() <Clock /> vào cùng 1 DOM node, chỉ một instance suy nhất của Clock có thể được sử dụng. Nó giúp ta có thể sử dụng thêm các tính năng như local state và lifecycle method
* Adding Local State to a Class:
* Ta cần 3 bước để chuyển props thành state:

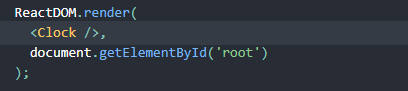
+ Bước 1: Thay this.props.<Tên thuộc tính cần gọi> với this.state.<Tên thuộc tính cần gọi> trong render method:



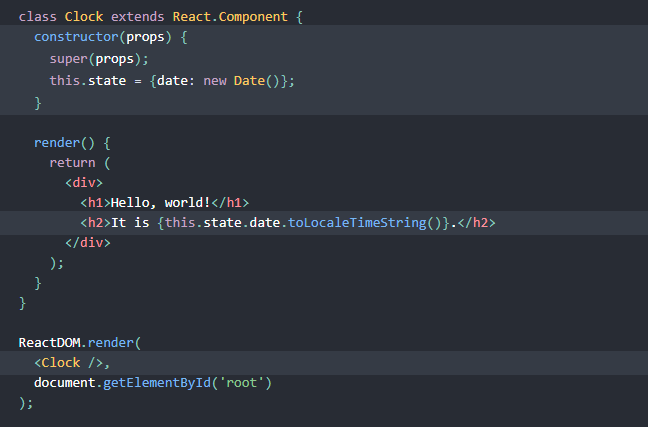
+ Bước 2: Tạo 1 class constructor để khởi tạo cho this.state (Class component phải luôn gọi super với props):



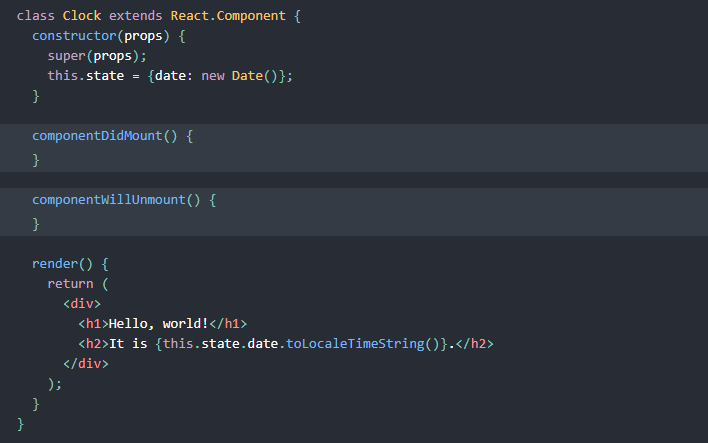
+ Bước 3: Xoá thuộc tính đã khởi tạo cho state khỏi thẻ của class component. VD:



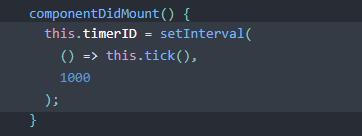
* Kết quả sau khi hoàn thành 3 bước:



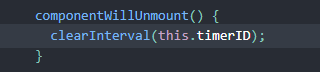
* Adding Lifecycle Methods to a Class:
* Thông qua các lifecycle methods thì ta có thể giải phóng các tài nguyên đã cấp phát cho các components khi các components không còn dùng đến nữa (destroy)
* Khi một component được render lần đầu tiên lên DOM thì đây là gọi là “mouting” trong
* Khi DOM của component bị component remove thì đây gọi là “unmouting” trong React
* Ta có thể định nghĩa các hàm để mount và unmount trong class của component
* VD:



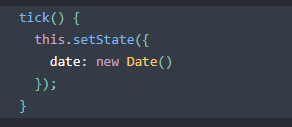
* Hàm componentDidMount sẽ run khi component output đã được render trong DOM. VD:



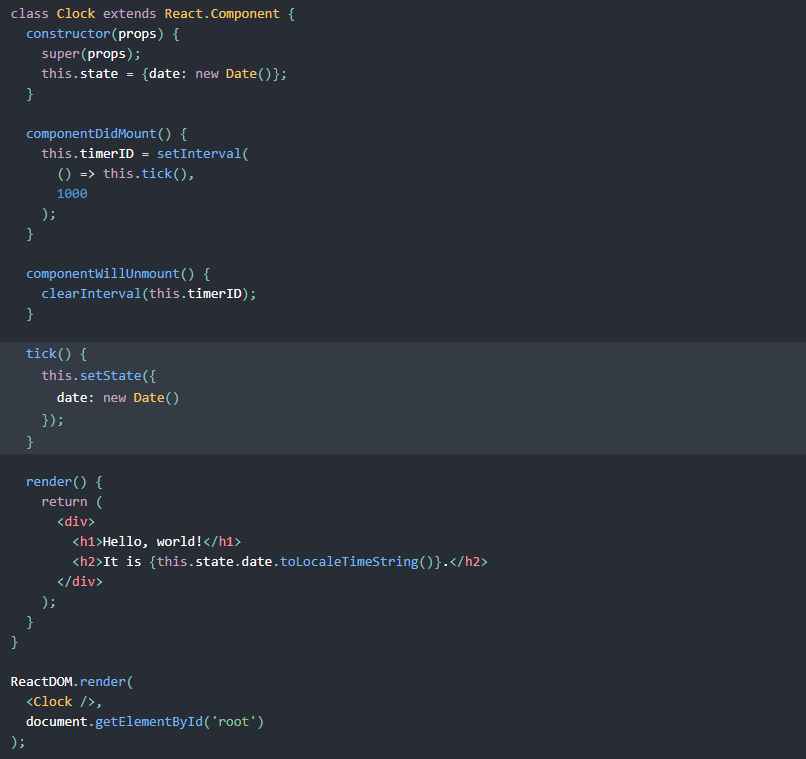
* Mặc dù this.props được thiết lập bởi React và this.state có ý nghĩa đặc biệt, bạn có thể tự do thêm các trường bổ sung vào lớp theo cách thủ công nếu bạn cần lưu trữ thứ gì đó không tham gia vào luồng dữ liệu (chẳng hạn như timer ID)
* Tiếp theo là hàm để destroy component resource khi không dùng nữa đó là hàm componentWillUnmount()



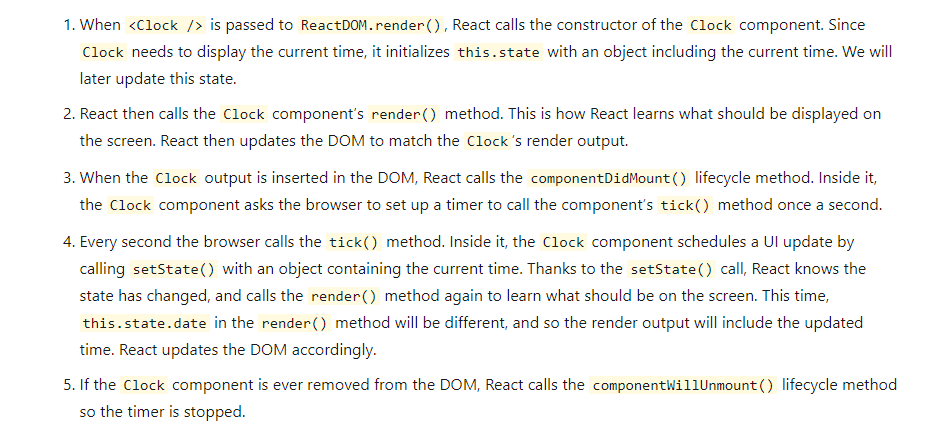
* Hàm tick() để update local state cho component thông qua việc gọi hàm setState() mỗi khi thời gian thay đổi



* Class Clock hoàn chỉnh sau khi tạo các hàm



* Luồng code chạy khi thực thi



* Using state correct:
* Có 3 điều cần lưu ý khi sử dụng setState()

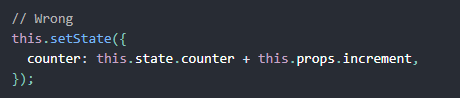
+ Không được sửa đổi state trực tiếp



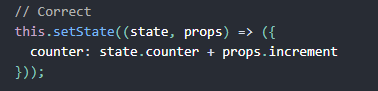
* Nếu dùng sai thì component sẽ không re-render

+ State update phải được xử lý bất đồng bộ

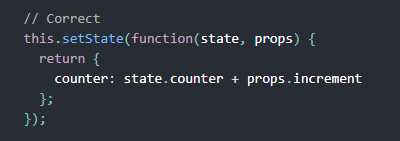
* Vì React sẽ gom nhiều lệnh gọi setState() để thực thi trong 1 update để tăng hiệu suất
* Vì this.props và this.state được update bất đồng bộ nên ta không thể dựa vào giá trị của chúng để tính toán cho state tiếp theo. VD:



* Thay vào đó ta có thể truyền vào đó 1 hàm thay vì 1 object. Function này sẽ nhận state trước đó làm tham số 1 và props là tham số 2 (props là khi các update được hoàn thành (ứng dụng)). VD:

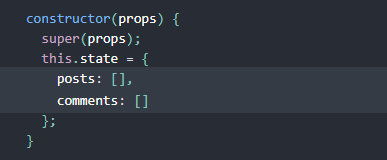


* Ta có thể lược bỏ dấu mũi tên vì hàm đó có thể truyền vào 1 hàm bình thường

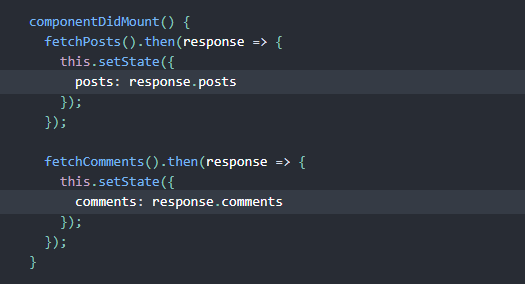


+ Các state updates phải được đồng nhất:

* Khi ta gọi hàm setState() thì React sẽ hợp nhất object ta cung cấp vào state hiện tại. VD state có thể bao gồm các biến độc lập

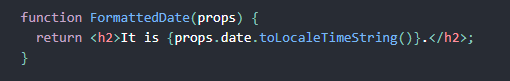


* Tuy nhiên ta có thể update riêng biệt cho từng biến trong state:



* Việc hợp nhất là khá nông nên this.setState({comments}) sẽ giữ nguyên this.state.posts nhưng sẽ thay đổi hoàn toàn this.state.comments
* The data flows down:
* Các component cha và con đều không thể biết rằng liệu 1 component nhất định là stateful hay stateless, và chúng không cần biết là liệu nó được định nghĩa là 1 class hay function. Đó là lý do vì sao state thường được gọi là local hay encapsulated vì nó không thể access từ các component khác mà chỉ có thể từ các component sở hữu và thiết lập nó
* 1 component có thể chọn để truyền state của nó xuống như 1 props cho các child components của nó





* FormattedDate component có thể nhận date trong props của nó và không cần biết là nó nhận từ state của Clock hay từ Clock’s props hoặc được gõ từ state
* Đây thường gọi là “top-down” hoặc “undirectional” (1 chiều) data flow
* Mỗi state thường được sở hữu bởi 1 vài component cụ thể, và mỗi data hoặc UI được tạo ra từ các state đó có thể ảnh hưởng đến các components “below” chúng trong cây
* Ta có thể tưởng tượng component tree như một waterfall về props, mỗi component’ state như là một source trong dòng nước tham gia vào một diểm tuỳ ý trên cây nhưng theo hướng đi xuống
* Mỗi component là độc lập. VD:



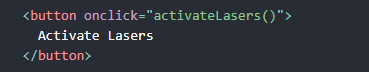
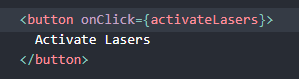
* App sẽ render 3 clock, mỗi clock sẽ có timer của riêng nó và update riêng biệt
* Trong React app, 1 component là stateful hoặc stateless được coi là 1 chi tiết triển khai của component có thể thay đổi theo thời gian. Ta có thể dùng các stateless components trong các stateful components và ngược lại

1. **Handling event:**

* Xử lý event trong React elements khá giống với giải quyết event trên DOM elements. Tuy nhiên có 1 số cú pháp khác biệt giữa 2 bên:

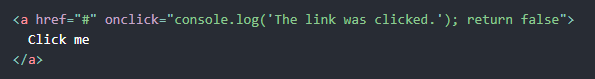
+ React events được đặt tên theo quy ước camelCase thay vì lowercase

+ Với JSX ta truyền 1 function như 1 event handler thay vì 1 string. VD trong HTML và React:

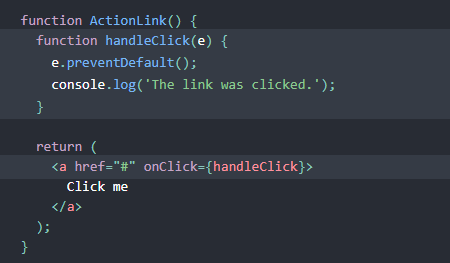


* Một sự khác nhau nữa là ra không thể return false trong React để ngăn thao tác mặc định (default behavior), trong HTML thì ta có thể làm như vậy. Thay vào đó ta phải gọi hàm preventDefault để ngăn thao tác mặc định trong React. VD trong HTML và React (ngăn không cho mở link trong tab mới):

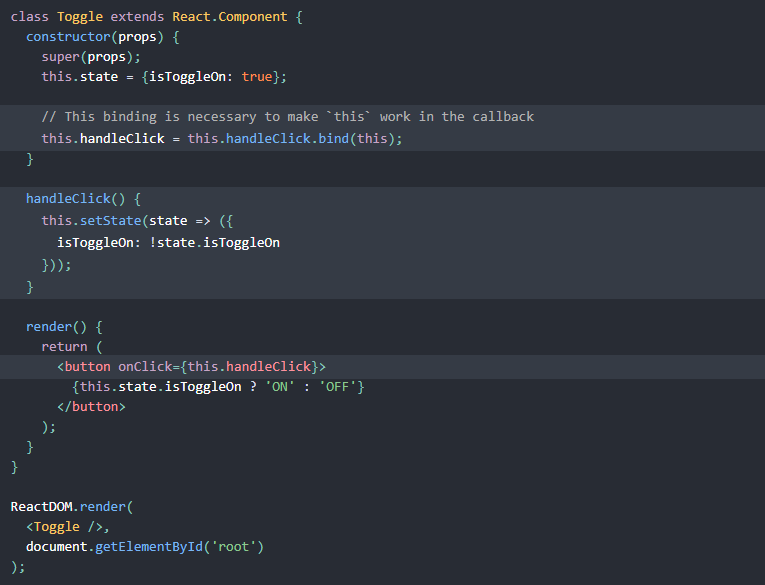
+ HTML:



+ React:

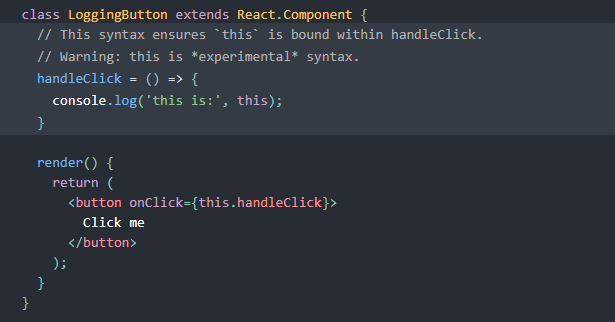


* e là một event tổng hợp. React định nghĩa các event tổng hợp này thông qua đặc tả của W3C nên ta có thể sử dụng thoải mái và không lo việc tương thích trình duyệt
* Các event trong React không hoạt động như các native event
* Khi sử dụng React thì ta không cần phải gọi addEventListener để add listeners cho 1 DOM element sau khi nó được tạo, thay vào đó ta có thể cung cấp 1 listener khi element được render xong lần đầu
* Khi ta định nghĩa 1 component sử dụng ES6 class, 1 pattern phổ biến cho 1 event handler là 1 hàm trên class. VD về 1 nút toggle chuyển giữa ON và OFF

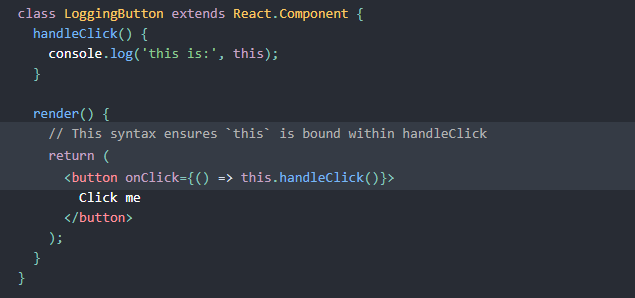


* Ta phải cận thận với ngữ nghĩa của this trong JSX callbacks. Trong Javascript, các phương thức của lớp mặc định không bị ràng buộc. Nên nếu ta không dùng this.handleClick và truyền vào onClick thì this sẽ bị “undefined” khi được gọi
* Đây không phải là thao tác của riêng React mà nó là quy ước về hoạt động của hàm trong Javascript. Cụ thể hơn nếu ta tham chiếu đến một hàm không có () phía sau nó, VD onClick={this.handleClick}, thì ta phải ràng buộc phương thức đó
* Nếu ta không muốn gọi bind thì ta có thể sử dụng 2 cách khác

+ Cách 1: Sử dụng trực tiếp class field để ràng buộc với callback. VD:



+ Cách 2: Ta có thể sử dụng toán tử mũi tên. VD:



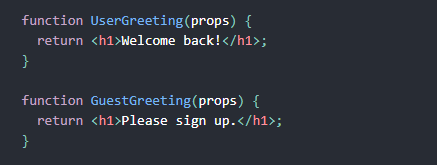
* Tuy nhiên vấn đề với các cú pháp trên là các callback sẽ được tạo mới mỗi khi LogginButton được render. Với hầu hết các trường hợp thì điều này là không sao. Nhưng nếu callback này được truyền vào như 1 tham số cho các components thấp hơn, thì các components này sẽ phải làm thêm việc re-render. Do đó ta nên sử dụng bind trong React component constructor hoặc sử dụng class field để hạn chế việc ảnh hưởng hiệu năng
* Passing Arguments to Event Handler:
* Trong một vòng lặp thì ta có thể cần truyền thêm tham số cho 1 event handler. VD nếu ta muốn xoá id nếu là row ID:



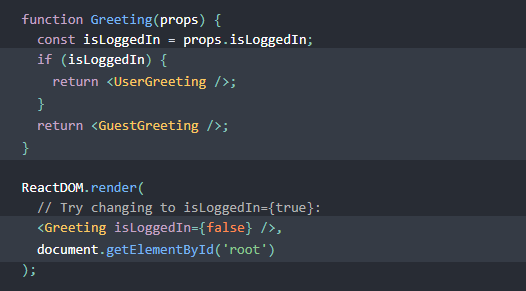
* Đây là 2 cách có thể sử dụng với cú pháp hàm mũi tên (arrow function) và Function.prototype.bind
* Trong 2 trường hợp thì tham số e đại diện cho React event sẽ được truyền vào tham số thứ 2 phía sau ID. Với hàm mũi tên thì ta phải truyền nó một cách rõ ràng nhưng với bind thì bất kì tham số nào khác sẽ được tự động chuyến về phía trước

1. **Conditional Rendering:**

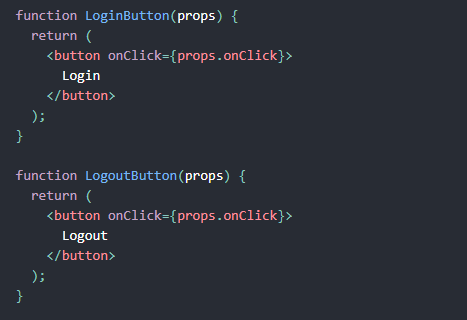
* Trong React, bạn có thể tạo các component riêng biệt để đóng gói behavior mà bạn cần. Sau đó, bạn chỉ có thể render một số trong số chúng, tùy thuộc vào state của ứng dụng
* Conditional rendering trong Reac hoạt động tương tự như conditions trong JavaScript
* Ta có thể dùng if hoặc toán tử điều kiện của JavaScript để tạo các elements đại diện cho state hiện tại, và React sẽ update UI tương ứng với chúng
* VD ta có 2 components:



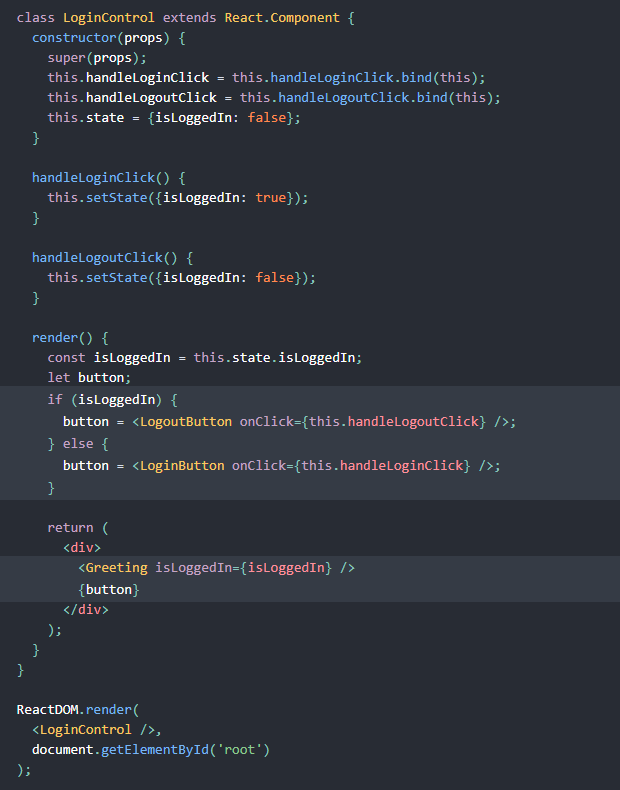
* 2 components này đại diện cho phần hiển thị khi người dùng log in hoặc chưa log in (xét dựa vào biến isLoggedIn)



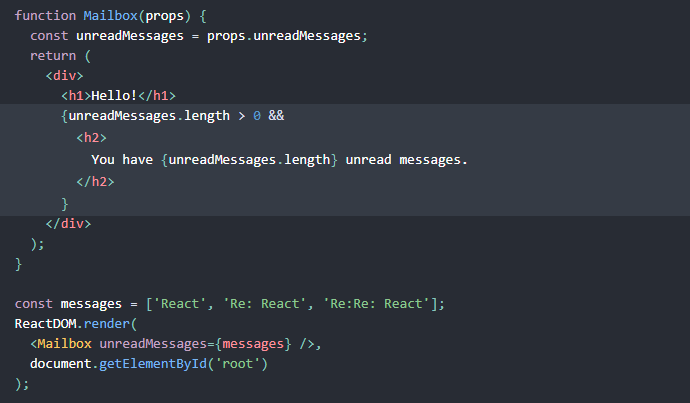
* Element Variables:
* Ta có thể dùng các biến để chứa các elements. Điều này có thể giúp bạn hiển thị có điều kiện một phần của component trong khi phần còn lại của output không thay đổi.
* VD ta có 2 component cho nút log in và log out:



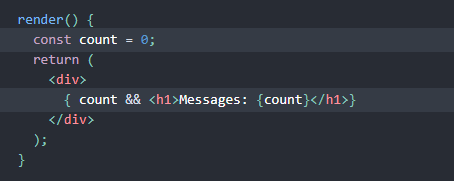
* Sau đó ta tạo ra 1 stateful component gọi là LoginControl. Nó sẽ vẽ nút Login, Logout ở trên dựa vào trạng thái hiện tại, nó cũng sẽ render Greeting như vd trước. Code:



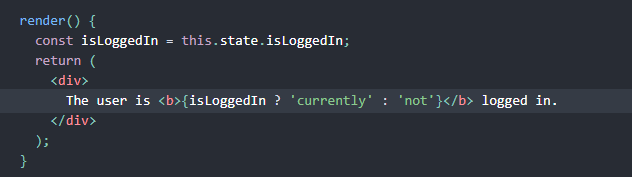
* Ta có thể rút ngắn các điều kiện xét thông qua một số cú pháp trong JSX như: inline if với toán tử logic &&, inline if else với toán tử điều kiện
* Inline If with Logical && Operator:
* Ta có thể nhúng các biểu thức trong JSX bằng cách gói chúng trong cặp ngoặc nhọn. Trong đó bao gồm toán tử && của JavaScript. Nó có thể tiện lợi cho điều kiện có chứa 1 element. VD:



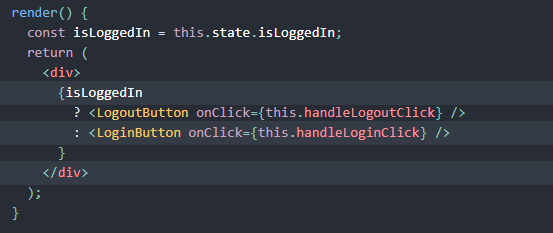
* Điều kiện này là hợp lệ trong JavaScript vì “true && biểu thức” thì sẽ xét theo biểu thức, còn “false && biểu thức” thì sẽ là false
* Nếu điều kiện là true thì element phía sau && sẽ được hiển thị ra output, còn ngược lại thì React sẽ skip nó
* Lưu ý: Nếu ta return 1 biểu thức sai thì element sau && sẽ bị skip nhưng nó sẽ trả về biếu thức sai. VD sau sẽ trả về <div>0</div>



* Inline If-Else with Conditional Operator:
* Đây là phương thức thứ 2 để render các element trong biểu thức điều kiện với việc xét điều kiện trong 1 dòng. Cú pháp: <biếu thức điều kiện> ? true : false
* VD dùng cho một biểu nhỏ (xét 1 string):

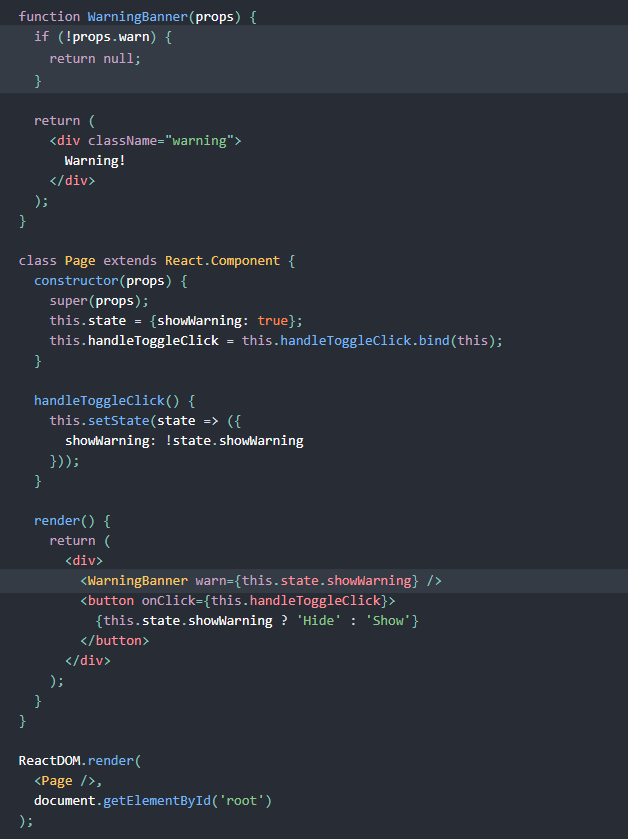


* Nó có thể dùng cho một biểu thức lớn hơn:



* Khi điều kiện phức tạp thì ta nên trích xuất component (tách các phần trong component lớn thành các component con nhỏ hơn)
* Preventing Component from Rendering:
* Đây là phần mà ta thường ít dùng đến, ta sẽ cho các component ẩn bản thân của nó mặc dù nó được render từ component khác. Để làm điều này thì ta có thể return null thay vì render output của nó
* VD ta có 1 banner hiển thị cảnh báo, banner này xuất hiện dựa vào giá trị của 1 tham số là warn. Nếu prop này có giá trị false thì banner sẽ không render:

1. Lists ad Keys:



* Khi ta trả về null từ hàm render của component thì nó không ảnh hưởng đến việc kích hoạt các llifecycle method của component. VD như hàm componentDidUpdate vẫn sẽ được gọi