|  |
| --- |
| **ReactJS** |
| **1. Khái niệm** |
| - ReactJS là một thư viện JavaScript có tính hiệu quả và linh hoạt để **xây dựng UI**  - React chia các phần UI thành các thành phần nhỏ - gọi là **React Component**  - React được dùng để xây dụng các ứng dụng **SPA ( Single Page Application )**  **Các từ khoá quan trọng:**   * **Virtual DOM:**   **- DOM** : Document Object Model - chỉ các element trong HTML  - **Virtial DOM** là DOM ảo.  Khi làm việc vơi DOM , khi một node thay đổi sẽ kéo theo tất cả các node khác phải  thay đổi theo sau đó mới được Render ra ngoài ( vd phải load lại trang )  - Khi sử dụng ReactJS thì việc thay đổi các DOM này diễn ra liên tục nên nếu sử dụng Dom quá trình Render sẽ mất nhiều thời gian  - Với ReactJS nó sẽ sử dụng Virtual DOM để so sánh quá trình **trước** và **sau** khi render một chuỗi JSX và chỉ cập nhập Node nào được update mà thôi   * **JSX:**   - JSX - Javascript XML , cho phép viết các đoạn mã HTML trong file JS một cách dễ dàng  - Để làm được điểu này React sử dụng một thư viện là Babel để chuyển các file JSX về ReactJS và thư viện React để chuyển về Javascript thuần   * **Components:**   - Trong dự án React các phần của UI sẽ được chia nhỏ thành các phần khác nhau và chỉ được Render ra khi có yêu cầu  - Các React Component có nhiệm vụ sẽ Render ra các đoạn mã HTML  - Có 2 loại React Component là: **Class Component** và **Function Component**   * **Props và State:**   **- Props:** là các tham số được truyền vào React Component  vd: <**MyComponent** title=“Học ReactJS” />  // React Component MyComponent có props là { title: ‘ Học ReactJS’ }  - **State**: là một Object lưu giữ giá trị thuộc tính nằm ngay trong Components và chỉ tồn tại trong phạm vi của Component đó. Khi thay đổi một giá trị nào đó của State ngay lập tức Component đó sẽ được render lại  class Car extends React.Component {    constructor(props) {      super(props);      this.state = {brand: "Ford"};    }    render() {      return (        <div>          <h1>My Car</h1>        </div>      );    }  }   * **React Lifecycle:**   - Là một vòng đời của một React Component : từ khi nó được tạo rồi render ra cho đến khi tháo nó ra  - ReactJS cho phép ta can thiệp vào từng giai đoạn trong quá trình này bằng các **hàm hỗ trợ lifecycle đối với Class Component** và **hook đối với Function Component** |
| **2. Cài đặt môi trường làm việc** |
| - Dùng thư viện **creat-react-app** để tạo khung sườn cho dự án:  **npx create-react-app my-app**  *// Dùng npx để sau khi tạo xong thư viện này sẽ được xoá khỏi hệ thống*  Khi chạy xong ta sẽ có các thư mục sau được tạo ra:  **- src :** Chứa những đoạn mã chúng ta sẽ viết  **- public :** Chứa các file tĩnh như ảnh, css, js  **- node\_modules:** Các module được sử dụng trong dự án  **- package.json:** File quản lý dự án , thiết lập kịch bản chạy cho dự án  **- package-lock.json:** File lưu các tên module trong dự án  *-* Khởi chạy dự án  **npm start**  *-* Xuất dự án  **npm run build** |
| **3. JSX** |
| - JSX là một **template languages** nhưng cho phép sử dụng các tính năng của Javascript  - JSX giúp tạo ra các React Element thay cho cú pháp: React.creatElement( )  **Chỉ định attributes với JSX:**  - Có thể chỉ định một Attributes trong JSX với cú pháp như HTML :  const element = <div **tabIndex=”0”** > </div>  - Hoặc chỉ định bằng biểu thức JavaScript đặt trong dấu ngoặc { } :  const **myIndex** = 0;  const element = <div **tabIndex={myIndex}**> </div>;  **JSX Object:**  - Ngoài các viết JSX dưới dạng HTML ta còn có thể viết dạng sau:  const element = {  type: “p”,  props: {  className: “welcome”,  children: “Chào mừng các bạn”  }  };  - Nó tương tự với cách viết sau:  const element = <p className=”welcome”> Chào mừng các bạn </p>  🡺 JSX Object dễ hơn cho việc debug nhưng nếu không cần thiết thì không nên viết dưới dạng HTML để dễ quan sát |
| **4. Component** |
| - Component giúp phân chia các UI ( giao diện người dùng) thành các phần nhỏ để dễ dàng quản lý và tái sử dụng.  - Trong mỗi React App đều có thể nhận rất nhiều component , trong mỗi component lại nhận các Props & State làm đối số rồi render ra mỗi khi Component đó được gọi.  **- Component là một FUNCTION hoặc METHOD khi được gọi sẽ trả ra một chuỗi JSX**  const App = () => <h1>Hello World </h1> |
| **4.1 Function Component** |
| - Đây là cách viết phổ biến và được sử dụng nhiều nhất trong các dự án, bởi tính nhanh gọn và không quá phức tạp như Class Component.  - Vd: Tạo một file MyComponent.js để xuất một Function Component như sau:  const Welcome = function(*props*) {      return (        <div>          <h1>Welcome ! I am a functional component </h1>        </div>      )    }    export default Welcome; |
| **4.2 Class Component** |
| Cách viết này là cách viết đầy đủ của một component, khi bạn viết một Class Component.  Với Class Component sẽ sử dụng được các chức năng của component như state, lifecycle, events , ….  Để tạo được Class Component ta phải sử dụng React.Component đó bắt buộc phải  **import React from ‘react’** để sử dụng được React.Component:  import React from 'react'  class MyComponent extends *React*.Component {      render(){          return (              <div>                  <h1> Class Component </h1>              </div>          )      }  } |
| **5. Props** |
| **- Props là một Object được truyền vào Component , các Component sẽ nhận vào props và return lại React Element có chứ thông tin nhận được từ Props**  - Props cho phép giao tiếp giữa các component với nhau  🡺 Component cha truyền props cho Component con , Component com xử lý props và return lại cho Component cha  const App = () => <*MyComponent* *title*="Hello" *name*="component" />  Lúc này component MyComponent giá trị của props sẽ là một Object chứa các giá trị truyền vào:  {      title: "Hello",      name:"component",  }   * **Props trong Function Component:**   - Nhận props trong Function Component bằng cách **chỉ định tham số trong function**  const MyComponent  = *props* =>  (      <h1> {*props*.title}   {*props*.name} </h1>  )   * **Props trong Class Component:**   - Nhận props trong Class Component bằng cách sử dụng **this.props**  class MyComponent extends *React*.Component {      render(){          return (              <h1>{*this*.props.title} {*this*.props.name} </h1>          )      }  }   * **Thư viện PropTypes:**   - **Vấn đề:** **1 + 2 = 3** còn **“1” + “2” = 12 ,** do truyền nhầm dữ liệu từ Number thành String  - **Giải quyết:** Thư viện Proptypes cho phép chúng ta chỉ định kiểu dữ liệu mong muốn cho props.  - **Cài đặt:** **npm i prop-types**  - **Import để sử dụng:** **import PropsTypes from ‘prop-types’**  - Cú pháp: **MyComponent.propTypes = { key : kiểu dữ liệu }**  Kiểu dữ liệu: Proptypes.number , Proptupes.array, ……. 🡺 đọc Doc để tìm |
| **6. State** |
| - State cho phép chúng ta lưu dữ liệu bên trong một Component và mỗi khi state thay đổi giá trị thì Component đó sẽ tự động được re-render lại.  - State là một Object chứa thông tin về Component và chỉ tồn tại bên trong Component chứa nó:  - Với Class Component bạn có thể sử dụng trực tiếp bằng **this.state** và **this.setState**  - Với Function Component bạn phải sử dụng **React HOOKS** bằng cách import từ thư viện react như sau:  import { useState } from 'react';   * **state trong Class Component**   - Để khởi tạo một state, ta gán giá trị cho biến **this.state** trong hàm **contructor( )**  class MyComponent extends *React*.Component {    constructor(*props*) {  *super*(*props*);  *this*.state = { course: "ReactJs"};    }    render(){      return (             <h1> {*this*.state.course}</h1>      )    }  }  - Để thay đổi giá trị cửa giá trị trong state ta dùng **this.setState**  class MyComponent extends *React*.Component {      constructor(*props*) {  *super*(*props*);  *this*.state = { course: "Học ReactJS" };      }      render(){        return (          <>               <h1> {*this*.state.course } </h1>               <button *onClick*={ () => {  *this*.setState( {course: "tại F8"})                   }               }> Ấn vào đây </button>          </>        )      }  }      **🡺 sau khi ấn 🡺**  - **this.setState( )** là một hàm nhận đối số là một **Object** chứa giá trị cần thay đổi  hoặc nhận đối số là một **function(** {preValue} **🡺 { return** {newValue} **} );**  *this*.setState( ({*course*}) => ({course : "tại F8"}) )  // Các Object đặt trong dấu ngoặc ( ) để không bị lỗi |
| **7. Các Component API trong ReactJS** |
| 1. **Set State API :**   - Hàm **setState()** cho phép cập nhật giá trị của state , tham số truyền vào của API này là gía trị của state bạn muốn update:  *this*.setSate({      website: 'freetuts.net'   })  Hoặc một function callback gồm các tham số lần lượt là state trước và props của component:  *this*.setState((*prevState*, *props*) => {      return newState;  })  🡺 ***Nên nhớ trong Class Component cả prevState và newState đều phải là dạng Object***   1. **Force Update :**   - ***this*.forceUpdate()** : Đây là một hàm cho phép re-render Component mà không cần bất cứ sự thay đổi nào của state  - **VD:** Tạo một nút mỗi lần nhấn sẽ re-render lại Component:  render(){      return (          <button *onClick*={() => *this*.forceUpdate()} > Click </button>      )  }   1. **Fim DOM Node :**   - Hàm này cho phép làm việc với DOM trong React , tuy nhiên nó dễ gây ra lỗi do ảnh hưởng đến các Virtual DOM nên không được khuyến khích sử dụng  - Để sử dụng bạn phải import từ thư viện react-dom:  import ReactDOM from 'react-dom';  - Để sử dụng hàm **findDOMNode()** ta có cú pháp sau:  ReactDOM.findDOMNode( document.getElementBy... )  VD: Component tạo ra một nút khi ấn sẽ thay đổi màu của thẻ h1:  class App extends *React*.Component {      changeColor() {        var title = document.getElementById("title");        ReactDOM.findDOMNode(title).style.color = "red";      }      render() {        return (          <div>            <h1 *id*="title">Tiêu đề</h1>            <button *onClick*={() => *this*.changeColor()}>  Change Color  </button>          </div>        );      }    }   1. **Bind function:**   - Khắc phục lỗi sai liên quan đến từ khoá ***this***  class MyComponent extends *React*.Component {      constructor() {  *super*();  *this*.state = {            title: "Freetuts.net"          };        }        changeTitle() {  *this*.setState({            title: "Freetuts.net New"          });        }          render() {          return (            <div>              <h1 *id*="title">{*this*.state.title}</h1>              <button *onClick*={*this*.changeTitle}>Change Title</button>            </div>          );        }  }  - Khi gọi hàm *this*.changeTitle sẽ gây ra lỗi : Không tồn tại biến this trong hàm changeTitle()  - Để khắc phục lỗi này ta có 2 cách sau:   * **Dùng .bind( ) trong ES6:**    <button *onClick*={*this*.changeTitle.bind(*this*)}>         Change Title   </button>   * **Dùng arrow function:**   <button *onClick*={() => *this*.changeTitle} }>         Change Title  </button> |
| **8. Component Life Cycle :** |
| - Vòng đời của Component: Gồm 4 giai đoạn chính:  **Initialization 🡺 Mounting 🡺 Updating 🡺 Unmounting**  **React cung cấp cho ta một số life cycle methods trong các giai đoạn**   * **Initialization :** *Khởi tạo các giá trị*   - Lúc này Component bắt đầu khởi tạo state và props bên trong phương thức constructor()   * **Mounting :** *component được hiển thị ra DOM*   - Nó thực hiện chuyển virtual Dom trong React thành DOM thật và hiển thị trên trình duyệt. Tại đây Component được render lần đầu tiên và ta có 2 method để tham gia vào giai đoạn này:   1. **componentWillMount( )**   - Được khởi chạy trước khi component được mount ( được render)  🡺 Lưu ý: thời gian mount là rất ngắn do đó không nên làm các tác vụ liên quan đến thay đổi props, state hay call API trong giai đoạn này  componentWillMount() {      console.log('Component will mount!')   }   1. **componentDidMount( )**   - Được khởi chạy khi component đã được render xong ( render thành công)  - Trong method này bạn có thẻ gọi API , thay đổi state hoặc props  componentDidMount() {      console.log('Component did mount!')  *this*.callAPI(); // có thể fetch API lấy dữ liệu nạp vào state   }   * **Updating :** *component được re-render*   - Trong giai đoạn này các giá trị của state và props được cập nhật để đáp ứng với các sự kiện của người dùng và dẫn đến việc re-render lại Component  - Ở giai đoạn này ta có 3 methods để tham gia vào   1. **shouldComponentUpdate( )**   - Là một hàm mặc định trả về **TRUE** , quyết định việc component có được re-render hay không  - Hàm này nhận 2 tham số truyền vào là **nextState** và **nextProps** là các Object chứa các giá trị mà state và props sẽ được cập nhật  **vd: *Phương thức kiểm tra giá trị của List cũ và List mới nếu khác nhau mới cho re-render lại***  constructor(props){  *super*(props);  *this*.state = {         list:[]   };  shouldComponentUpdate(nextProps, nextState){      return *this*.state.list !== nextState.list  }   1. **componentWillUpdate( )**   - Phương thức này được gọi trước khi re-render , nhận vào 2 tham số là nextState và nextProps  componentWillUpdate(nextProps, nextState) {      console.log('Component will update!');  }   1. **componentDidUpdate( )**   - Phương thức này được gọi sau khi component được re-render xong và nhận 2 đối số là prevState và prevProps  componentDidUpdate(prevProps, prevState) {      console.log('Component did update!')   }   * **Unmount:** *Component được tháo khỏi DOM*   - Đây là bước cuối cùng và kết thúc một vòng đời của Component và quá trình này chỉ có 1 phương thức duy nhất: **componentWillUnmount( )**  componentWillUnmount() {      console.log('component will unmount')    } |
| **9. Handling Events - Xử lý sự kiện** |
| - Xử lý sự kiện trong React rất giống với xử lý sự kiện trên các phần tử DOM nhưng có một số khác biệt về cú pháp :   * Sự kiện trong React được đặt tên bằng camelCase: onClick , onChange * với JSX bạn truyền một hàm để bắt sự kiện thay vì một chuỗi như HTML  |  |  | | --- | --- | | **HTML** | **JSX** | | <button onclick="changeName()">    Change Name  </button> | <button *onClick*={changeName}>    Change Name  </button> |  * Không thể dùng return false để chặn sự kiện mặc định trong React mà phải sử dụng preventDefault()  |  |  | | --- | --- | | **HTML** | **JSX** | | <form onsubmit="() => false"> | <form *onSubmit*={(*e*) => {  e.preventDefault()  }  }> |   - Ở đây e là một Object chứa tất cả các event   * **Lưu ý với *this* trong xử lý event**   **vd: Trong class Component ta có một method sau:**  changeText(e) {  *this*.setState({        textareaChange: e.target.value      });  }  **Để truyền hàm này vào xử lý sự kiện ta có 3 cách sau:**   1. **Sử dụng arrow function**   <button *onClick*={() => *this*.changeText()}></button>   1. **Sử dụng bind**   <button *onClick*={*this*.changeText.bind(*this*)}></button>   1. **Sử dụng this trong constructor( )**    constructor(*props*) {        ...  *this*.changeText = *this*.mouseOver.bind(*this*)      }    Sau đó có thể sử dụng được method như sau:  <button *onClick*={*this*.changeText} button> |
| **10. Render có điều kiện trong ReactJS** |
