|  |
| --- |
| **SPA - Single-Page Application**  **MPA - Multi-Page Application** |
| - ReactJS là 1 trong những thư viện tạo ra SPA  >> SỰ KHÁC BIỆT  - **SPA:**  Được cho là cách tiếp cận hiện đại hơn  Không yêu cầu tải lại trang trong quá trình sử dụng  SPA nhanh hơn khi sử dụngdo chỉ tải 1 trang và phần lớn tài nguyên trong 1 lần tải đầu tiên  SPA có phần Front-end riêng biệt  SPA không thân thiện với SEO  SPA trải nghiệm trên di động tốt hơn  SPA còn gọi là CSR : Client side rendering  - **MPA:**  Là cách tiếp cận cổ điển hơn  Tải lại trong quá trình sử dụng ( click vào đường link, chuyển sang,… )  MPA chậm hơn khi sử dụng  MPA Front-end và Back-end phụ thuộc nhau nhiều hơn được đặt trong cùng 1 dự án  MPA trải nghiệm trên di động tốt hơn  MPA còn gọi là SSR : Server side rendering |
| **Thêm ReacJS vào file HTML** |
| **1. Gifthub , NPMJS , UNPKG**  **Gifthub : Nơi lưu trữ mã nguồn**  <https://github.com/facebook/react> - Chứa source code của React  **NPMJS - Node Packed Manager : Nơi lưu trữ sản phẩm được xây dựng ra từ mã nguồn**  <https://www.npmjs.com/package/react> - Nơi post dự án để cộng đồng sử dụng  **UNPKG - Unpackage : Nơi tải các thư viện lưu trữ bên NPMJS dưới dạng CDN**  **cdn - Content Delivery Network** : Mạng phân phối nội dung, là mạng lưới gồm nhiều máy chủ đặt tại các vị trí địa lý khác nhau giúp truyền tải dữ liệu nhanh hơn từ mọi vị trí trên thế giới. Một nội dung từ 1 máy chủ sẽ được đồng bộ tới các máy chủ khác trong mạng lưới.  <https://unpkg.com/react> - dẫn đến link để nhúng react : <https://unpkg.com/react@17.0.2/index.js>  Trong đó 17.0.2 là phiên bản của React  Và index.js dùng để thay cho 2 phiên bản React ta đề cập ở phía dưới  'use strict';  if (process.env.NODE\_ENV === 'production') {  module.exports = require('./cjs/react.production.min.js');  } else {  module.exports = require('./cjs/react.development.js');  }  Ta thấy ReactJS có 2 phiên bản, production - Tối ưu cho sản phẩm  và development - Tối ưu cho developer ( có các báo lỗi thuận tiện cho Debug )  Để vào 2 file Js trên ta sẽ vào 2 link sau:  <https://unpkg.com/react@17.0.2/cjs/react.production.min.js> - File Production Min  <https://unpkg.com/react@17.0.2/cjs/react.development.js> - File Development  **1. ReactJS - reactjs.org : Tài liệu về ReactJs**  1.1 Link trực tiếp file script vào HTML  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>  <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>  <!-- Don't use this in production: -->  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>  - Sau khi file script được nhúng ta sẽ có một biến global : React là một Object chữa thuộc tính và method để làm việc với ReactJS  vd: console.log(React.version) // Trả ra version của ReactJs |
| **React.creatElement( )** |
| React.creatElement( ) dùng để tạo ra React Element  Syntax:  **React.creatElement( type , props , children, children n ….. );**  - **type**: loại element mà bạn muốn tạo  - **props**: Các thuộc tính muốn cài cho element đó  - **children** : là một Node con của element đó, có thể là text node hoặc loại khác  - **children n** : có thể truyền nhiều node con vào node cha  const h1React = React.createElement('h1',{              title: 'Hello',              className: 'heading'          },'Hello Guy');  **//** tạo ra một React Element: **<h1** *title***=“Hello”** *class***=”heading”> Hello Guy </h1>**  - Thực ra **children** cũng là một prop của element nên ta còn có thể viết như sau    const h1React = React.createElement('h1',{              title: 'Hello',              className: 'heading',              children: 'Hello Guy'    });  - Nếu có nhiều children thì nó sẽ được thể hiện dưới dạng một mảng  children: [ children1 , children2 , … ] |
| **React-Dom** |
| - React-Dom là một cầu nối giữa React và DOM , dùng React để tạo React Element sau đó dựa vào React-Dom để có thể render vào DOM  // Ngoài React-Dom còn có React-Native là cầu nối giữa React và Native , để làm việc trên iOS và Android  - Cài đặt React-DOM :  <script *src*="https://unpkg.com/react-dom@16.13.0/umd/react-dom.production.min.js"> </script>  - Link đường dẫn vào ngay sau đường dẫn của ReactJS, nếu link trước sẽ bị lỗi  - Dùng **console.log( ReactDOM.version )** để kiểm tra phiên bản của React-DOM , lưu ý ReactJS và React-DOM phải cùng phiên bản để hoạt động tối ưu nhất  - Với React-DOM , ta dùng React.DOM.render( ) để render  **Cú pháp:**  ReactDOM.render( *element* , *container* , *callback*)  - **element :** React Element muốn render  Nếu có nhiều element muốn truyền ta truyền vào dưới dạng Array  [ element1, element2, …]  - **container** : DOM muốn chứa React Element |
| **JSX - Javascript XML** |
| \*\* Để sử dụng JSX ta phải làm như sau:  1. Chèn link script sau:   <script *src*="https://unpkg.com/babel-standalone@6/babel.min.js">  </script>    2. Thêm định dạng type= “text/babel” để định dạng lại đoạn script:    <script *type*="text/babel">        // ...code here can usr JSX      </script>  - JSX không phải HTML , JSX cần có Javascript, Babel để dùng nó  - Babel : Là thư viện hỗ trợ chuyển đổi ES6 sang ES5 sử dụng được trên các trình duyệt cũ  Với JSX : Ta có thẻ gán trực tiếp 1 thẻ HTML element cho một biến  const h1 = <h1> JSX </h1>;  Khi sử dụng Babel nó sẽ tự động chuyển dòng lệnh kia về dạng sau:  var h1 = React.createElement('h1',{},'JSX');  Khi viết dạng JSX ta muốn chèn code Javascript vào ta phải bỏ nó trong dấu ngoặc { } ;  vd:    const reactCourse = 'JSX';    const h1 = <h1 className=“course-heading” > {reactCourse} </h1>  Sau đó ta lại dùng **ReactDOM.render( )** để xuất ra  -- LƯU Ý: Bất cứ chỗ nào viết bằng **React.creatElement( )** được thì viết bằng **JSX** được, do đó hãy chỉ viết mọi thứ bằng JSX để clean code ! |
| **JSX render Arrays** |
| Để render nhiều element ngang cấp ta có nhiều cách:  - **Cách 1 : Gói các element cần xuất vào một thẻ div trống:**  **🡺 Nhược điểm: tạo ra một thẻ div không cần thiết**  const jsx = (      <div>          <h1> Heading </h1>,          <span> Span </span>      </div>  )  - **Cách 2 : Dùng Array**  **🡺 nhược điểm mất thời gian viết “ , ” và không giống HTML**  const jsx = [          <h1> Heading </h1>,          <span> Span </span>  ]  **- Cách 3: Thay vì thẻ <div> ta dùng < React.Fragemt >**  **🡺 Nhìn giống HTML hơn**  const jsx = (      <*React.Fragment*>          <h1> Heading </h1>          <span> Span </span>      </*React.Fragment*>  )  Cách viết trên tương tự với việc :  const jsx = React.createElement( React.Fragment, null,          <h1> Heading </h1>,          <span> Span </span>  ) |
| **React Element Types & React Component** |
| 1. React element Types: string, function/class  2. React Component :  Xét ví dụ sau:  const jsx = (          <div *className*="wrapper">              <div *className*="header"  > Header  </div>              <div *className*="content" > Content </div>              <div *className*="footer"  > Footer  </div>          </div>  )  Ta tách phần <<div *className*="header"  > Header  </div> ra thành  “header-component” để có thể tái sử dụng nhiều lần:  **1. Với dạng Function Component:**    function Header() {          return (              <div *className*="header"  > Header  </div>          )      }    const jsx = (          <div *className*="wrapper">  **<Header></Header>**              <div *className*="content" > Content </div>              <div *className*="footer"  > Footer  </div>          </div>      )  \*\* Lưu ý tên Function phải viết Hoa chữ cái đầu : ***FunctionName…***  Cú pháp của JSX còn hỗ trợ ta thay vì viết : **{ Header( ) }**  Ta có thể viết: **< Header> </Header>**  Hoặc ngắn gọn hơn: **< Header/ >**  **2. Với dạng Class Component :**     class Content extends *React*.*Component* {          render() {              return   <div *className*="content" > Content </div>          }      }  const jsx = (          <div *className*="wrapper">  **<Header />**  **<Content />**              <div *className*="footer"  > Footer  </div>          </div>      ) |
| **Props** |
| **1. React elements : - Là những thẻ JSX nhìn giống HTML**  - Sử dụng props giống như Attributes của thẻ HTML  - 2 props **class**, **for** được chuyển thành **className** , **htmlFor**  - Phải tuân theo qui ước có sẵn  const jsx = (      <div>          <label htmlFor= “email”> Nhập email </ label >,          <input id=“email” className=“email”> </input>      </div>  )  // **htmlFor** là **prop** của **React element** **: <label />**  // **id** và **classname** là props của **React element** **: <input />**  **2. React Components: - Là những function/class trả ra JSX**  - Sử dụng props giống như đối số cho Component  - Tự do đặt tên cho props:  - Đặt tên theo camelCase , có thê bao gồm dấu gạch ngang \_    function Header(*props*){         return(            <h1 *className*={*props*.className}> {*props*.innerText} </h1>         )     }     <*Header* *className*="header" *innerText*="Tiêu đề"/>  Tương tự:    Header({class:"header", innerText:"Tiêu đề"})  Đối số của function Header là **props** là một **object** có các **KEY** là **className** và **innerText**  **\*\* Chú ý:**  - Prop “key” là prop đặc biệt  - Props cơ bản là đối số của Component  🡺 Props có thể là bất kì kiểu dữ liệu gì  - Sử dụng Destructuring |
| **DOM-event** |
| Trong JSX nếu dùng event ta phải chuyển về camelCase  vd: onclick 🡺 **onClick** = { **myFunction()** } |
| **Tạo Component linh hoạt** |
| const Form = {          Input() {              return <input/>          },          Checkbox(){              return <input *type*="checkbox" />          }      }      function App(){            const type = 'Checkbox'            const Component = Form[type]           return (              <div *id*="wrapper">                  <*Component* /> // <Form[type] /> sai cú pháp              </div>           )       } |
| **Hướng dẫn sử dụng thư viện React Webpack** |
| **webpack.js.org**  **Thư viện Webpack giúp module hoá ứng dụng Front-end của bạn**  **Giúp làm gọn file , loại bỏ những thứ không cần thiết giúp file nhẹ hơn** |
| **Tạo React App** |
|  |
| **npm** |
| **- Project scope :**  \*\*Thêm thư viện vào dự án  **npm install react react-dom --save** 🡺 cài vào dependencies  **npm i react react-dom --save** 🡺 cài vào dependencies  *(* ***i*** *là shorthand của* ***install*** *)*  **npm install react --save-dev react-dom --save** 🡺 cài vào devDependencies  **npm i react --save-dev react-dom --save** 🡺 cài vào devDependencies  \*\*Để xoá thư viện khỏi dự án  **npm uninstall react react-dom**  **- Global scope:**  \*\*Thêm thự viên vào máy tính  **npm i --global create-react-app**  **npm i --g create-react-app**  \*\*Để xoá thư viện khỏi máy tính  **npm uninstall -g create-react-app** |
| **npx :** |
| Khi cài node sẽ tự động cài cả npm và npx  npx giúp truy cập thư viện qua file .bin |
| **HOOKs - Là các method cung cấp sẵn bởi ReactJS** |
| - Dùng **HOOKS** để gắn vào các **FUNCTION COMPONENT** tăng sức mạnh lên cho nó  - Nếu không dùng HOOK , Function Component chỉ có khả năng render ra UI  - Khi dùng HOOKS, Function Component sẽ có thêm nhiều tính năng khác như call API, xử lý dữ liệu trước khi trả ra JSX  **import {**  **useState,**  **useEffect,**  **useLayoutEffect,**  **useRef,**  **useCallback,**  **useMemo,**  **useReducer,**  **useContext,**  **useImperativeHandle,**  **useDebugValue,**  **} from ‘react’**  -- Khi import thư viện **react** vào thì ta đã có thể sử dụng được các Built-in Hook này vì chúng là các method của Object ‘React’. vd: **React.useState()** , **React.useEffect()** ,….. |
| **HOOK : useState** |
| **useState** : **Sử dụng trạng thái của dữ liệu**  🡺 Giúp thể hiện trạng thái của dữ liệu ra giao diện người dùng , giao diện sẽ tự động cập nhật khi dữ liệu được thay đổi.  **import { useState } form ‘react’**    **function MyComponent() {**  **const [state , setState] = useState( initState );**  **…...**  **}**  Với useState nhận giá trị đầu vào là **initState -** *trạng thái khởi tạo* và sẽ return lại một mảng gồm 2 phần tử là **state**-*trạng thái hiện tại* và **setState**-*thay đổi giá trị state*  Trong đó **setState** là tên một function khi bạn truyền cho nó một đối số nó sẽ làm thay đổi giá trị của **state**  **[ setState là thay thế State bằng một giá trị mới và render lại Function Component ]**  **vd: xét React Component sau:**   function App() {    const [ count , setCount ] = useState(0)    console.log('setCount:' , setCount)    return (      <>        <p> {count}</p>        <button *onClick*={() => setCount( count + 1 )}>          Tăng số        </button>      </>    );  }  🡺 Ban đầu count sẽ bằng giá trị khởi tạo = 0 ; giá trị khởi tạo chỉ sử dụng 1 lần duy nhất ban đầu.  **[ Initial state chỉ được dùng cho lần đầu ]**  🡺 Khi Click vào Button ta sẽ gọi hàm setCount( 0 + 1 ) nên lúc này count = 1 , sau đó React sẽ gọi hàm App lại lần nữa để render ra giá trị count mới.  **[ Component được re-render sau khi setState ]**  **🡺** Trong một Function Component có thể sử dụng một hook được nhiều lần:  const [ count , setCount ] = useState(0);  const [ age , setAge ] = useState(18);  const [ name , setName ] = useState(‘Binh’);  **🡺 setStage( *callback* )**  - Nếu truyền một callBack vào **setStage** thì nó sẽ lấy giá trị trả lại của callBack để gán vào **state và render lại React Component**  *onClick*={() => setCount( preState => preState + 1) }  **🡺 Gọi hàm setStage( ) nhiều lần:**  - Nếu gọi hàm setStage nhiều lần thì nó vẫn chỉ thực hiện render một lần ở vị trí gọi hàm setStage cuối cùng  **🡺 Thay initStage bằng hàm callBack**  **- useState( callback )** nó sẽ dùng giá trị trả về của hàm callBack để làm initState và hàm callBack này chỉ được sử dụng một lần  **\*\* Khi cần một hàm tính toán và dùng kết quả của nó làm initState thì phải khởi tạo nó ngay trong useState để tăng hiệu năng tránh sử dụng bên ngoài để mỗi lần render phải gọi hàm đó lại một lần nữa.** |
| **Khái niệm : Mounted và Unmounted** |
| - **Mounted** : Chỉ khoảng khắc ngay khi React Component được gọi  - **Unmounted** : Chỉ khoảng khắc ngay khi React Component được tắt đi |
| **HOOK: useEffect** |
| **useEffect**: **Dùng khi muốn sử dụng các** **Side Effect**  -- Thuật ngữ Side Effect nói tới việc một chương trình phần mềm khi có tác động xảy ra dẫn đến việc thay đổi dữ liệu của phần mềm đó.  **🡺 useEffect sẽ giúp chúng ta thực hiện các tác vụ sau:**   * **Update DOM** * **Call API** * **Listen DOM events** * **Cleanup ( clear Timer , remove listener )**   **Cú Pháp: useEffect( callback , [] )** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |