**MENU**

[**1.** **SPA và MPA** 3](#_Toc139055780)

[**1.1** **Giới thiệu** 3](#_Toc139055781)

[**1.2** **So sánh** 3](#_Toc139055782)

[**1.3** **Lựa chọn SPA hay MPA** 4](#_Toc139055783)

[**2.** **Document.creatElement() (Tạo Element trong thẻ <script>)** 5](#_Toc139055784)

[**2.1** **Document.appendChild() (Thêm element con vào element cha)** 6](#_Toc139055785)

[**3.** **Thêm thư viện ReactJS vào Website** 7](#_Toc139055786)

[**3.1** **Cách thư viện được lưu trữ và phân phối** 7](#_Toc139055787)

[**3.2** **Thêm thư viện ReactJS** 10](#_Toc139055788)

[**3.3** **React official docs** 11](#_Toc139055789)

[**4.** **React.creatElement() (Tạo Element bằng thư viện ReactJS)** 11](#_Toc139055790)

[**4.1** **So sánh giữa document.creatElement() và React.creatElement()** 11](#_Toc139055791)

[**4.1.1** **Thêm Element vào website** 11](#_Toc139055792)

[**4.1.2** **Thêm Element lồng Element** 13](#_Toc139055793)

[**5.** **React-DOM (kết nối thư viện ReactJS và DOM tree)** 14](#_Toc139055794)

[**5.1** **ReactDOM.render()** 15](#_Toc139055795)

[**6.** **ReactJS version 18** 16](#_Toc139055796)

[**6.1** **ReactDOM.render()** 17](#_Toc139055797)

[**7.** **JSX (Javascript XML)** 18](#_Toc139055798)

[**7.1** **JSX không phải là HTML** 19](#_Toc139055799)

[**+Có thể gán JSX vào biến** 19](#_Toc139055800)

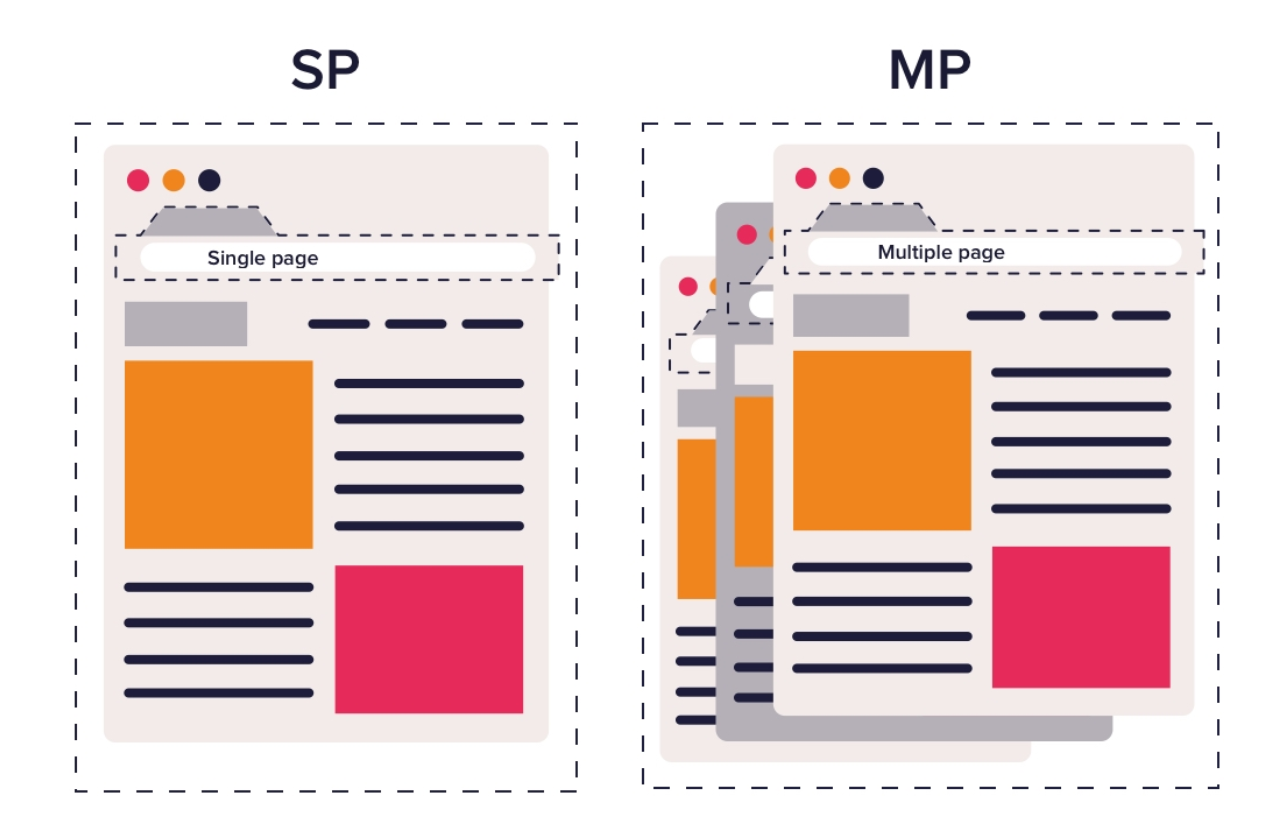
[**+Có thể sen lẫn code Javascript trong JSX** 19](#_Toc139055801)

[**7.2** **Babel (Thư viện trung gian giúp JXS làm việc được với React-DOM)** 20](#_Toc139055802)

[**7.2.1** **Cách cài thư viện Bable** 20](#_Toc139055803)

1. **SPA và MPA**

**-**Là 2 kiến trúc phát triển web hiện nay



* 1. **Giới thiệu**

**SPA**

-Single page application được cho là cách tiếp cận hiện đại hơn

-Không yêu cầu tải lại trang trong quá trình sử dụng

-Khi chúng ta sử dụng ReactJS thì có nghĩa là chúng ta đang sử dụng kiến trúc **SPA**

**MPA**

-Multi page application là một các tiếp cận cổ điển (vẫn phổ biến đến nay)

-Tải lại trang trong quá trình sử dụng (click vào đường link, chuyển trang, …)

* 1. **So sánh**

**+Tốc độ**

-**SPA** nhanh hơn khi sử dụng. Phần lớn tài nguyên được tải trong lần đầu. Trang chỉ tải thêm dữ liệu mới khi cần

-**MPA** chậm hơn khi sử dụng. Luôn tải lại toàn bộ trang khi truy cập và chuyển hướng

**+Bóc tách Code**

-**SPA** có phần Front-end riêng biệt

-**MPA** Front-end và Back-end phụ thuộc nhau nhiều hơn, được đặt trong cùng 1 dự án

**+SEO**

**-**SEO (Search Engine Optimization) là quá trình tối ưu hóa công cụ tìm kiếm, nhằm cải thiện vị trí và hiển thị của trang web trong kết quả tìm kiếm tự nhiên của các công cụ tìm kiếm như Google, Bing, Yahoo, và nhiều hơn nữa. Mục tiêu của SEO là tăng lượng lưu lượng truy cập tự nhiên (organic traffic) đến trang web, đồng thời cải thiện khả năng hiển thị và sắp xếp của trang web trong kết quả tìm kiếm.

Quá trình SEO bao gồm việc tối ưu hóa các yếu tố nội dung và cấu trúc trang web, tạo liên kết (link building), cải thiện trải nghiệm người dùng, nghiên cứu từ khóa, và theo dõi và đánh giá hiệu quả của các chiến lược tối ưu hóa. SEO đòi hỏi sự hiểu biết về cách hoạt động của các công cụ tìm kiếm, quy tắc và tiêu chuẩn của ngành, và việc áp dụng các kỹ thuật phù hợp để tăng tính thân thiện với công cụ tìm kiếm của trang web.

Qua việc tối ưu hóa SEO, trang web có cơ hội cao hơn để xuất hiện trên các trang kết quả tìm kiếm hàng đầu, thu hút được lượng lớn người dùng tiềm năng và tăng khả năng tiếp cận đối tượng khách hàng mục tiêu.

-**SPA** không thân thiện với SEO như **MPA**

**+UX**

-**SPA** cho trải nghiệm tốt hơn, nhất là các thao tác chuyển trang

**-SPA** đem lại trải nghiệm tốt hơn trên thiết bị di động

**+Quá trình phát triển**

**-SPA** dễ dàng tái sử dụng code (chia component) => làm 1 giao diện có thể sử dụng API cho web và mobile luôn

-**MPA** có thể chia cimponent nhưng sẽ phức tạp hơn => phải tạo nhiều API cho mỗi thiết bị sử dụng

**+SPA( CSR ) và MPA (SSR)**

-**SPA** còn được biết đến là **CSR –** Client Side Rendering ! Lý do là khi chúng ta gửi một request thì trình duyệt sẽ tải (render) trang web trên máy của chúng ta (phía client)

-**MPA** còn được biết là **SSR** – Server Sider Rendering ! Lý do là khi chúng ta gửi request thì phía server sẽ render trang web và trả về trình duyệt ! Sau đó mới hiển thị lên máy chúng ta

**+Phụ thuộc Javascript**

-**SPA** phụ thuộc hoàn toàn vào Javascript (không có JS là phế luôn). Vì khi chúng ta phát triển theo **SPA** thì trang web chỉ có một file.html duy nhất và tất cả các trang còn lại được xây lên nhờ Javascript !

-**MPA** có thể không cần Javascript ! mà vẫn có thể sử dụng nhiều file.html tương tác với nhau vẫn oke ! Nhưng thời điểm này mà không có JS thì sao gọi là web nữa

**+Làm việc với server**

-**SPA** thì khi nhiều người tương tác thì trình duyệt sẽ render trên từng máy của client nên sẽ giúp ích cho server rất nhiều

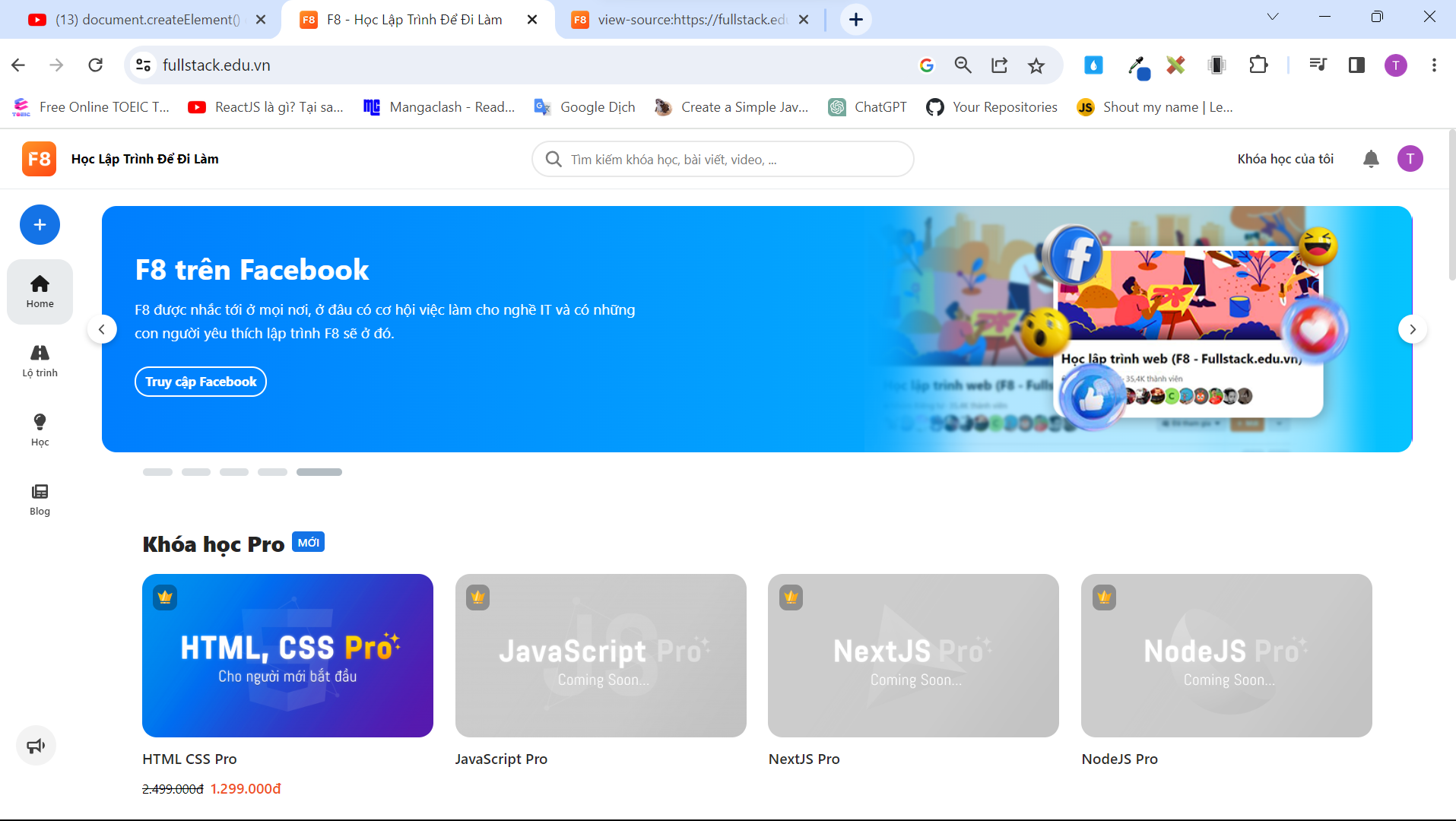
-**MPA** thì khi mỗi người thực hiện một request thì server phải nhận và render từng cái một ! Khiến server làm việc nhiều hơn nên chi phí khá tốn !

* 1. **Lựa chọn SPA hay MPA**

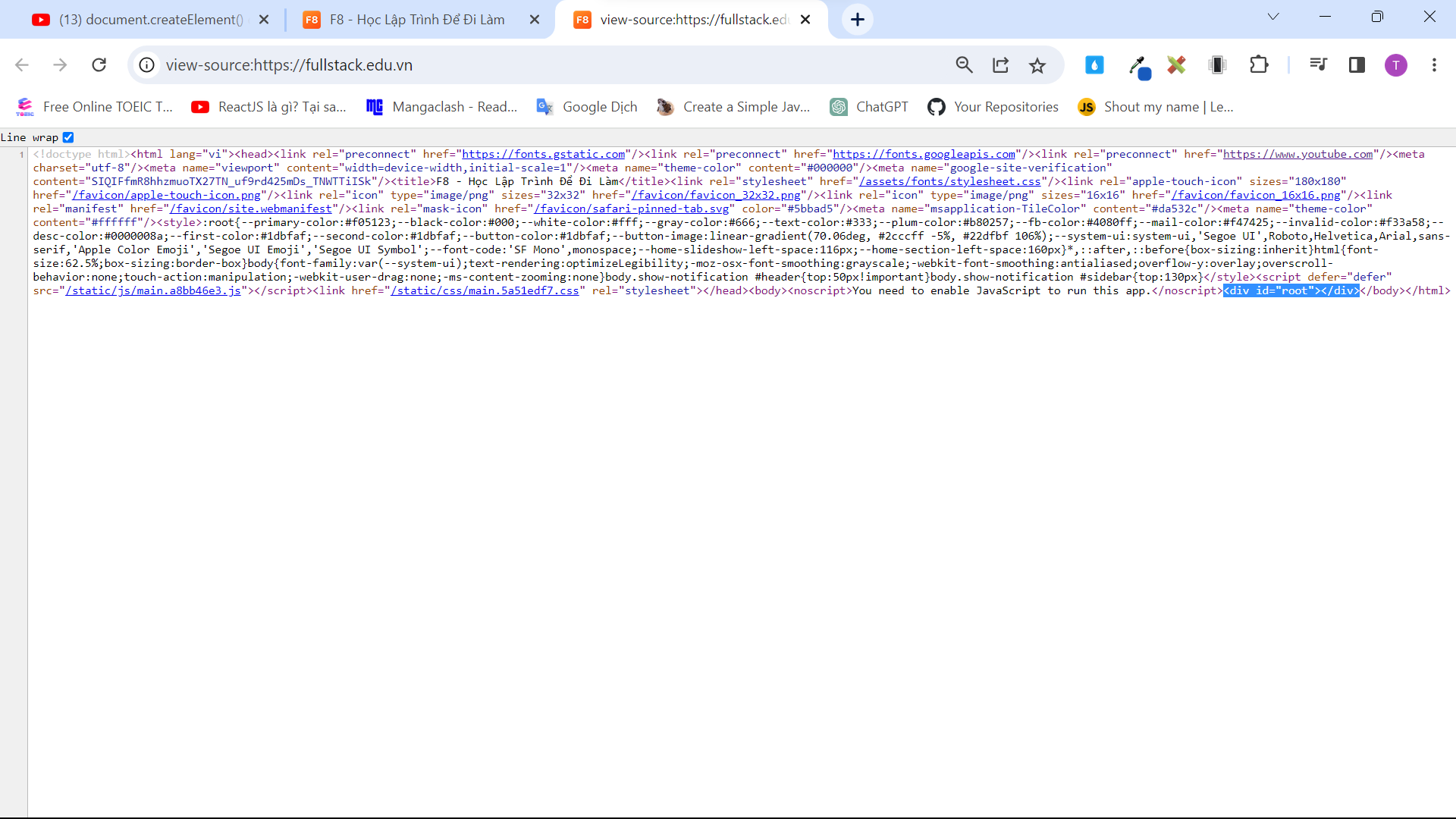
**-**Trường hợp nếu khách hàng yêu cầu built một trang web về giới thiệu công ty, sản phẩm,… mà không có nhu cầu mở rộng hoặc phát triển trang web hay chạy trên mobile thì chúng ta nên sài **MPA** để có thể phát triển ngay và luôn !

**-**Còn nếu như khách hàng muốn mở rộng thêm nhiều tính năng mới lạ, phát triển luôn bên mobile và muốn trang web phải mượt mà, trải nghiệm người dùng tốt nhất thì chúng ta hãy dùng **SPA**

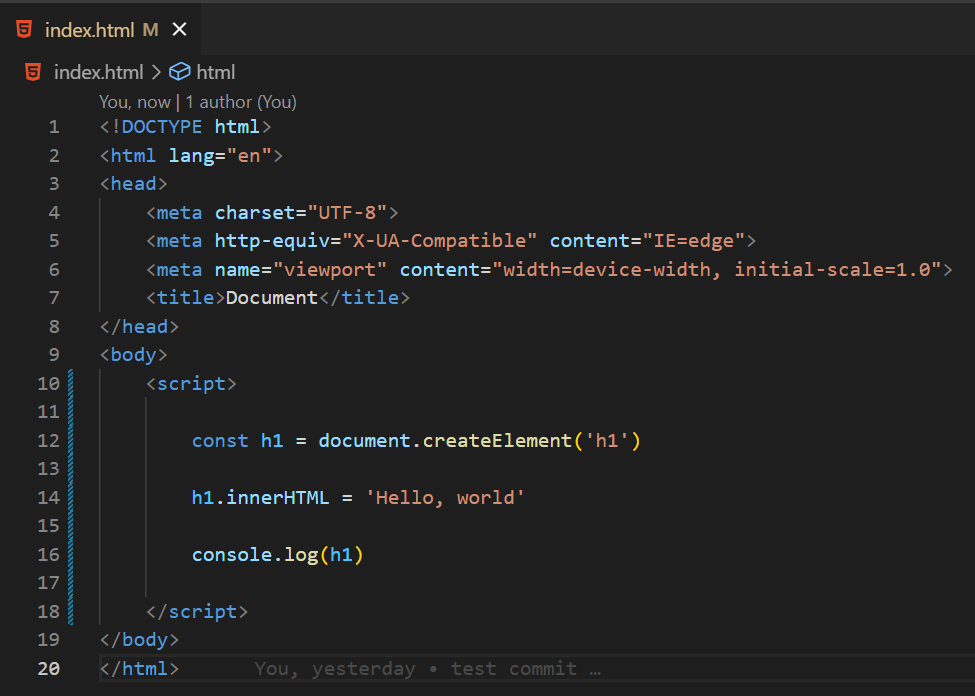
1. **Document.creatElement() (Tạo Element trong thẻ <script>)**



-Chúng ta có trang web của fullstack.edu.vn ! Có thể thao tác qua lại trên thanh điều hướng



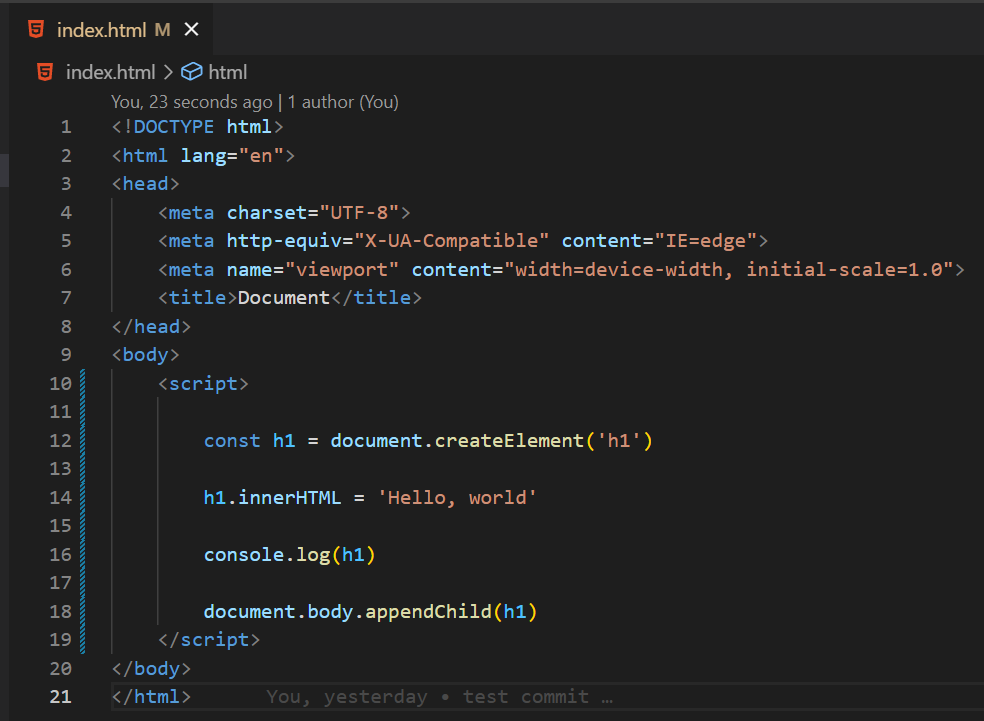
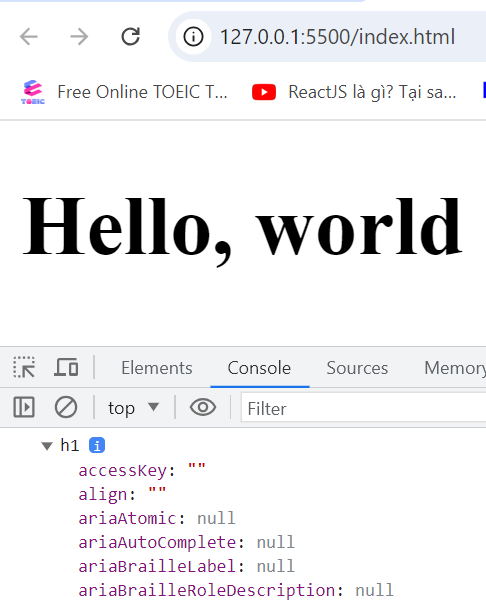
-Nhưng điều thú vị ở đây là ! trong phần source code của trang web chúng ta không hề thấy cả núi code HTML cả mà đa phần là JavaScript (chỉ vài dòng)! Và chỉ có duy nhất 1 thẻ <div> có id là **root** ! Để code được giao diện trên chắc cả ngàn dòng code mà sao lại chỉ hiện chó 1 thẻ <div> duy nhất ! đó là cách reactJS làm việc nói chính xác hơn là **SPA**

** **

**-**Để mô phỏng lại thì chúng ta sẽ làm ví dụ nhỉ ! Chúng ta tạo một element là <h1> bằng phương thức **document.creatElement()** và cho nó text Node = ‘Hello, world’ thông qua thuộc tính **innerHTML** sau đó in ra màn hình console !

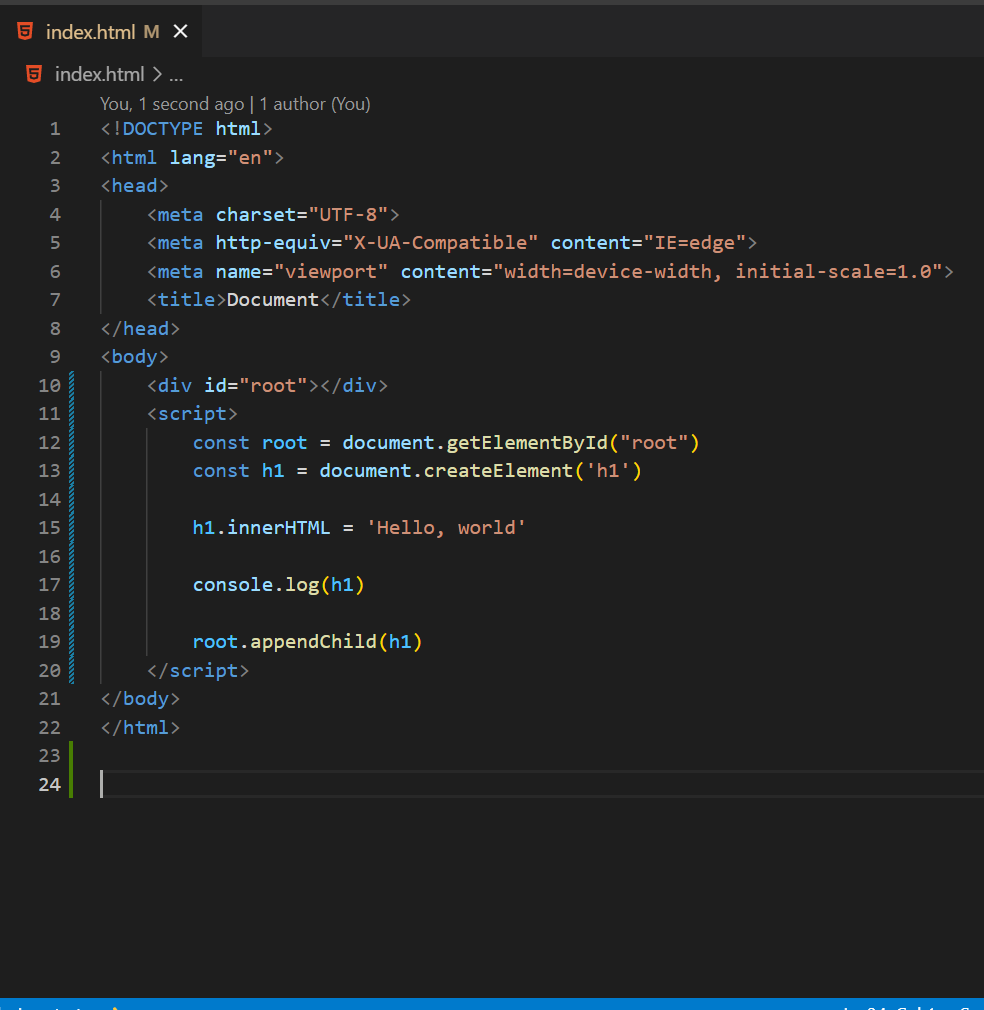
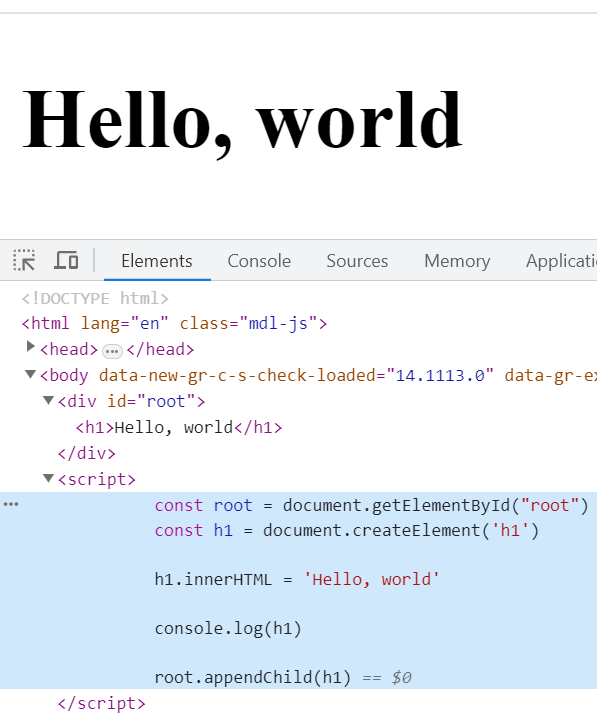
**-**Nhưng để ý là phía trên màn hình giao diện của chúng ta vẫn trắng tinh chưa có gì !

* 1. **Document.appendChild() (Thêm element con vào element cha)**

-Để thêm thẻ <h1> vừa tạo hiển thị lên trang web thì ta sử dụng phương thức **appendChild()** và **document.body** có nghĩa là đối tượng **document** đi đến thẻ **<body>** và thêm con cho nó (kiến thức DOM tree)

-Nếu mở source code trang lên thì sẽ thấy không có dòng code **<h1>** nào cho đến khi **file.js** được tạo mới hiển thị **<h1>**

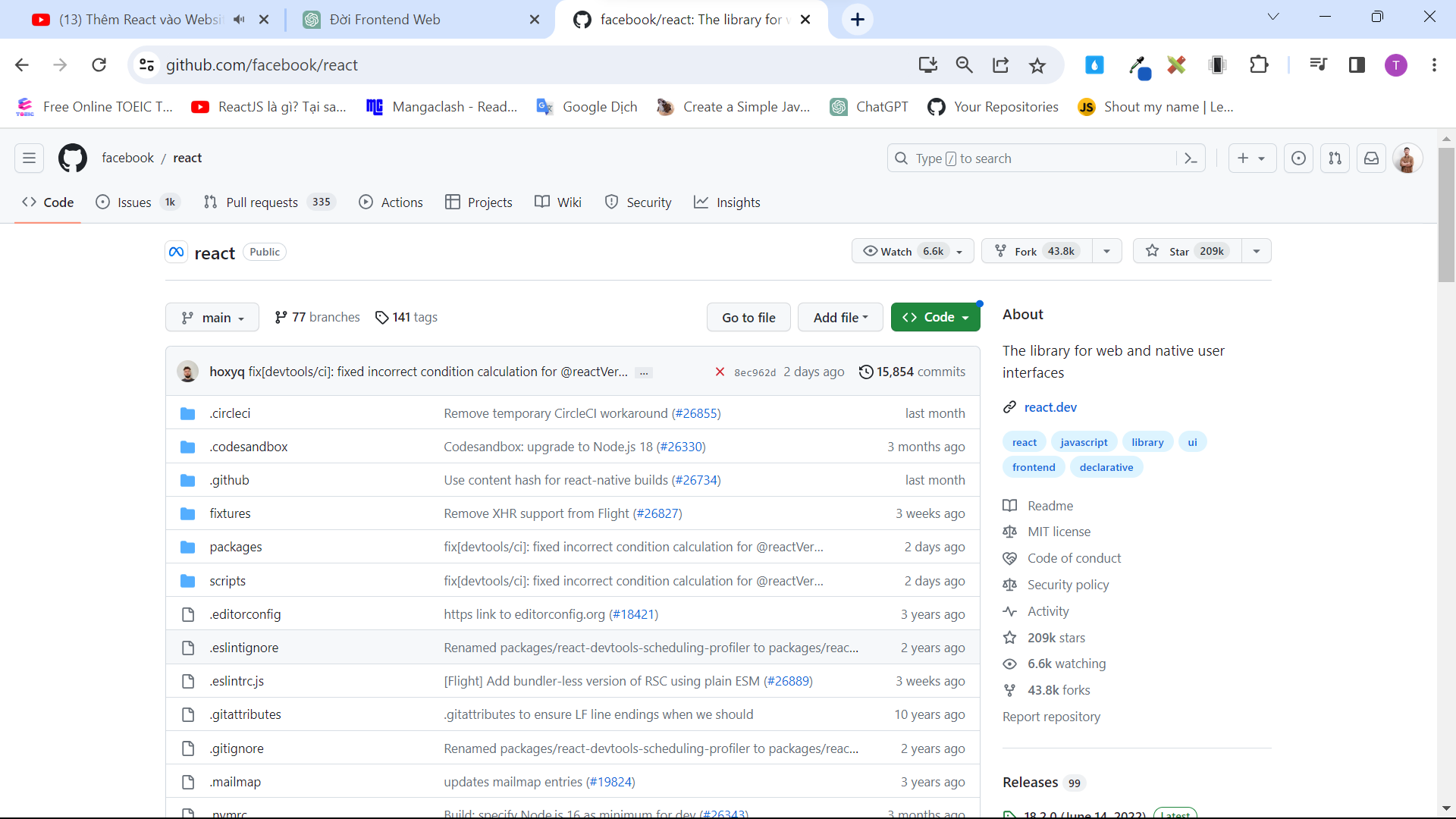
 

-Để làm giống với fullstact.edu.vn thì chúng ta chỉ việc tạo một the **<div>** có **id = “root”** và gọi đến Element root đó thay **document.body** thành **root** thôi

-Đó là cách **SPA** làm việc ! và thư viên ReactJS cũng hoạt động như vậy

1. **Thêm thư viện ReactJS vào Website**
   1. **Cách thư viện được lưu trữ và phân phối**

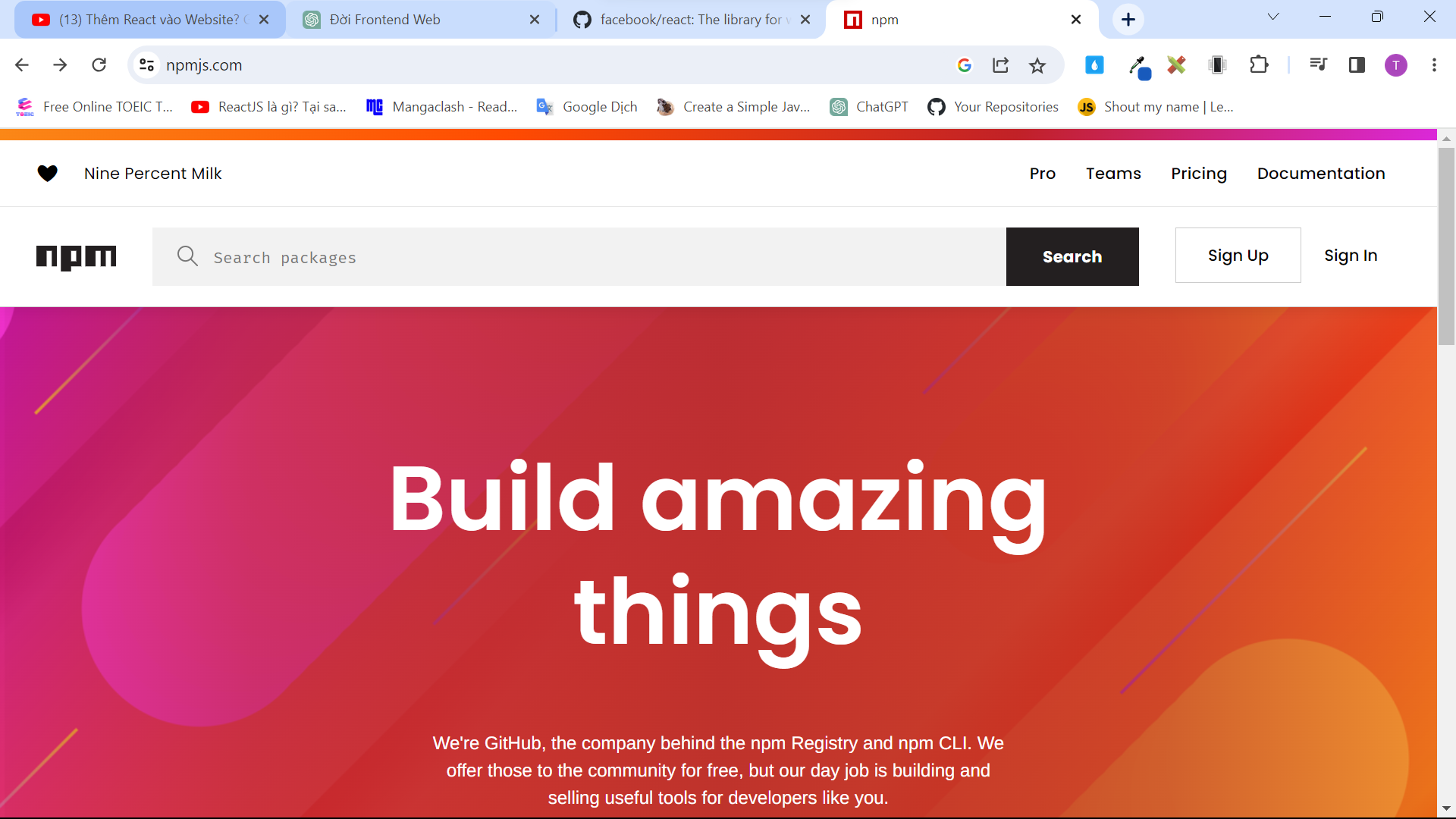
**+Github**

****

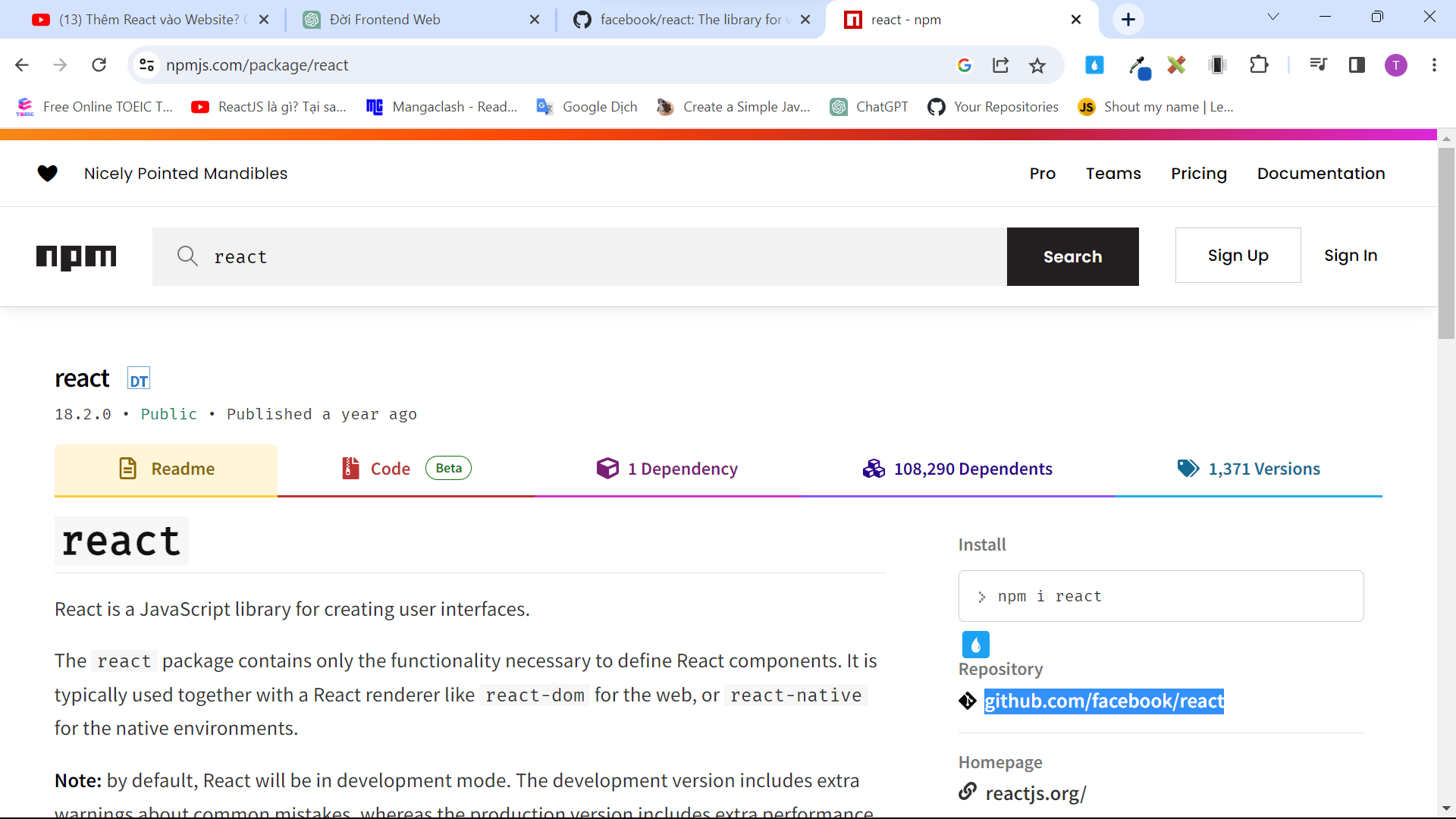
-Là nơi mà lưu trữ gốc của thư viện React ! Chúng ta có thể lấy về và tạo ra thư viện react riêng cho mình cũng được !

-Nhưng không nhất thiết phải tải hết của người ta về ! mà chỉ sử dụng một vài phương thức thôi !

**+NPM**

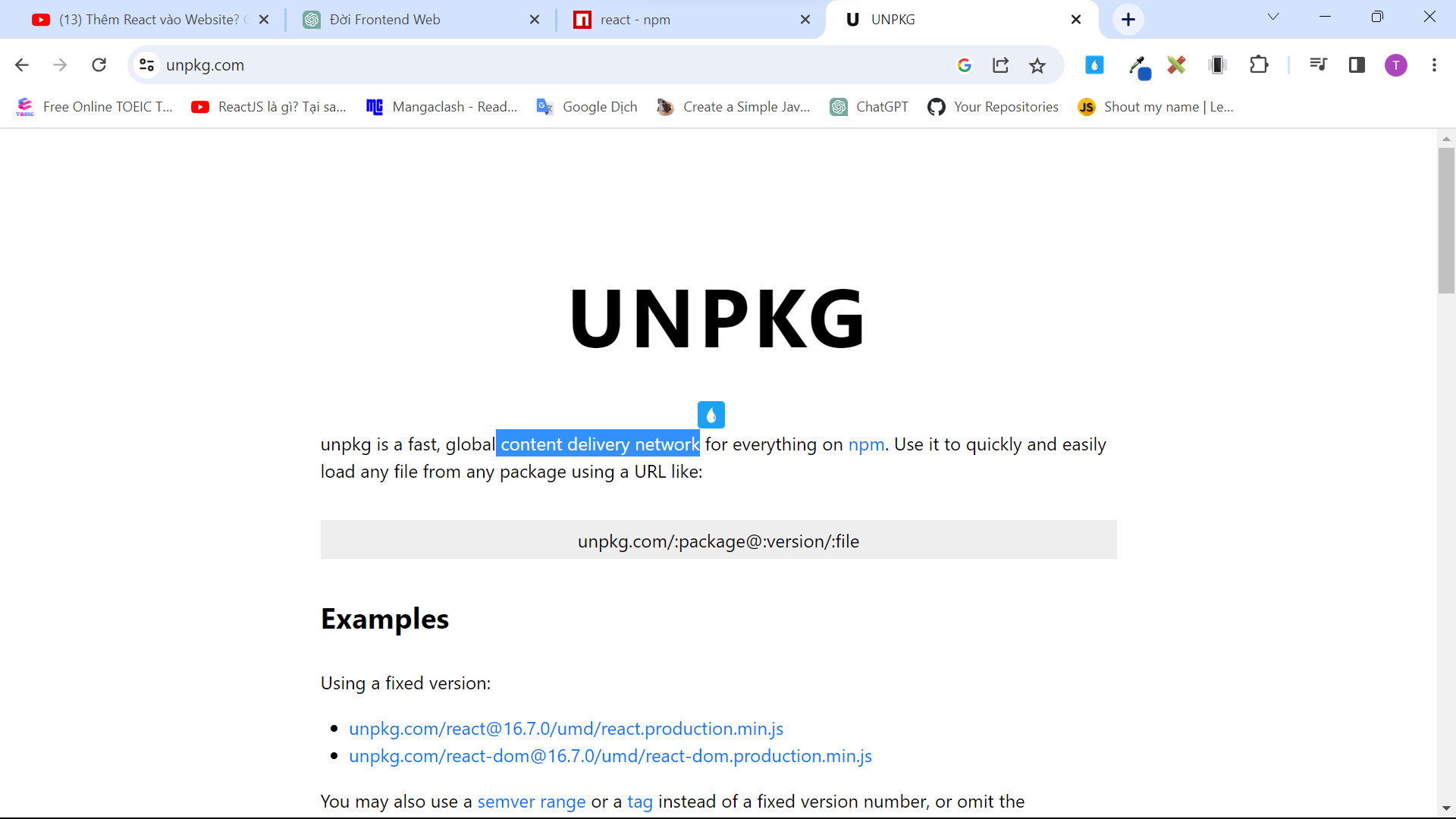
****

**-NPM** là Node Package Manager là nơi quản lý thư việnvà công bố cho người dùng tải về



-Chúng ta nhập từ khóa react vào và chọn thư viện react !để ý nguồn đúng với link github cho thấy cả 2 là một thôi !

**+UNPKG (cdn)**

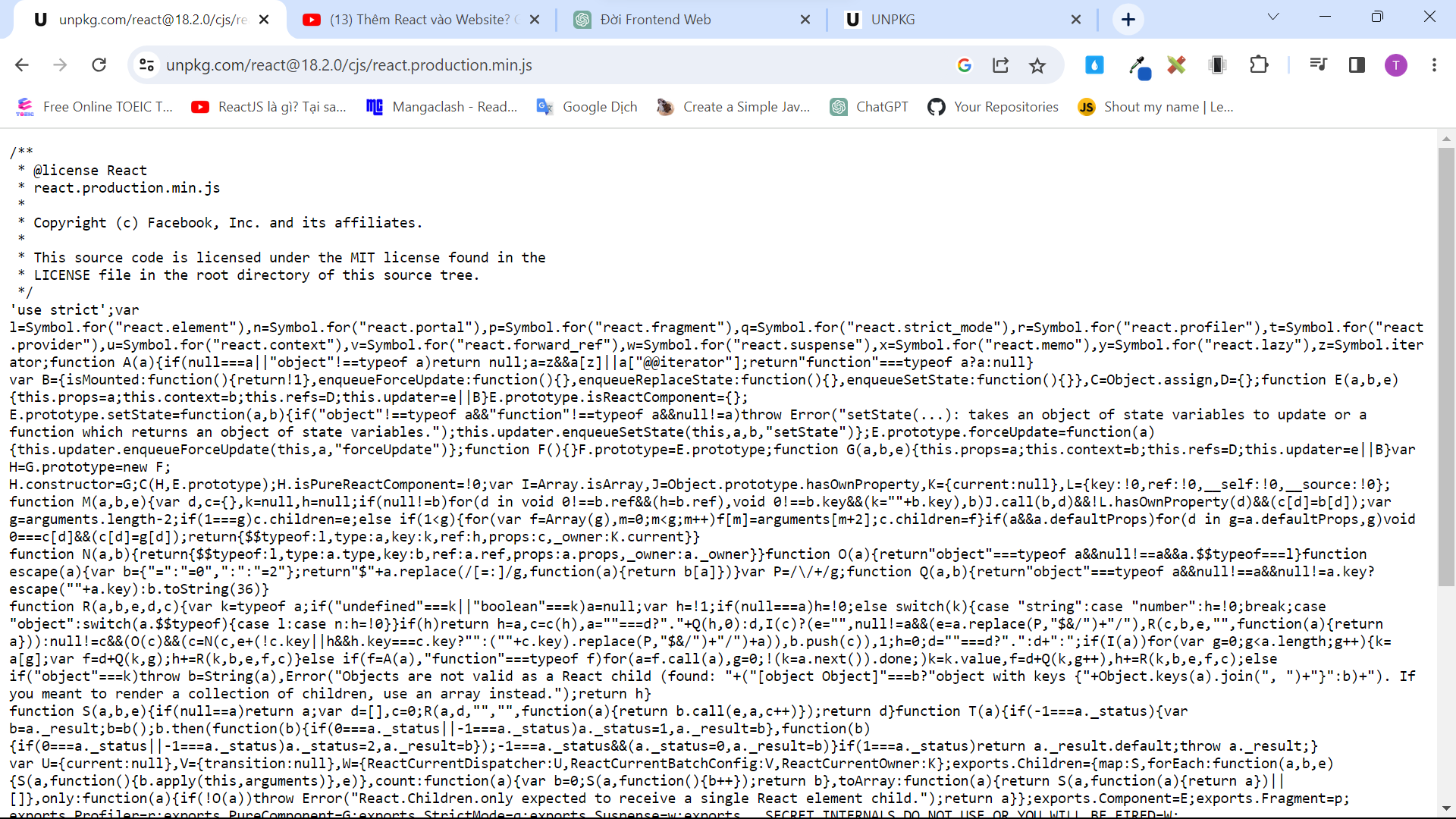
****

**-**Có nhiều cách để tải thư viện bên **npm** về lắm nhưng nhanh nhất vẫn là sài **cdn (**tự tìm hiểu nhé**)**

****

**-**Chúng ta có thể gõ **unpkg.com/tên thư viện** để có thể biết nó có những phiên bản gì

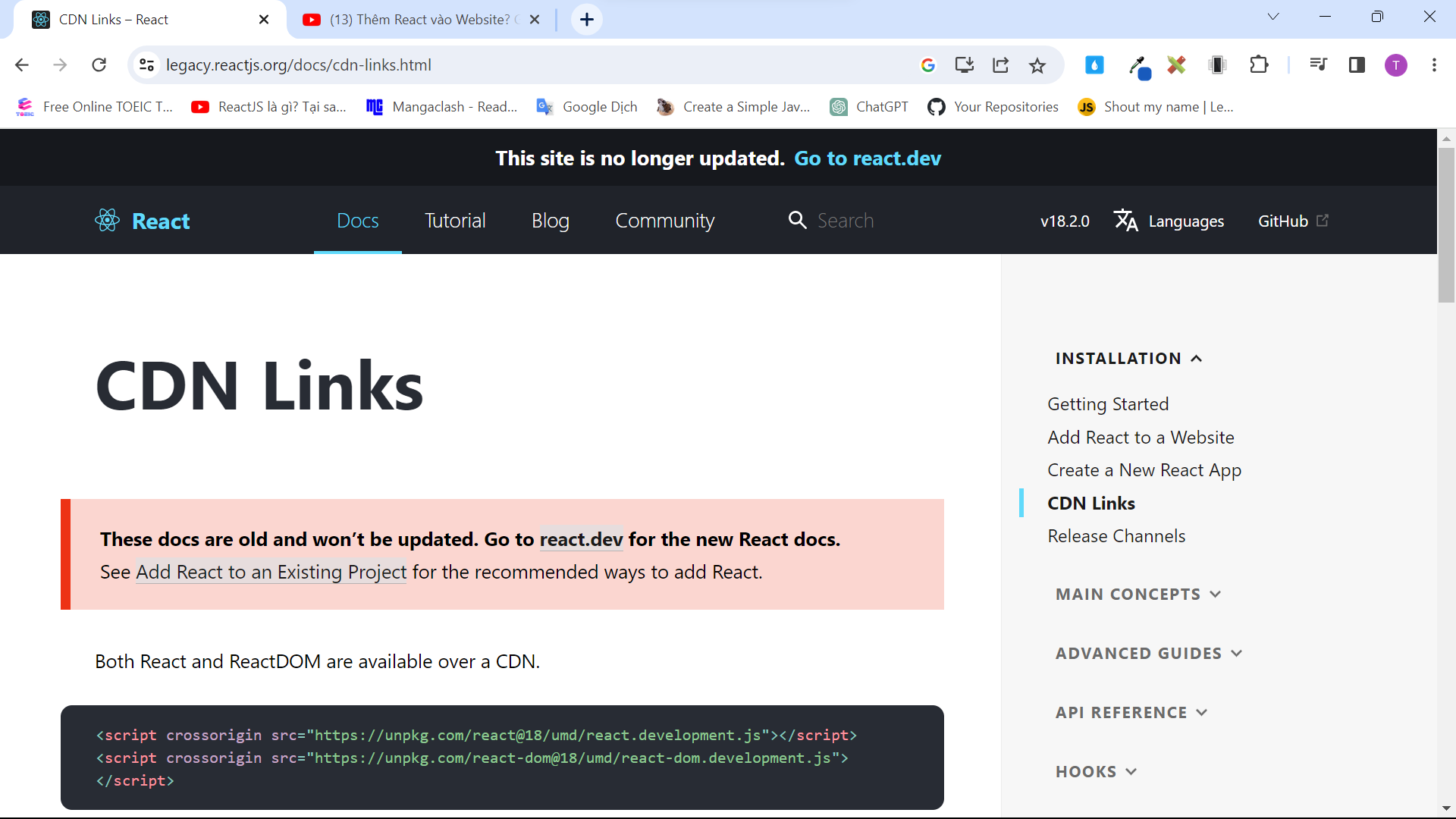
**-**Ở đây gồm phiên bản cho dev (báo lỗi, bug) và cho người dùng cuối (gọn nhẹ, dễ dàng sử dụng) !

****

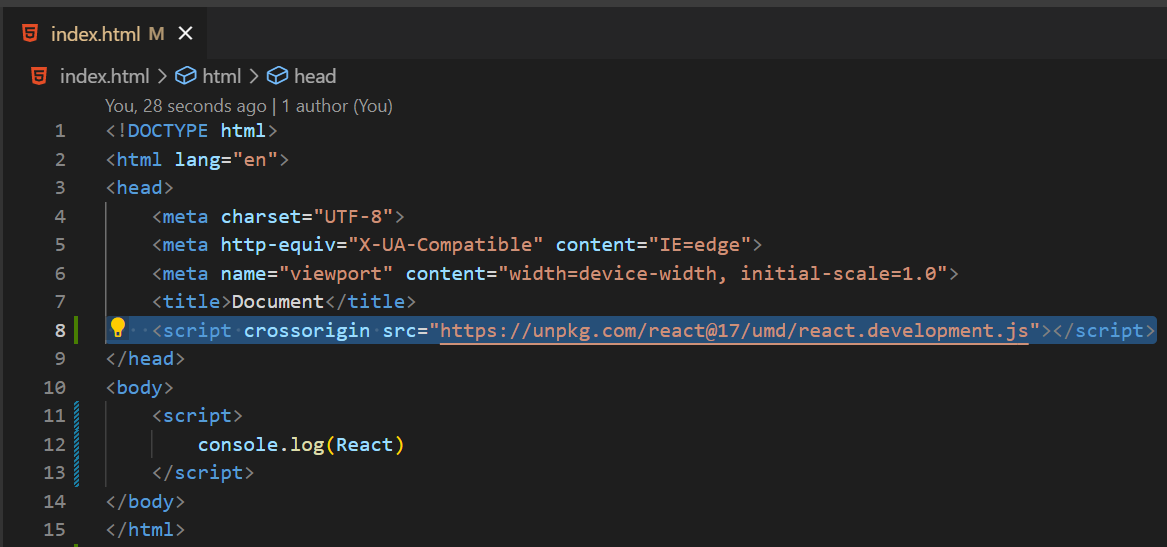
**-**Chúng ta có thể đi sâu hơn để thấy toàn bộ thư viện React chỉ vỏn vẹn một file ! Nhưng đội ngũ kỹ sư của facebook đã bỏ rất nhiều kinh nghiệm và công sức mới có được cho chúng ta sài

**-**Vì vậy suy cho cùng chúng ta chỉ có thể sử dụng các phương thức của thư viện để build lên sản phẩm của mình chứ không ai code giùm ta gì cả và hoàn toàn là 100% code Javascript ! nên chưa có nền tản cứng về Javascript thì gãy !

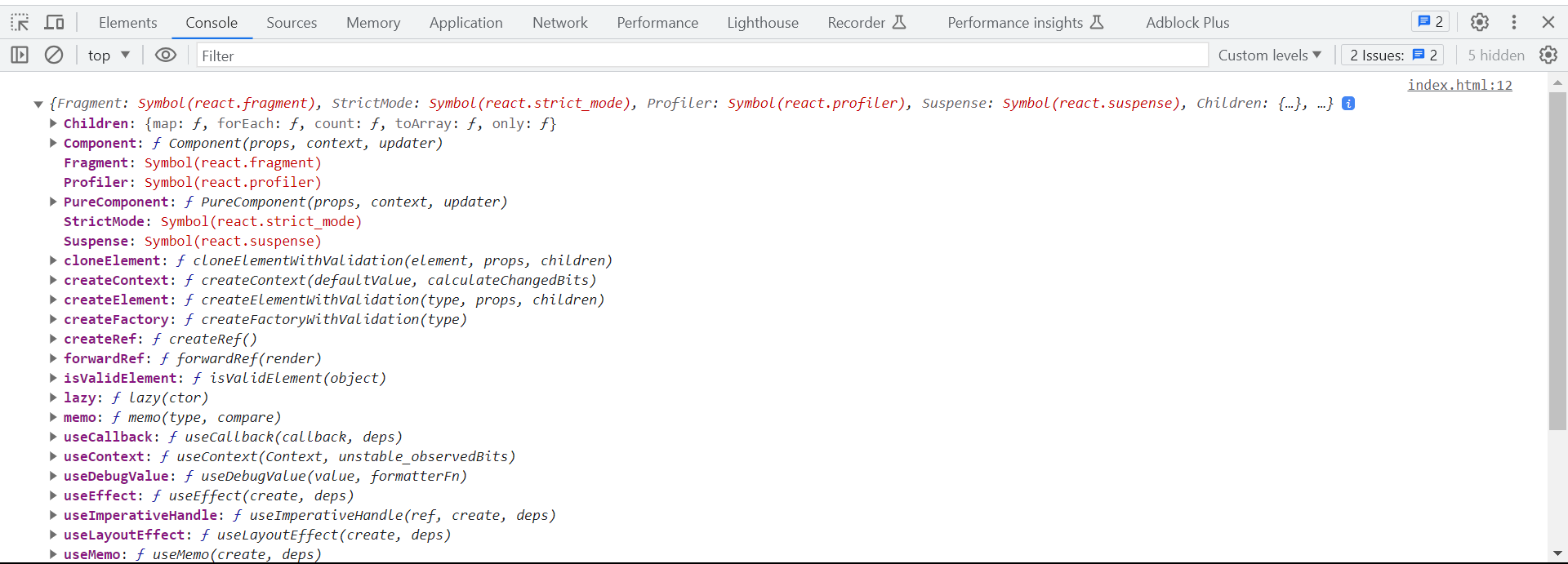
* 1. **Thêm thư viện ReactJS**

****

**-**Để thêm thư viện **ReactJS** thì chúng ta sẽ truy cập vào trình duyệt gõ cdn reactJSlà ra ! (**Trong bài số 3 này sẽ áp dụng phiên bản React 17.0.2**)

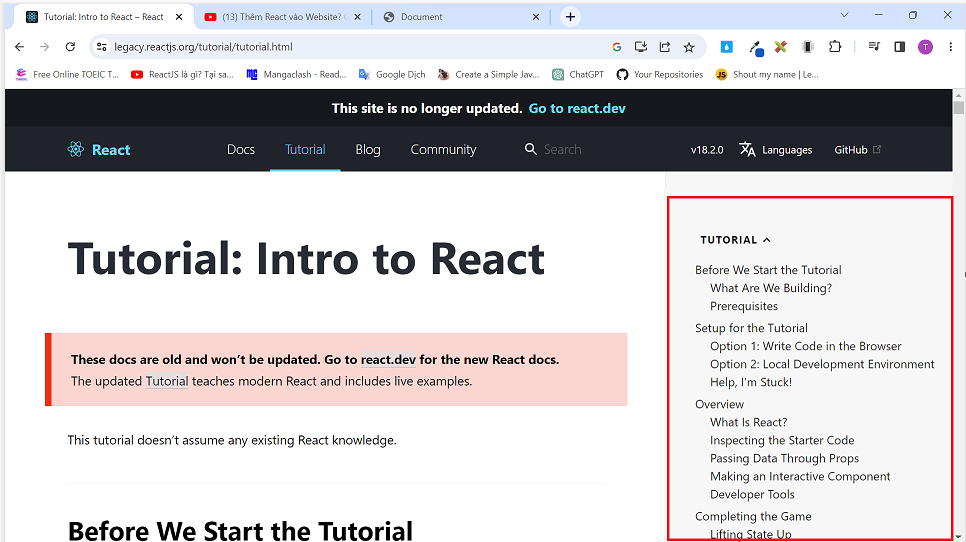


-Sau đó dán **cdn** vào **file.html** để kiểm tra add thư viện thành công thì chúng ta thử in đối tượng React xem ! ( =)) nếu khác phiên bản thì đổi thành 17 sài nhé )



-Nếu ra được tất cả thuộc tính và phương thức của React thì đã add thành công ! Sau đó chúng ta chỉ việc chõ đến sài thôi !

* 1. **React official docs**



**-**Trong trang chủ react thì có phần Tutorial khá bài bản và đi từ gốc đến ngọn ! nên đọc để hiểu thêm về react làm việc như thế nào !

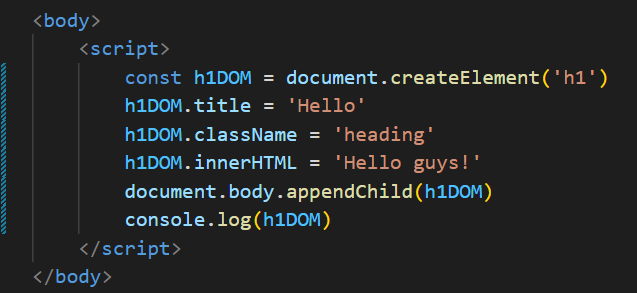
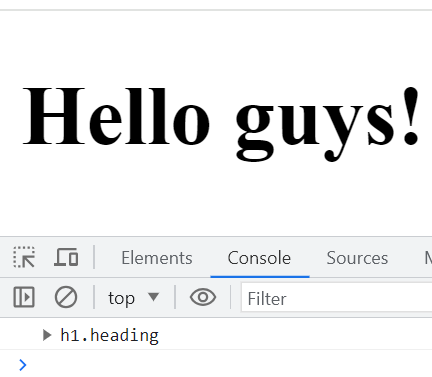
1. **React.creatElement() (Tạo Element bằng thư viện ReactJS)**

**-Document.creatElement() và React.creatElement()** thật chất đều có nhiệm vụ thêm Element vào website bằng code JS ! Khác cái là một cái ta dùng DOM element và một cái dùng thư viện React

* 1. **So sánh giữa document.creatElement() và React.creatElement()**
     1. **Thêm Element vào website**

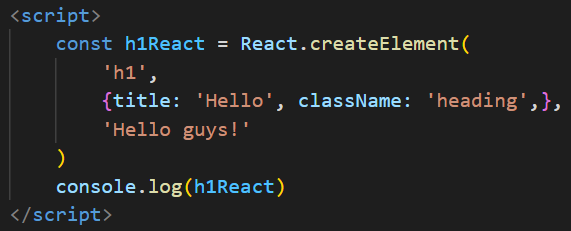
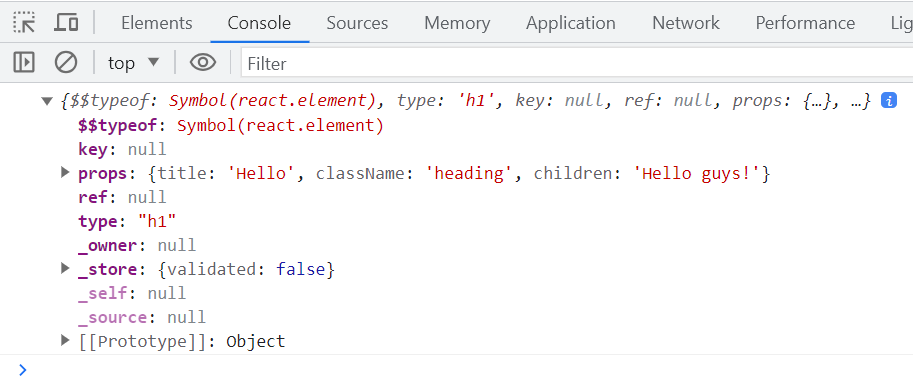


-Giả sử chúng ta muốn thêm Element trên vào website thông qua code JS thì sẽ làm như thế nào ?

**** ****

-Đây là cách DOM document chúng ta hay dùng ! Tạo element bằng phương thức **creatElement()** và thêm các thuộc tính **title**, **className**, **innerHTML** tương ứng ! Cuối cùng là thêm Element con vào element cha bằng phương thức **appendChild()**

**-**Vì **body** là Element đã có sẵn trong DOM rồi nên chúng ta chỉ việc gọi  **appendChild()** để thêm thẻ h1 vừa tạo vào thôi !

**** ****

**-**Còn đây là khi chúng dùng thư viện **ReactJS** để tạo Element ! **createElement()** của ReactJS khá khác so với DOM document !

**-**Cú pháp: React.creatElement(type ,props, children, n)

**-**Ở đây chúng ta thấy **createElement()** có 3 đối số chính:

**+**đối số 1: là **type** tương tự như đối số **createElement()** của DOM document

**+**đối số 2: là **props** nó là một đối tượng chứa các thuộc tính như title, className, style… tượng tự như thêm thuộc tính đối với DOM document

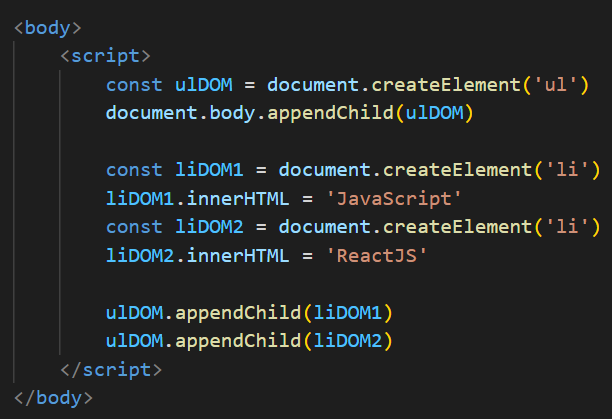
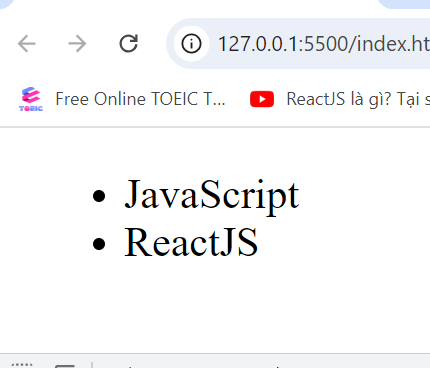
+đối số 3: là **children** tương tự giống như ta dùng **innerHTML** trong DOM ! Để ý là **children** thì react tách ra thành đối số 3 và nó cũng chính là một **props**

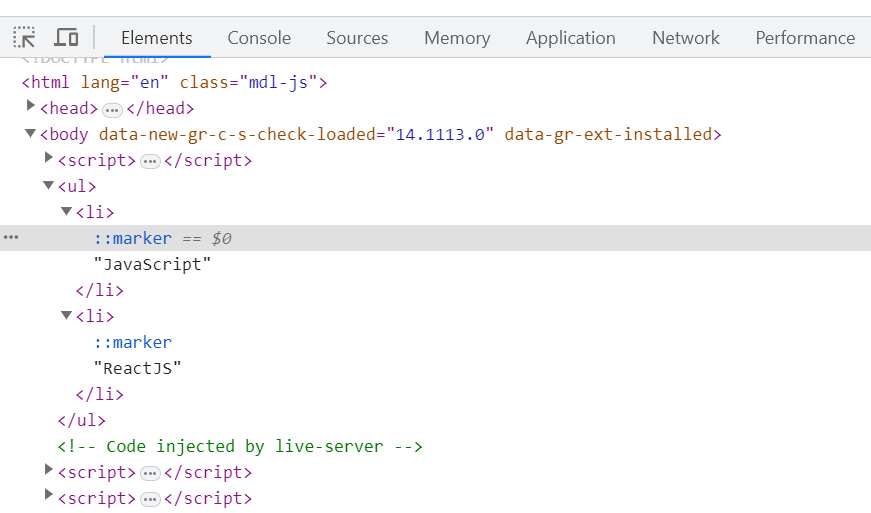
+đối số 4: là n children ! có thể có nhiều children kể từ đối số 3 !

* + 1. **Thêm Element lồng Element**

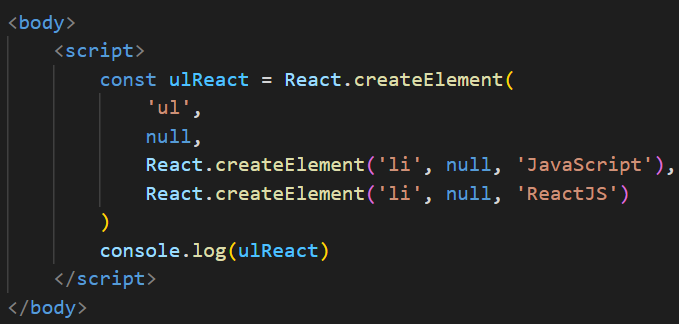
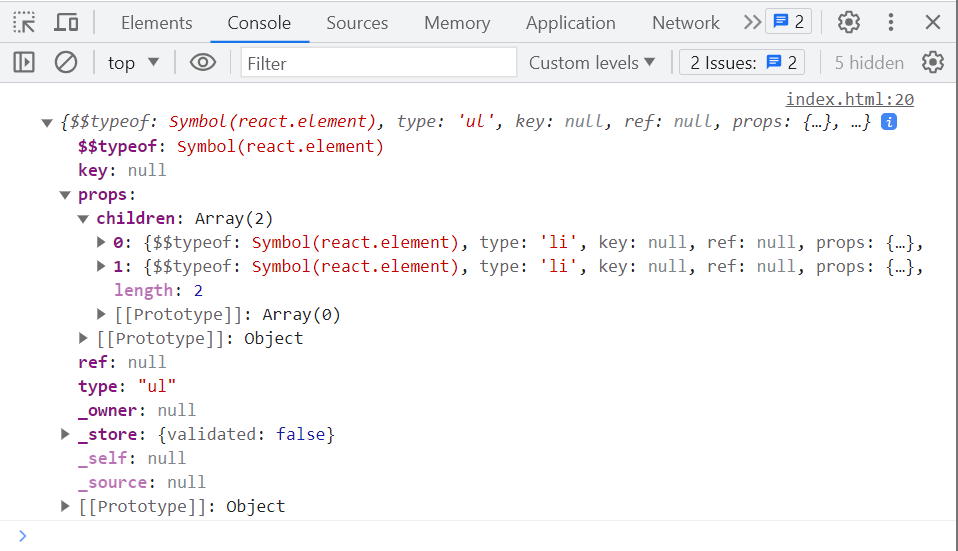
****

**-**Chúng ta sẽ add các Element trên vào website để xem DOM document và React sẽ làm như thế nào nhé !

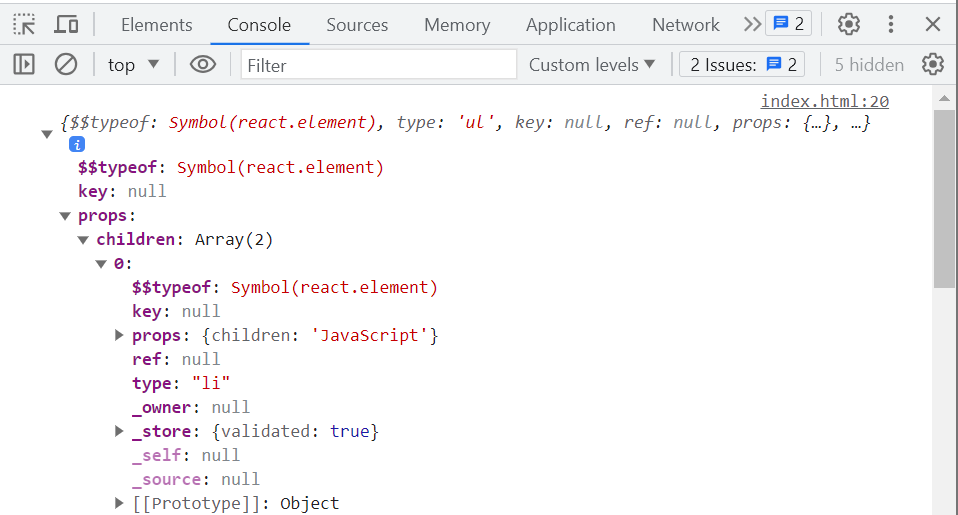
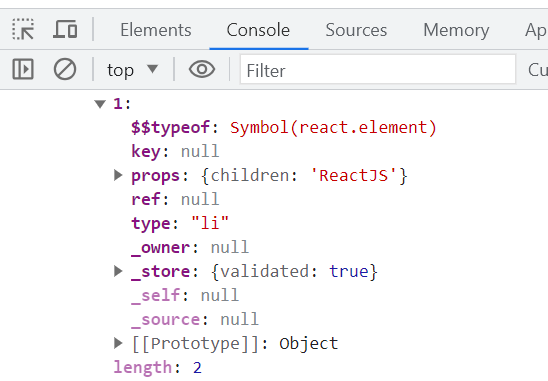
 



-Đây là thêm Element khi chúng sử dụng DOM document ! Chắc không cần giải thích lại nhỉ ! chỉ cần phối hợp các phương thức với nhau thôi !

-Đây là cách ReactJS sử lý Element lồng nhau ! Trong code nhìn ngắn gọn hơn !

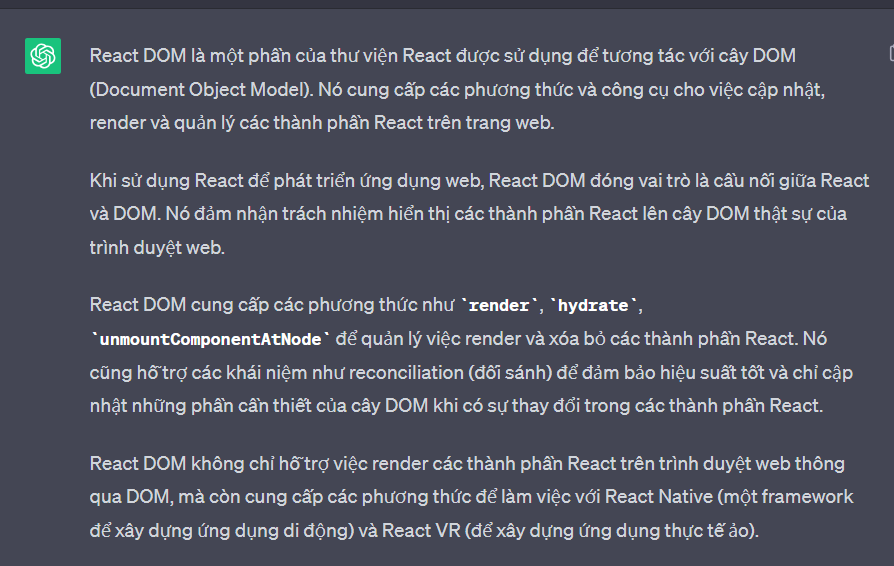
 

-Nếu để ý thì chúng ta sẽ thấy 2 thẻ <li> được đặt trong một Array **children !** Và bên trong có một **props** nữa chứa children con !

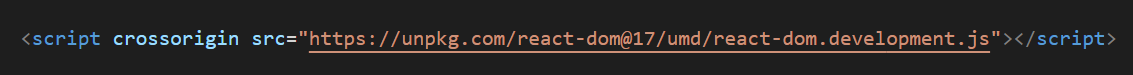
-Ngoài ra chúng ta có thể thêm các thuộc tính cho Element !

1. **React-DOM (kết nối thư viện ReactJS và DOM tree)**



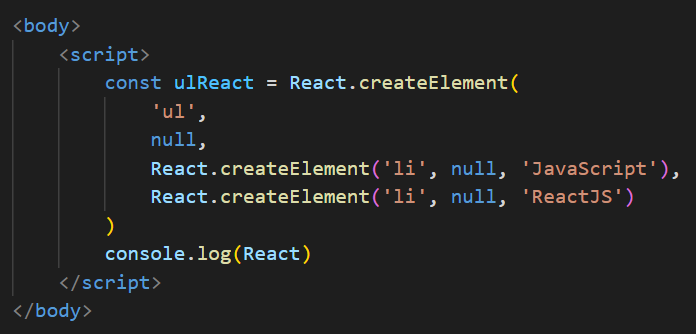
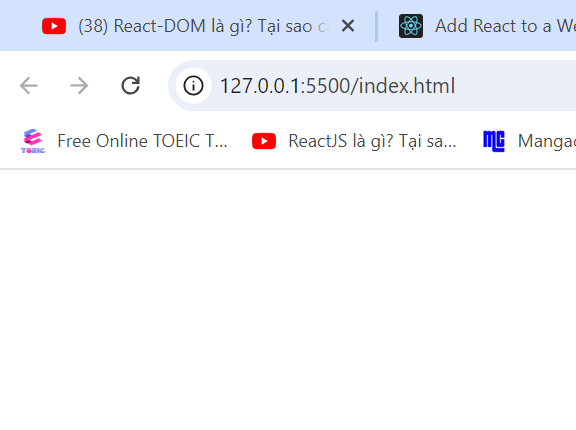
-React-DOM và React-Native có vai trò giống nhau là kết nối giữa thư viện React và cây DOM trên trình duyệt !

-React tách ra thành React-DOM và React-Native (tự tìm hiểu nhé) để dễ mở rộng hệ sinh thái ấy mà



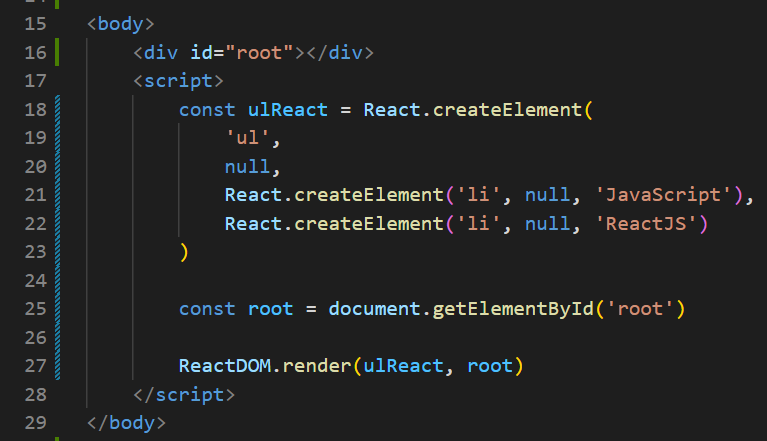
-Để cài React-DOM chúng ta chỉ cần **search: cdn ReactDOM** là và copy **<script>** vào **file.html** (Nhớ là để để sau cdn của ReactJS nhé)

* 1. **ReactDOM.render()**

-Giả sử chúng ta có một tạo một vài Element bằng **React.createdElement()** là 1 thẻ <ul> bên trong chứa 2 thẻ <li>. Nhưng đây chỉ là tạo thôi chứ chưa có thể render ra trình duyệt được !

Vì vậy phương thức **render()** trong ReactDOM sẽ giúp chúng ta làm việc đó.

-Khi ta làm việc với React thường thì chúng ta sẽ chỉ cần tạo một thẻ <div> trong file.html còn tất cả code còn lại thì sẽ do JS tạo ra (cơ chế SPA)

-Vì vậy ở dòng số 16 chúng ta tạo một thẻ <div> có id = “root”

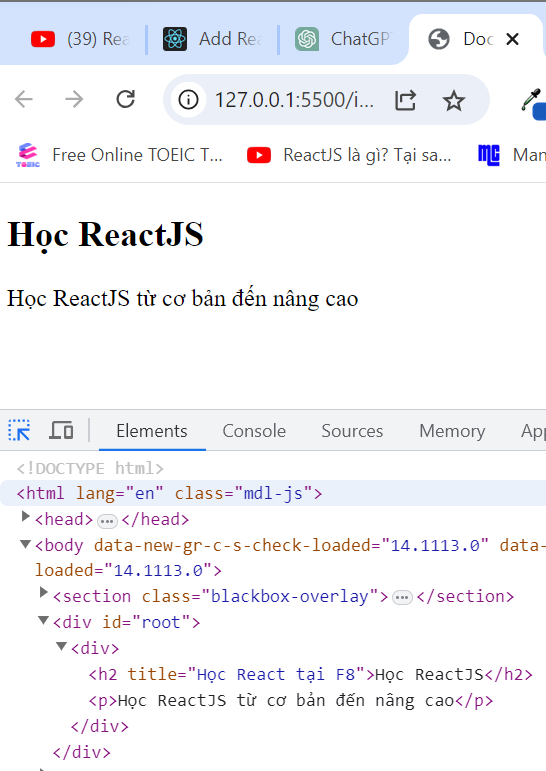
-Dòng số 25 thì chúng ta lấy Element root đó !

-Cuối cùng sẽ sử dụng phương thức **render()** của ReactDOM ! Lưu ý là ReactDOM ghi cho đúng chính tả nhé !

-Phương thức **render()** gồm 2 đối số chính:

+Đối số 1: Được gọi là **component** đây là thành phần mà chúng ta muốn hiển thị trên browser

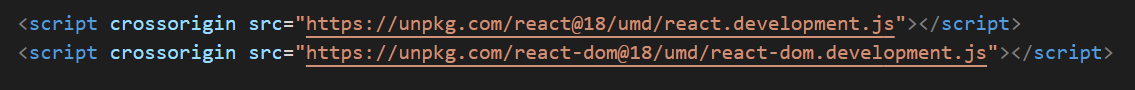
+Đối số 2: Được gọi là **container** là phần tử chứa **component**

-Tương tự như trên !

1. **ReactJS version 18**

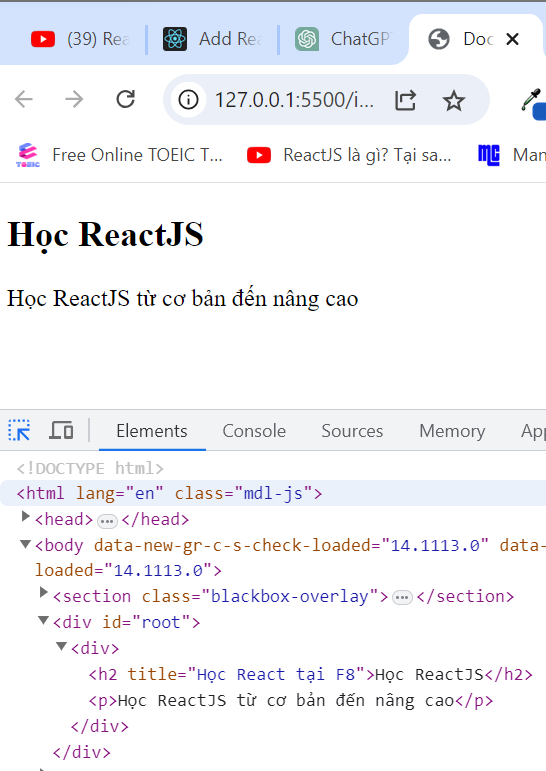
**-**Hiện tại bản mới nhất của ReactJS là phiên bản 18 ! chúng ta sẽ cập nhật lên nhé !



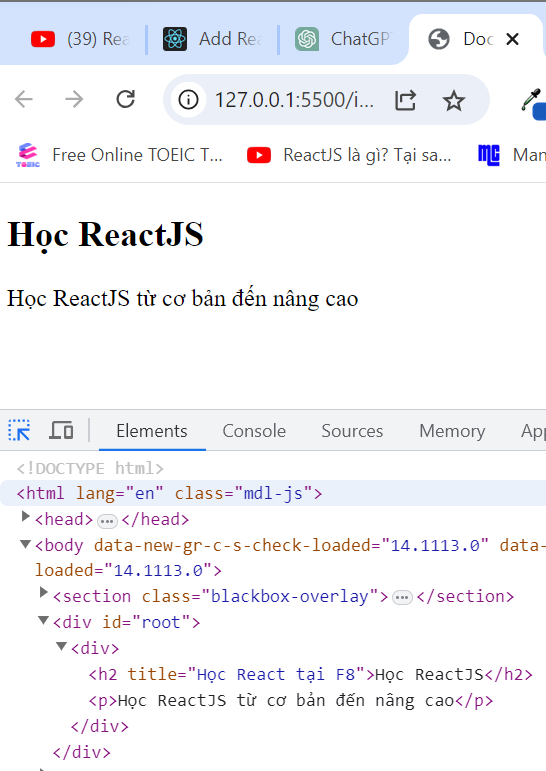
-Để cập chúng ta **search: cdn reactJS** nhé !Hoặc đơn giản là thay đổi số 17 thành 18 trong file.html thôi là xong rồi !

* 1. **ReactDOM.render()**

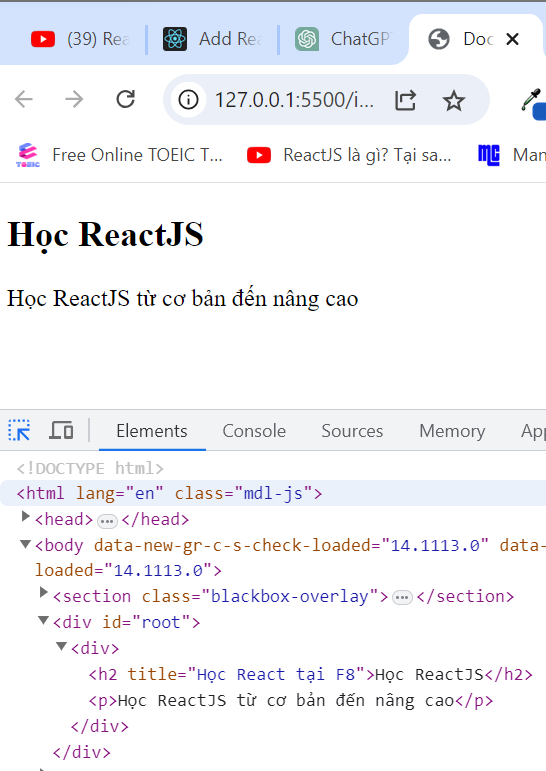
-Trong phiên bản 18 thì phương thức **render()** trong ReactDOM có thây đổi chút !



-Đây là phiên bản 17 cũ ! Hàm render() có 2 đối số truyền vào là **component** và **container**

-Còn đây là phiên bản ReactJS version 18 ! chúng ta phải lấy được **container** trước bằng phương thức **createRoot()** của ReactDOM (trong vd này thì sẽ được gán vào biến **root**) ! sau đó sử dụng đối tượng này **render()** ra **component** (quên 2 thằng này thì coi lại bài 5.1)

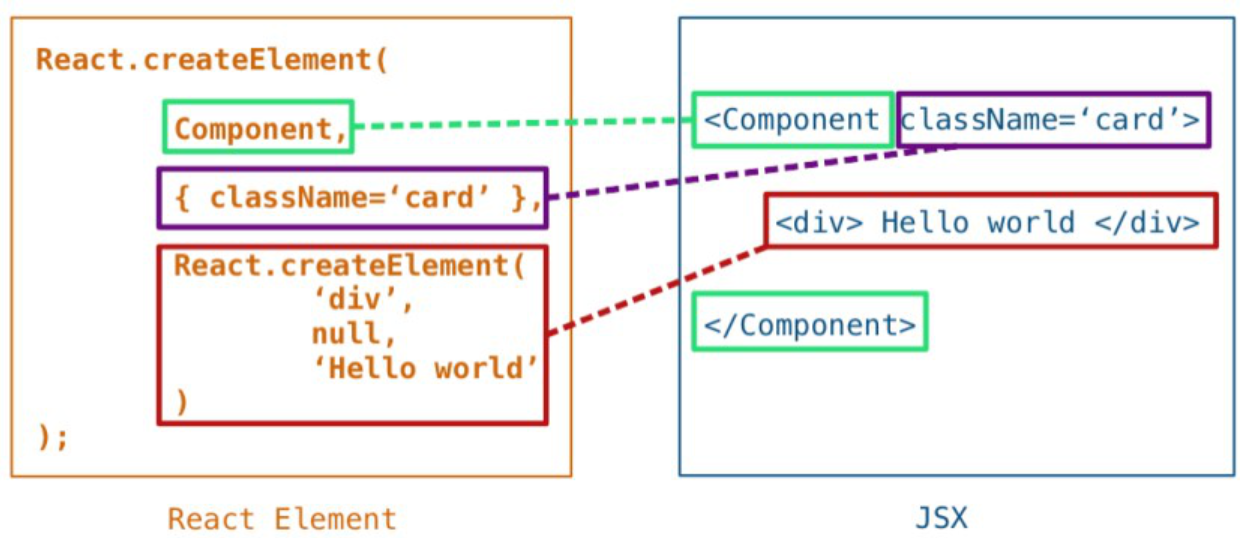
****

**-**Hoặc có thể viết như này nếu thấy khó hiểu !

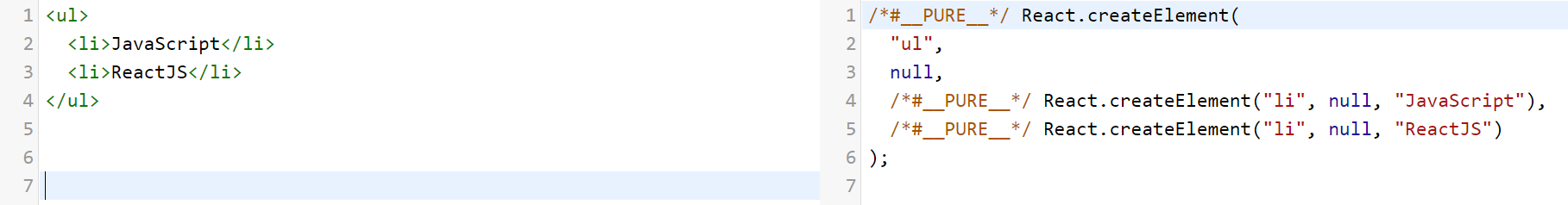
1. **JSX (Javascript XML)**

-Có thể thấy những bài trước chúng ta đã tạo ra các Element bằng Javascript chính xác hơn là thông qua thư viện ReactJS ! Nhưng trong có vẽ khá cực nếu tưởng tượng chúng ta code cả trang web thì sao ?! Và may mắn là thực tế không ai code như vậy đâu ! Nhưng nhờ như vậy chúng ta mới hiểu ReactJS chạy như thế nào và **SPA** ra làm sao !

-Ngắn gọn hơn là **JSX** giúp chúng ta có thể viết **HTML** trong file.js



-Vì không thể nào một website mà chúng ta có thể code thuần ReactJS như những bài trên ! do đó **JSX** ra đời giúp chúng ta viết code dễ dàng ngắn gọn hơn ! Thật tuyệt vời khi biết được là **JSX** có cú pháp giống như **HTML** là các thẻ đóng mở **<></>** ! Thực chất ngôn ngữ **HTML** là mở rộng của **JSX** !

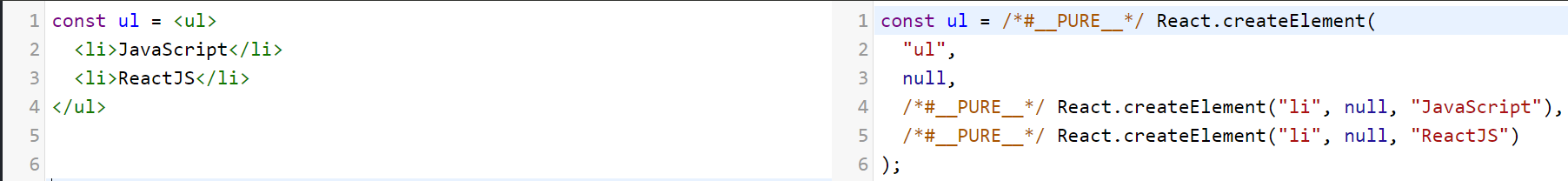
****

**-**Bên phải là chúng ta sẽ code theo **JSX** ! Nhìn khá giống với **HTML** nhỉ ! Và bên trái thì quá quen thuộc là **React Element** ! Có nghĩa là chúng ta sẽ viết **JSX** và nó sẽ tự biên dịch sang **React Element** cho chúng ta

* 1. **JSX không phải là HTML**

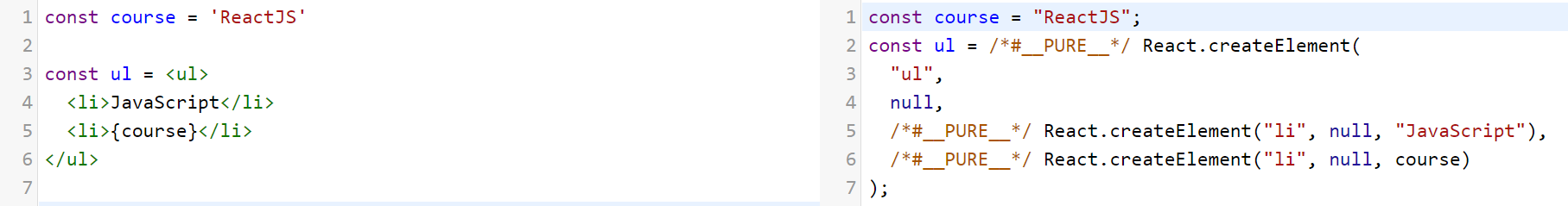
-Chúng ta phải biết rằng **JSX** chỉ giống cú pháp **HTML** thôi ! chứ nó không giống nhau hoàn toàn nhé ! Có 2 lưu ý về **JSX** mà chúng ta cần biết khi sử dụng.

**+Có thể gán JSX vào biến**

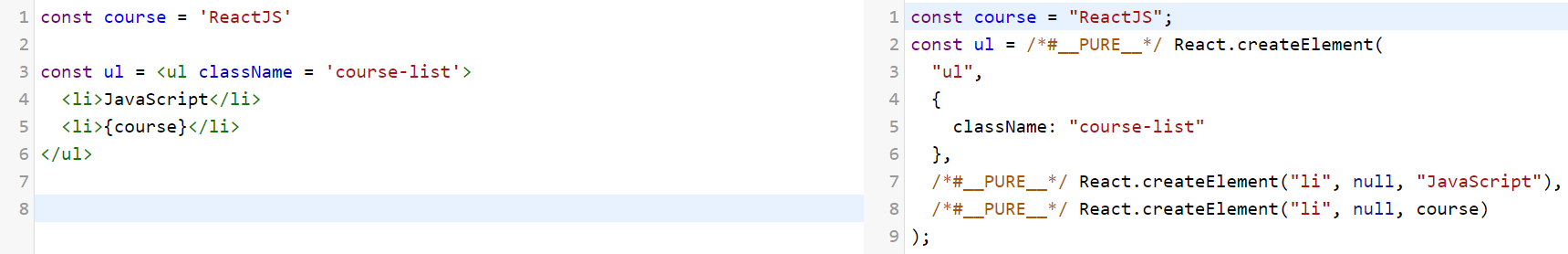


-Đây cũng là điều khác biệt so với HTML ! nhưng chúng ta đã thử code chay React Element thì việc đó quá đổi bình thường !

**+Có thể sen lẫn code Javascript trong JSX**



-Bằng cách dùng dấu **{}** chúng ta có thể nhúng code Javascript (tất cả về JS như biến, hàm, loop) vào **JSX.** Như ví dụ trên thì khi biên dịch sang ReactJS nó sẽ hiểu **children** ở đây là biến **course.**



-Chúng ta có thể truyền thuộc tính và nó sẽ biên dịch sang ReactJS cho chúng ta ! đương nhiên phải tuân theo cách thêm thuộc tính vào DOM nhé ! chứ không phải code HMTL thuần đâu !



-Lưu ý là khi muốn truyền một object vào thì chúng ta cần sử dụng 2 dấu **{{}}** lòng nhau ! dấu **{}** đầu tiên là nơi để viết JS thôi ! còn dấu **{}** bên trong nữa mới là truyền object

* 1. **Babel (Thư viện trung gian giúp JXS làm việc được với React-DOM)**

**-**Hiện nay phiên bản phổ biến mà chúng ta hay dùng để code JS là ES6 trở lên ! vì vậy các máy tính hoặc trình duyệt đời cũ sẽ có thể không đọc được code của chúng ta ! vì vậy cần có công cụ **Babel** để có thể biên dịch cho các máy và trình duyệt cũ hiểu được

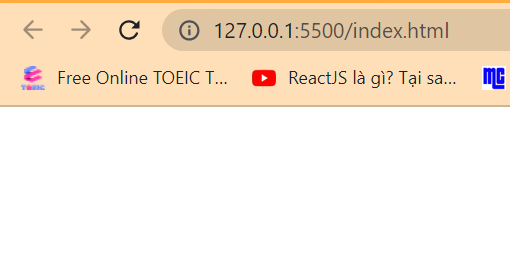
**-**Như chúng ta được biết là **React-DOM** chỉ nhận **React Element** thôi! Vì vậy **JSX** không thể nào truyền thằng vào **React-DOM** được ! do đó cần phải có bên trung gian liên kết 2 thằng này ! vì vậy cần thư viện **Bable**

**-**Và đương nhiên muốn sài **JSX** thì phải cài **Bable**

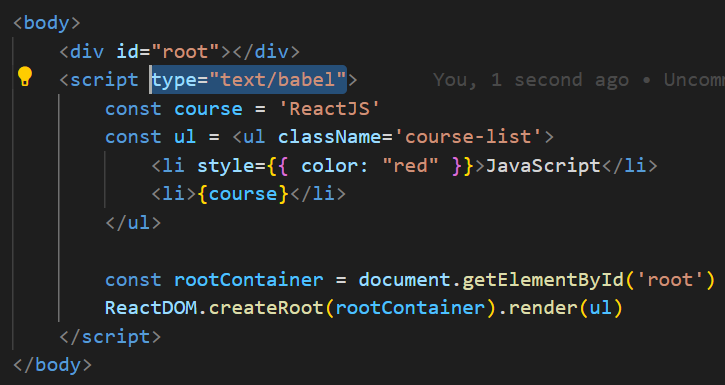
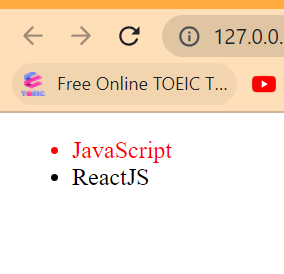
* + 1. **Cách cài thư viện Bable**



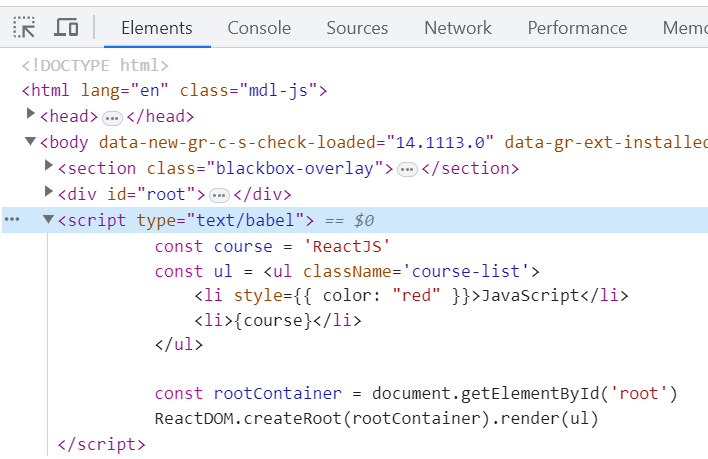
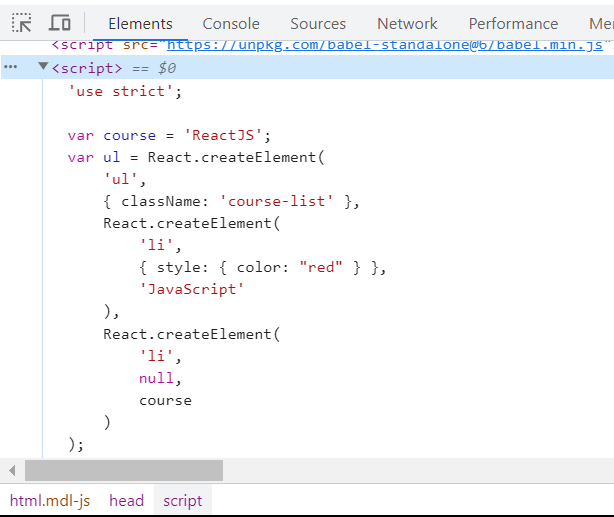
**-**Vì là làm đơn giản nên chúng ta cũng lấy cdn của thư viện **Bable** bỏ vào file.html luôn (nằm chung chỗ với cdn ReactJS và ReactDOM á)

-Chúng ta thử code **JSX** vào nhé (vì lười nên cope cái đoạn trên dán vô luôn). Có thể thấy là… không có gì sảy ra cả ! Thật ra là cái **<script></script>** mà chúng ta viết DOM nó vẫn còn hiểu là code JS thuần ! nên chúng ta thêm code HMTL vào sẽ báo lỗi cú pháp ngay !

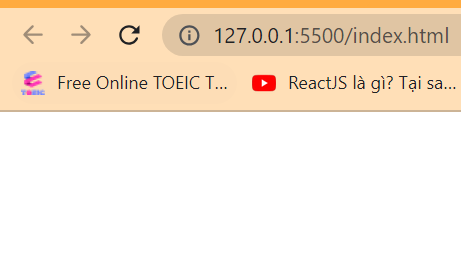
-Vì vậy chúng ta sẽ định dạng kiểu thư viện **Bable** cho thẻ <script> của chúng ta bằng cách thêm **type** như ở trên

-Ban đầu DOM sẽ đọc **<script>** của chúng ta ở **<body>** được viết bằng **JSX** và xác định nó là kiểu **Bable** sau đó ở **<header>** nó sẽ tạo ra một cái **<script>** và biên dịch đoạn **JSX** thành **ReactJS**.

-Giờ đây chúng ta không cần phải tự viết **React.createElement()** nữa mà chỉ cần cài thư viện Bable và code theo cú pháp **JSX** vô cùng đơn giản thôi !

* 1. **JSX render Arrays**

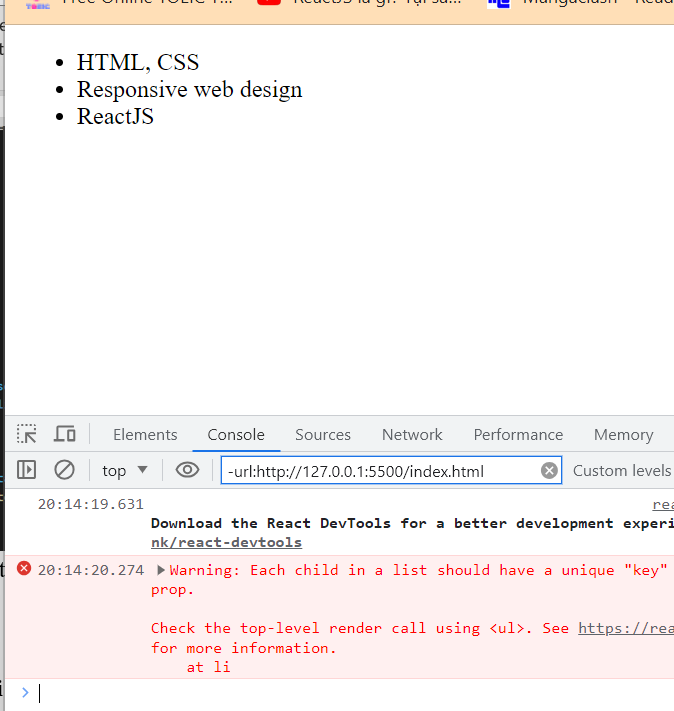
 

-Giả sử chúng ta có một Array là **course** ! Bên trng có các đối tượng – mỗi đối tượng có một thuộc tính là name ! làm cách nào mà chúng ta code **JSX** Để có thể render mảng ra trang web ?

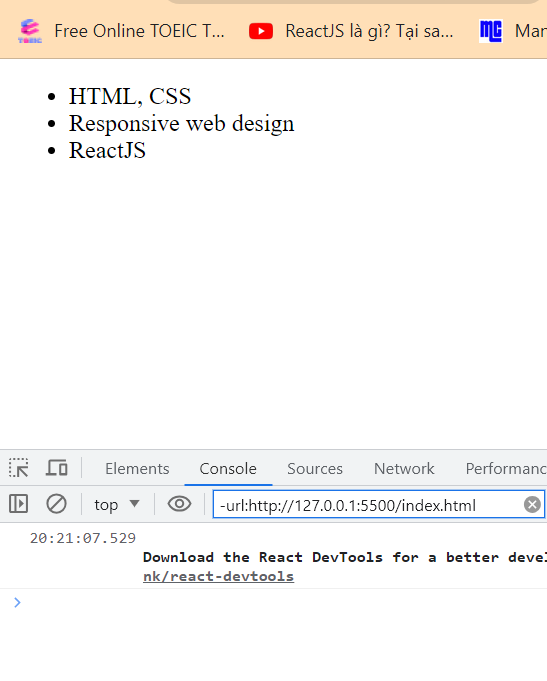
 

-Đầu tiên chúng ta tạo thẻ **<ul></ul>** và chèn code Javascript vào ! Sử method **map()** trong Array để trả về một mảng mới ! Cuối cùng là return ra các thẻ <li></li> có giá trị là **course.name**

* + 1. **Gặp lỗi khi render từ mảng (prop “key” )**

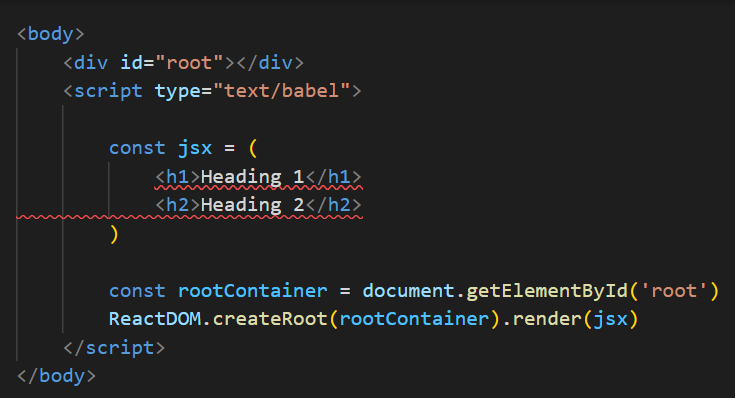
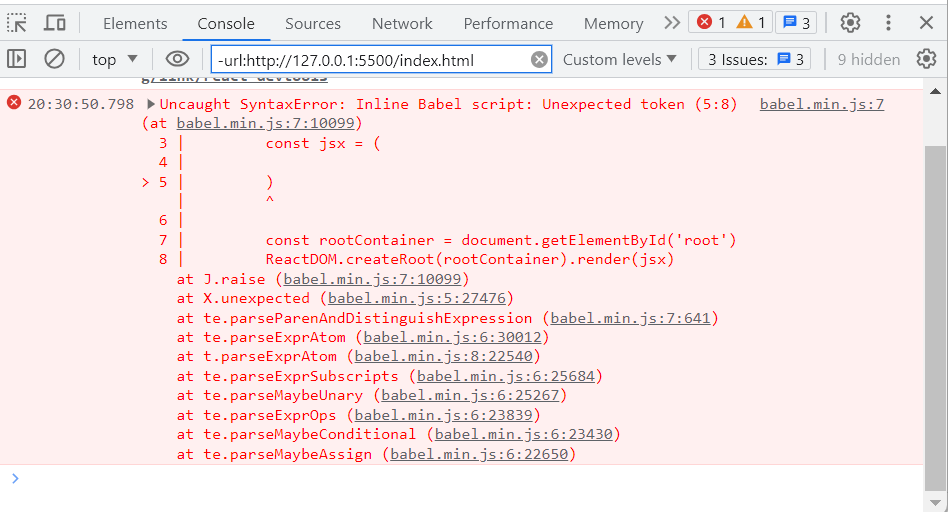


-Có thể thấy dù code chúng ta vẫn hoạt động tốt ! nhưngthư viện **Babel** vẫn cảnh báo chúng ta rằng mỗi children cần có một **key** duy nhất ! (children ở ví dụ này được hiểu là **<li></li>**)

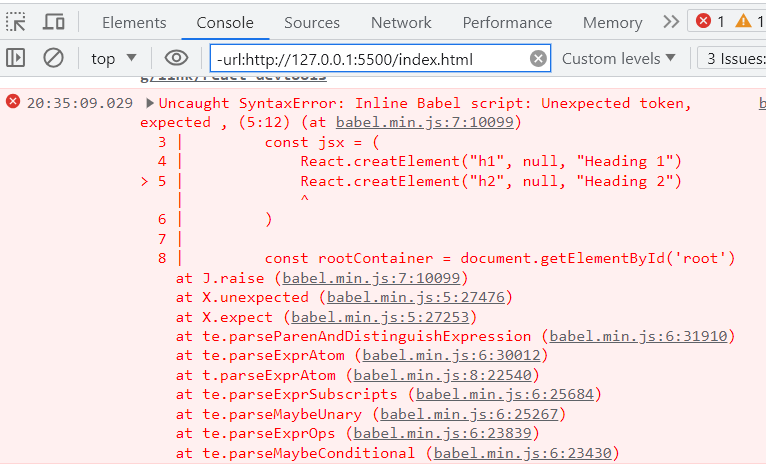


-Để làm việc đúng theo thư viện **Babel** chúng ta có thể thêm một tham số **index** vào dòng số 34 ! sau đó ở thẻ **<li></li>** chúng ta thêm thuộc tính **key={index}** ! giờ đây mỗi children đều có một **key** duy nhất không trùng nhau (có nhiều cách để làm nhé).

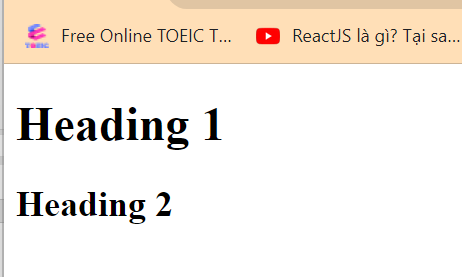
* + 1. **Chỉ có thể render một Element trong JSX ?**

**** ****

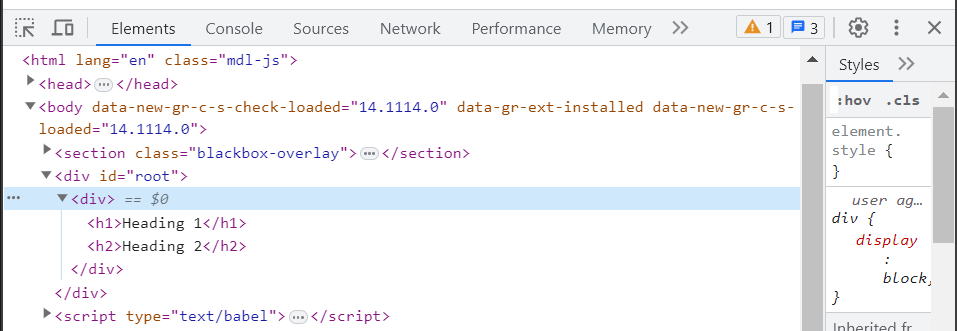
-Có thể thấy chúng ta khi chúng ta render ra website 2 Element thì nó sẽ báo lỗi ngay !

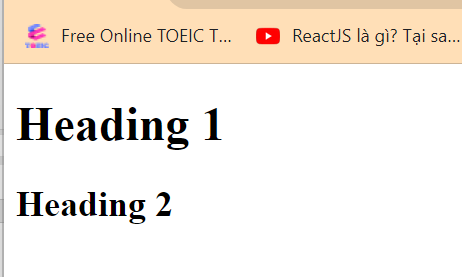
-Đây là code mà **Bable** sẽbiên dịch từ **JSX** sang **ReactJS** cho chúng ta ! Có thể thấy là bị lỗi cú pháp !

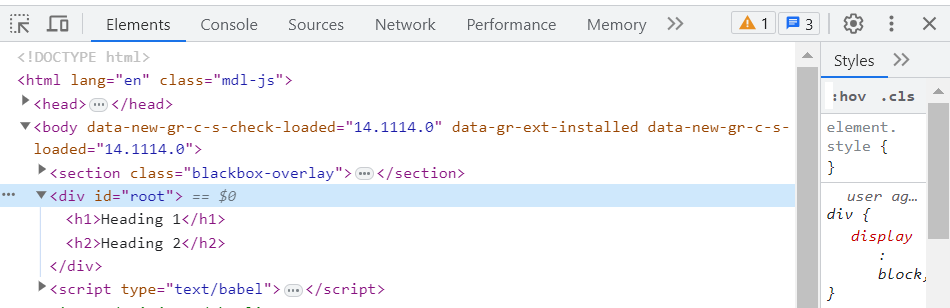
-Để có thể fix được lỗi thì chúng ta chỉ cần bộc các Element trong một thẻ khác thôi (vd ở đây là thẻ **<div></div>** !



-Nhưng cách này vô tình tạo ra một thẻ dư nằm bên ngoài và xuất hiện trên website !



-Để tối ưu hóa nhất thì chúng ta nên sử dụng cặp thẻ **<React.Fragment></React.Fragment>** ! Cặp thẻ này có nhiệm vụ là wrap(bọc) tất cả Element của chúng ta lại tránh gây lỗi và tuân thủ cú pháp của **Bable**



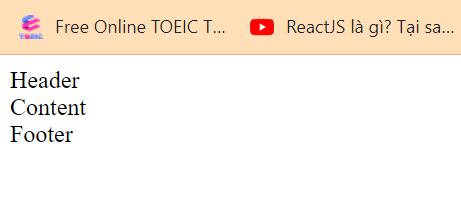
-Có thể thấy giờ đây không còn thẻ **<div>** nào dư bao bọc bên ngoài nữa !

1. **React Element type**

-Nếu như đã học **React.creatElement()** (bài 4) thì chúng ta sẽ nhớ đến cú pháp của nó React.creatElement(type ,props, children, n) ! Bài này chúng ta sẽ tìm hiểu sâu hơn về đối số đầu tiên **“type”**.

-Chúng ta đều biết ngay ở bài đầu tiên ! ReactJS sẽ chia code chúng ta thành các Component nhỏ hơn để dễ dàng quản lý, sữa đỗi bảo trì và clean code hơn !

-Như ở bài số 4 thì chúng ta biết được rằng **type** được truyền vào ở dạng **String** ! Nhưng ngoài ra thì **type** còn có thể truyền vào **function/class** nữa ! để giúp việc chia Component dễ dàng hơn !

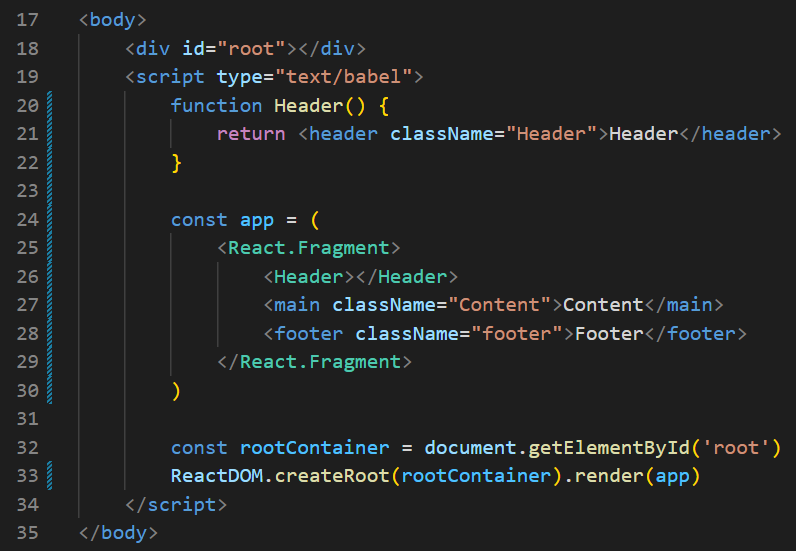
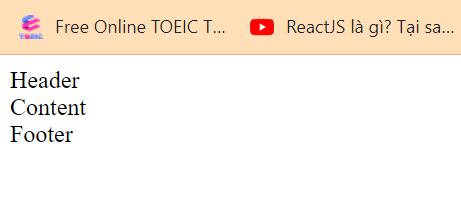
 

-Thông thường chúng ta hay thảy tất cả code vào một chỗ như trên ! Nhưng ReactJS thì sẽ không làm như vậy mà sẽ chia nhỏ thành các **function** hoặc **class** và bỏ vào các file.js khác nhau sau đó import vào một file.js ! (đoạn code trên sẽ làm ví dụ cho toàn bài nên lưu ý nhé)

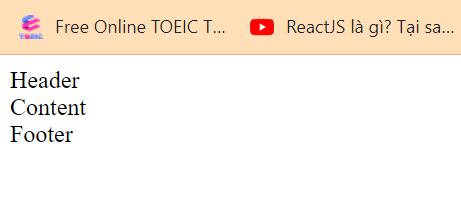
=> **React Element type:** String, Function/Class

* 1. **React Function Component**

-Kể từ khi có khái niệm Hooks ! thì người dùng sử dụng **Function** có thể làm được nhiều thứ hơn **Class** nên khá là ưa dùng Function (đến 99%)

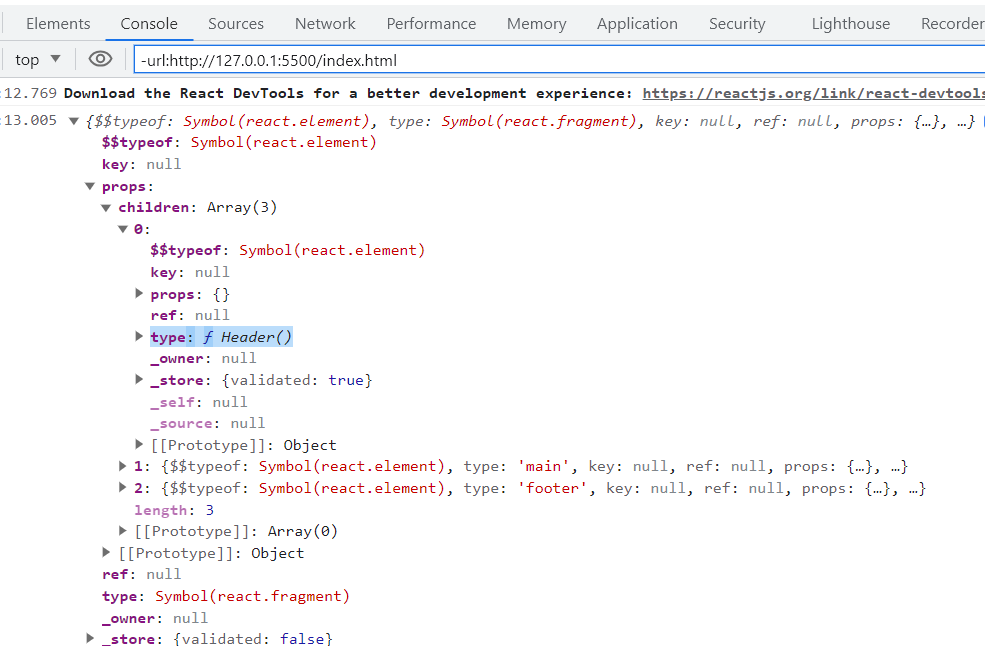
-Chúng ta sẽ tạo ra một Function chứa nội dung code **JSX** (lưu ý là tên hàm chúng ta viết hoa chữ cái đầu nhé) ! Sau đó chúng ta gọi hàm đó dưới dạng tag ! Và có thể thấy code của chúng ta đã được tách ra và vẫn hoạt động bình thường !

-Ngoài ra có thể viết tắt như này cũng được !



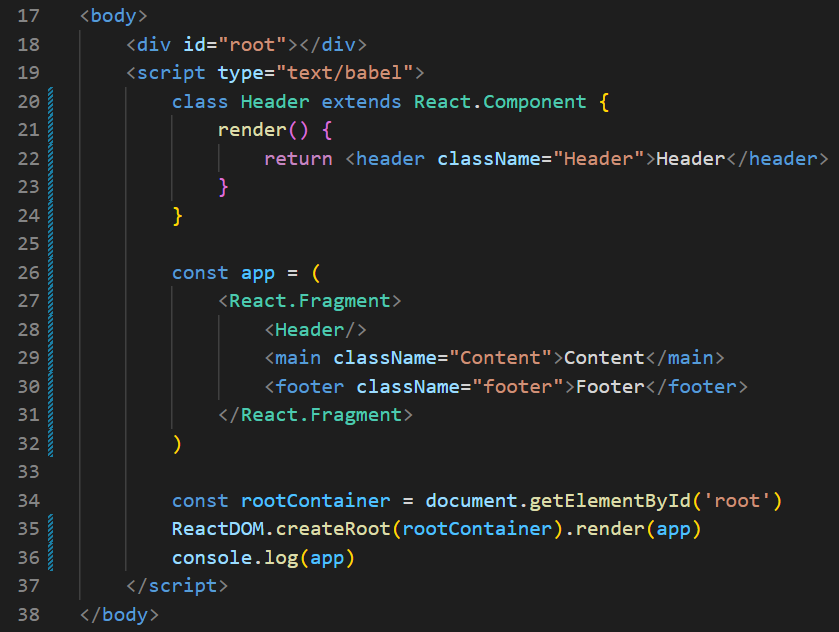
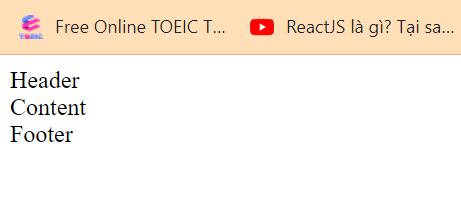
-Có thể thấy **Bable** biên dịch code **JSX** của chúng ta thành React element như thế nào !



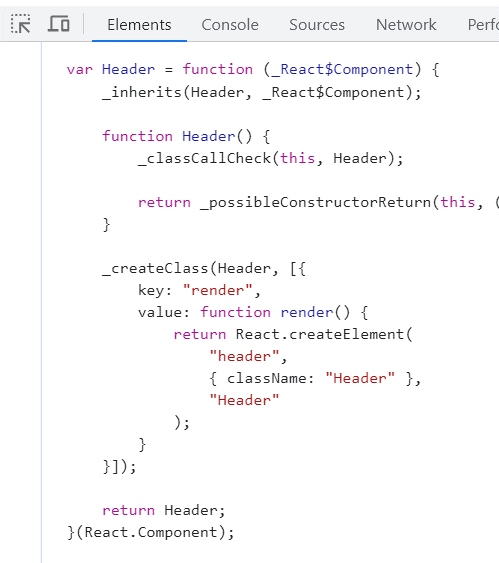
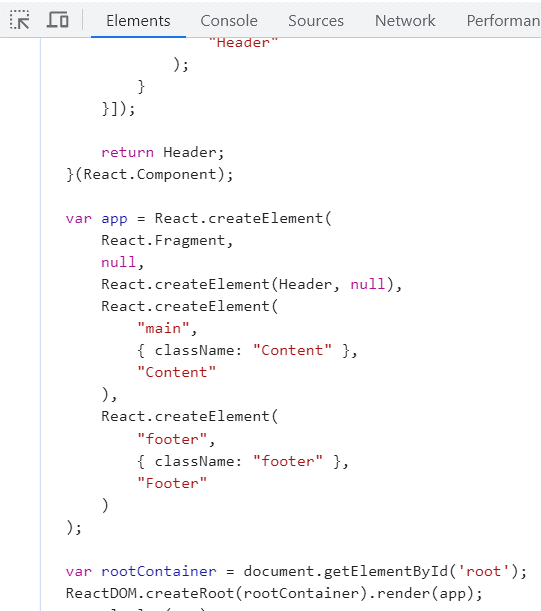
-**console.log(app)** ra để thấy ! type giờ đây có kiểu dữ liệu là **function**

* 1. **React Class Component**

**-**Tuy chia component bằng class không thông dụng như function nhưng đôi khi chúng ta cũng phải cần đến nó trong một vài trường hợp !

-Nắm được OOP thì 1 phát hiểu luôn ! Nếu sử dụng class thì chúng ta cần phải kế thừa **React.Component** sau đó sử dụng method **render()** để render code **JSX**

-Có thể thấy **Bable** biên dịch code **JSX** của chúng ta thành React element như thế nào !