| Tìm hiểu về việc kiểm soát việc render trong ReactJS   * **Gán element vào biến một cách có điều kiện**   export default class MyComponent extends *React*.Component {    render() {      const isLogin = true;      const notification = (isLogin) ? <h3>Đăng nhập thành công</h3> : <h3>Vui lòng đăng nhập</h3>      return <div>{notification}</div>;    }  }  🡺 Sử dụng toán tử **Comditional operator - ? :**   * **Biểu thức điều kiện trong JSX**   export default class MyComponent extends *React*.Component {    render() {      const isLogin = true;      return (        <div>        {isLogin ? (<h3>Freetuts.net</h3>) : (<h3>Vui lòng đăng nhập</h3>)}      </div>      )    }  }   * **Ngăn chặn component render bằng null**   export default class MyComponent extends *React*.Component {    render() {      const isLogin = true;      return (        <div>        {isLogin ? (<h3>Freetuts.net</h3>) : **null** }      </div>      )    }  } |
| **11. List và Keys trong React** |
| Khi render nhiều thẻ <li> trong ReactJS , thì nó sẽ luôn yêu cầu chúng ta gắn key cho các thẻ li đó để định danh các thẻ đó  Vd: Khi render một danh sách nhưng chưa gắn key , ta sẽ nhận được thông báo sau  function ListComponent(*props*) {    const myList = ["php", "javascript", "python", "C++"];    const listItems = myList.map((*item*) =>      <li>{*item*}</li>    );      return (      <ul>{listItems}</ul>    );  }  **Cảnh báo:**    **Cách khắc phục:**  - Có thể tạo các id gắn vào các giá trị trong myList và dùng nó làm giá trị cho Key như sau:  function ListComponent(*props*) {    const myList = [      {        id : 'p',        name : 'php'      },      {        id : 'j',        name : 'javascript'      },      {        id : 'py',        name : 'python'      },      {        id : 'c',        name : 'C++'      },    ]    //Thêm thuộc tính key vào trong thẻ jsx    const listItems = myList.map((*item*) =>      <li *key* = {*item*.id}>{*item*.name}</li>    );      return (      <ul>{listItems}</ul>    );  }  **\*\* Lưu ý:**  - Keys là **giá trị duy nhất**, nếu có 2 key cùng giá trị sẽ nhận cảnh báo  - KHÔNG nên dùng index làm giá trị cho Key như sau:  const listItems = myList.map((*item*, *index*) =>    <li *key* = {*index*}>{*item*.name}</li>  );  **🡺 Điều này sẽ ảnh hưởng đến việc xử lý danh sách sau này như sắp xếp lại, lọc, …** |
| **12. Kỹ thuật Lifting State Up** |
| **- Kỹ thuật này giúp bạn khi một Component Con re-render sẽ kéo theo Component-Cha re-render luôn**   * **Lifting State Up** - Truyền dữ liệu từ con lên cha:   Component Cha truyền một function vào props của Component Con  function ComponentCha(*props*) {    //Hàm này sẽ được gọi khi nhận được dữ liệu    const receiveData = function (*data*) {      console.log('Data nhận được', *data* )    }    //Gọi compoennt con và truyền vào một props    //có giá trị là một hàm    return (      <*ComponentCon* *onReceiveData* = {receiveData}/>    )  }  Component Con khi nhập được funtion từ props lại truyền một đối số vào function đó  function ComponentCon (*props*) {    return(      <div>          <button *onClick*={() => {  *props*.onReceiveData('data gửi đi')          }}>Gửi lại cho componentCha</button>      </div>    )  }  🡺 Đây là cách gửi dữ liệu qua lại giữa các Component  **🡺 Làm được các dạng như nhập số tiền VND vào ô input sẽ tự động chuyển qua USD** |
| **13. Refs** |
| - **React Refs** hỗ trợ chúng ta tham chiếu trực tiếp đến một element trong DOM, đọc dữ liệu và chỉnh sửa element đó như JavaScript DOM mà không cần thông qua props hay state.  - **Cú pháp:**  Trong class component: *this*.myRef = *React*.createRef();();  Trong function component:  *let* myRef = *React*.createRef();  Sau đó gắn nó vào element muốn sử dụng:  *ref*={*this*.myRef} // trong Class Component  *ref*={myRef} // trong Function Component  - **Cách sử dụng:** **Refs** được chỉ định khởi tạo trong **constructor( )** , và sau đó được gắn vào trong **một element trong hàm render( )**  class MyComponent extends *React*.Component {    constructor(*props*) {  *super*(*props*);  *let* myRef = *React*.createRef(); //Cú pháp khởi tạo Ref    }      render() {      return (        <input  *name*="email"  *onChange*={*this*.onChange}  *ref*={*this*.myRef} //gắn vào element muốn xài  *type*="text"        />      );    }  }  🡺 Lúc này thông tin trong thẻ input có thể được sửa đổi thông qua **myRef:**  **vd:** tạo một nút button khi ấn vào sẽ focus vào thẻ input trên  <button *onClick*={() => *this*.myRef.current.focus()}>     Focus Input  </button>  Thuộc tính **current** trong **myRefs** chứa giá trị element được tham chiếu khi element đó được render ra. |
| **14. Contexts** |
| **- Context cung cấp cho chúng ta cách thực hiện chia sẻ dữ liệu tới các component trong cây mà không cần truyền qua props theo từng cập bậc**  - Context đưa dữ liệu lên Local State và phân phối nó tới chỗ cần xài  **Context API :**   1. **React.createContext :**   - Cú pháp khởi tạo một context có tên MyContext:  const MyContext = *React*.createContext()   1. **MyContext.Provider :**   - Giúp context truyền dữ liệu vào element cần sử dụng  <*MyContext.Provider* *value*="Giá trị sẽ được truyền vào">        <*ComponentNaoDo* />  </*MyContext.Provider*>   1. **MyContext.Consumer:**   Một React Component được khởi chạy mỗi lần giá trị của context thay đổi  - Dùng khi nếu Component nhận value và làm thay đổi value đó  <*MyContext.Consumer*>    {*value* => /\* render something based on the context value \*/}  </*MyContext.Consumer*>   1. **MyClass.contextType:**   Thuộc tính để lấy giá trị của context  ComponentNaoDo.contextType = MyContext;  Lúc này trong Class Component : MyClass sẽ nhận được giá trị của context thông qua **this.context** |
| **15. Higher-Order Components ( HOC )** |
| **HOC là một Function nhận vào một Component và trả về một Component**  **HOC là một phương pháp phát triển component.**  **VD**: Tạo một HOC giúp làm mờ ảnh mỗi khi được Hover vào;  - Tạo một function **withHoverOpacity** nhận đối số là một Component như sau:  const withHoverOpacity = (*ImageComponent*) => {    return class extends *React*.Component {      constructor(*props*) {  *super*(*props*);  *this*.state = {          opacity: 1,        };        //bind this  *this*.onMouseLeave = *this*.onMouseLeave.bind(*this*);  *this*.onMouseEnter = *this*.onMouseEnter.bind(*this*);      }      //Được gọi khi chuột được di vào      onMouseEnter() {  *this*.setState({          opacity: 0.5,        });      }      //Được gọi khi chuột được rời đi      onMouseLeave() {  *this*.setState({          opacity: 1,        });      }      render() {        return (          <div  *style*={{ opacity: *this*.state.opacity }}  *onMouseEnter*={*this*.onMouseEnter}  *onMouseLeave*={*this*.onMouseLeave}          >            <*ImageComponent* />          </div>        );      }    };  };  - Sau đó ta có một Component Image1 như sau:  const Image1 = (*props*) => {    return <img *src*="https://freetuts.net/public/logo/logo.png" *alt*="freetuts" />;  };  - Khi cần sử dụng HOC trên ta chỉ việc truyền Component vào:  const ImageWithHoverOpacity1 = withHoverOpacity(Image1);  🡺 Khi đó ta có thể tạo ra một component xuất ra hình ảnh có tính năng mờ đi khi hover vào |
| **16. HOOKs trong ReactJS** |
| **HOOK** cho phép sử dụng **State** và **life cycle** bên trong **Function Component**  - Có tất cả 10 Hook trong React từ 16.8 nhưng sau đây là 4 HOOK hay sử dụng:   * **useState( ) :**   - Cho phép làm việc với State trong function Component  **- Cú pháp:**  const [state, setState] = useState(giaTriBanDauCuaState);   * **useEffect( ) :**   - Là một hàm nắm bắt tất cả sự thay đổi của code. Nó được khởi chạy khi giá trị của một biến nào đó được thay đổi, hay một component nào đó được render ra.  🡺 useEffect( ) hoàn toàn thay thế được các method về life cycle trong Class Component  **- Cú pháp:**  useEffect(functionDuocKhoiChay, arrayChuaCacGiaTriThayDoi)   * **useContext( ) :**   - Cho phép nhận về giá trị của context mỗi khi nó thay đổi  **- Cú pháp:**  const giaTriCuaContext = useContext(TenContext);   * **useReducer( ) :**   - Được sử dụng để xứ lý các State phức tạp và chia sẻ state giữa các component  **- Cú pháp:**  const [state, dispatch] = useReducer(reducer, initialArg, init); |
| **16.1 useEffect( )** |
| - Có thể hiểu đơn giản : u**seEffect( )** là sự hợp nhất của 3 phương thức **componentDidMount** , **componentDidUpdate**, và **componentWillUnMount**  **- Cú pháp:**  useEffect(effectFunction, arrayDependencies)  **Trong đó:**   * **effectFunction** : Hàm sẽ được gọi khi giá trị trong mảng thay đổi * **arrayDependencies**: Mảng giá trị phụ thuộc * **useEffect như componentDidMount - *Hàm được gọi sau khi component render xong***   - Chúng ta có thể bắt sự kiện **componentDidMount** trong một Function Component bằng cách sử dụng **useEffect** và chỉ định **arrayDependencies = [ ]**  useEffect(effectFunction, [])  - Nó thường được sử dụng để gọi API   * **useEffect như componentDidUpdate - *Hàm được gọi sau khi component re-render xong***   - Chúng ta có thể bắt sự kiện **componentDidUpdate** trong một Function Component bằng cách sử dụng **useEffect** và chỉ định **arrayDependencies = null**  useEffect(effectFunction)   * **useEffect như componentWillUnMount - Hàm được gọi trước khi component unmount**   - Để sử dụng như **componentWillUnmount**, ta cần **return về 1 function** trong effectFuntion, funtion được return đó sẽ **đóng vai trò như componentWillUnMount**  useEffect(() => effectFunction) |
| **16.2 useContext( )** |
| - Để sử dụng được context trong Function Component ta sử dụng **useContext( )** và truyền vào **một tham số là Context Object** ( đã được tạo bằng React.createContext )  const MyContext = *React*.createContext({ foo: 'bar' });  const MyClass = () => {  //Lấy giá trị của context  const value = useContext(MyContext);  return (  ...  );  } |
| **17. Rexdux** |
|  |
| **18. React Rounter** |
| - **React router** là một thư viện hỗ trợ xây dựng SPA ( single page application)  - Sử dụng URL làm điều kiện xem xét rằng Component nào sẽ được render  - **Cài đặt: npm i react-router-dom**  **- Sử dụng:**  **import { BrowserRouter , Route , Switch, Link } from ‘react-router-dom’**     |  |  | | --- | --- | | **BrowserRouter** | **Bọc các component cần được Render lại** | | **Switch** | **Dùng bọc các Router lại** | | **Route** | **Chỉ định Render component theo Path được chỉ định** | | **Link** | **Chuyển URL đến Path chỉ định** |   **- Ví dụ cụ thể:**  Trong file index.js dưới đây ta muốn sử dụng react-router-dom để chỉ định sẽ render những component nào dựa vào URL như sau:  import ReactDOM from 'react'  import { BrowserRouter} from 'react-router-dom';  import { Route, Switch } from 'react-router-dom';    ​import Home from './components/Home';  import About from './components/About';  import Shop from './components/Shop'  import Error from './components/Error'  *ReactDOM*.render(    <*BrowserRouter*>       <*Switch*>              <*Route* *path*="/" *component*={Home} *exact* />              <*Route* *path*="/about" *component*={About} />              <*Route* *path*="/shop" *component*={Shop} />              <*Route* *component*={*Error*} />        </*Switch*>    </*BrowserRouter*>,    document.getElementById('root')  );  **\*\* Lưu ý:**  - Với **path= “/”** ta thêm một props nữa là **exact** , bởi hầu hết các path đều đi qua “/”.  - Với các URL có path không nằm trong những path được chỉ định ở trên nó sẽ đều truy cập vào <Route> không có thuộc tính path  **Trong file Home.js ta sử dụng < Link > như sau:**   import React from "react";   import { Link } from 'react-router-dom';     export default function Home() {     return (       <div>           <h1> Home Page</h1>           <*Link* *to*="/about">About / </*Link*>           <*Link* *to*="/shop">Shop / </*Link*>           <*Link* *to*="/404">404 / </*Link*>       </div>     );   };  **<Link > sẽ tạo ra các đường dẫn khi click vào sẽ dẫn đến path được chỉ định** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |