**Họ và tên: Nguyễn Quốc Tĩnh**

**ReactJS**

**1. Setup**

* NMP là viết tắt của Node package manager là một công cụ tạo và quản lý các thư viện lập trình Javascript cho Node.js. Trong cộng đồng Javascript, các lập trình viên chia sẻ hàng trăm nghìn các thư viện với các đoạn code đã thực hiện sẵn một chức năng nào đó. Nó giúp cho các dự án mới tránh phải viết lại các thành phần cơ bản, các thư viện lập trình hay thậm chí cả các framework.
* Để chạy được phần mềm NPM hệ thống của bạn cần phải cài đặt NodeJS. Phần mềm NPM được tích hợp sẵn trong bộ cài NodeJS, vì vậy khi cài đặt xong NodeJS bạn sẽ có cả hai.
* NPM cung cấp 2 chức năng chính bao gồm:
  + Là kho lưu trữ trực tuyến cho các package/module. Chúng ta có thể tìm kiếm các package trên search.nodejs.org.
  + Quản lý các module javascript và phiên bản của chúng trong các dự án của chúng ta đơn giản hơn, dễ dàng hơn, tiết kiệm thời gian hơn.
* Ví dụ:
  + npm install toastr –g

Với -g tức là ta sẽ install trên global. Nó sẽ được lưu tại usr/local/lib

* + npm install axios –save

Với --save thì ta sẽ lưu trên local. Và việc quản lý này sẽ được theo dõi tại file là package.json. Và khi tải thì nó sẽ được lưu trữ tại thư mục node\_modules trong project của chúng ta

**2. JSX**

* JSX viết tắt là Javascript XML, là một cú pháp mở rộng cho JavaScript. Chúng tôi khuyến khích sử dụng JSX với React để mô tả giao diện (UI). JSX có thể trông giống Ngôn ngữ Khuôn mẫu (Template language), nhưng JSX đi kèm với toàn bộ tính năng của JavaScript.
  + JSX giúp cho việc xây dựng các ứng dụng React một cách nhanh hơn, dễ tối ưu trong việc complie code sang javascript.
  + JSX rất dễ xem các lỗi trong quá trình triển khai bởi hầu hết các lỗi sẽ được hiển thị trong quá trình compile, không như các đoạn mã HTML có thể thừa thiếu các thể div khiến giao diện bị hiển thị sai. JSX lại hoàn toàn ngược lại, khi bạn quên đóng div chẳng hạn thì nó lập tực sẽ hiển thị lỗi.
  + Cú pháp khá giống với HTML nên dễ dàng cho việc viết chuyển đổi.
* JSX cho ra những “phần tử”(Element) React. Chúng ta sẽ khám phá việc chúng được render vào DOM như thế nào ở phần tiếp theo. Dưới đây là những kiến thức cần thiết cơ bản của JSX để có thể bắt đầu.
* Nhúng Biểu thức trong JSX: Bạn có thể nhúng bất kỳ biểu thức JavaScript nào trong JSX bằng cách đóng nó trong dấu ngoặc nhọn.
* JSX cũng là biểu thức: Sau khi biên dịch, biểu thức JSX trở thành các đối tượng JavaScript thông thường. Điều này có nghĩa là bạn có thể sử dụng JSX bên trong các câu lệnh if và cho các vòng lặp, gán nó cho các biến, chấp nhận nó như các đối số, và trả về nó từ các hàm

function getGreeting(user) {

    if (user) {

      return <h1>Hello, {formatName(user)}!</h1>;

    }

    return <h1>Hello, Stranger.</h1>;

  }

* Thuộc tính chỉ định với JSX:
  + Bạn có thể dùng dấu nháy để khai báo một chuỗi như là thuộc tính của thẻ:

const element = <div tabIndex="0"></div>;

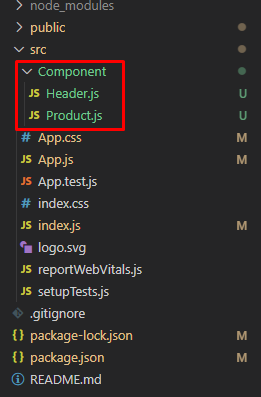
* + Bạn có thể dùng dấu ngoặc nhọn để nhúng một biểu thức Javascript vào trong thuộc tính:

const element = <img src={user.avatarUrl}></img>;

* + Đừng để dấu nháy xung quanh ngoặc nhọn khi nhúng biểu thức JavaScript vào trong một thuộc tính. Bạn có thể dùng dấu nháy (cho giá trị chuỗi) hoặc ngoặc nhọn (cho biểu thức), nhưng không được dùng cả hai cho cùng 1 thuộc tính. **3. Component and Props**

3.1. Component

* Components giúp phân chia các UI (giao diện người dùng) thành các phân nhỏ để dễ dàng quản lý và tái sử dụng. Giả sử mình có một website gồm nhiều phần bố cục khác nhau và mình muốn chia nhỏ các phần ra để dễ quản lý. Các component thực hiện công việc giống như các functions trong JavaScript nhưng chúng độc lập và nhiệm vụ chính là trả về HTML thông qua hàm render.
* Khởi tạo một React Component
  + Trước khi thực hiện viết components, chúng ta nên khởi tạo một thư mục có tên components trong thư mục src để chứa tất cả các component trong dự án.



* Có 2 loại component là:
  + Function Component
    - Ví dụ:

// Create a ES6 class component

class Message extends React.Component {

    constructor(props) {

      super(props);

      this.state = {

        content: "Hello World!"

      };

    }

    // Use the render function to return JSX component

    render() {

      return (

           <p>{this.state.content}</p>

        );

      }

  }

  const element1 = document.getElementById('root')

  // Use the ReactDOM.render to show your component on the browser

  ReactDOM.render(

      <Message />, element1

  )

* + Class Component.
    - Ví dụ:

const element1 = document.getElementById('root')

function Car() {

  return <p>Hello World!</p>;

}

ReactDOM.render(<Car />, element1);

3.2. Props

* Props là một object được truyền vào trong một components, mỗi components sẽ nhận vào props và trả về react element. Props cho phép chúng ta giao tiếp giữa các components với nhau bằng cách truyền tham số qua lại giữa các components.
* Các giá trị của props không bị thay đổi, khi muốn thay đổi trạng thái của component thì người ta chỉ thay đổi state của component chứ props thì không thay đổi được. Khi một components cha truyền cho component con một props thì components con chỉ có thể đọc và không có quyền chỉnh sửa nó bên phía components cha. Cách truyền một props cũng giống như cách mà bạn thêm một attributes cho một element HTML.
* Ví dụ:
  + Product.js

import React, { Component } from 'react';

class Product extends Component {

    render() {

        return (

            <div className="card" style={{ width: '18rem' }}>

                <img src="https://upload.motgame.vn/photos/motgame-vn/2021/05/iPhone\_13\_3.jpg" className="card-img-top" alt="..." />

                <div className="card-body">

                    <h5 className="card-title">{this.props.name}</h5>

                    <p className="card-text">{this.props.price} VND</p>

                    <a href="#" className="btn btn-primary">MUA</a>

                </div>

            </div>

        );

    }

}

export default Product;

* + Truyền props vào các Component

import React, { Component } from 'react';

import Header from './Component/Header';

import Product from './Component/Product';

class App extends Component {

  render() {

    return (

      <div>

        <Product name="Iphone 13" price="50,000,000 VND" />

      </div>

    );

  }

}

export default App;

**4. State and Lifecycle**

**4.1. State**

* Trong React, state là các thành phần để xây dựng nên một component. State là lưu lưu trữ các giá trị của component, khi state thay đổi thì component cũng được render lại.
* Đối tượng state được tạo trong hàm constructor (hàm khởi tạo) của component, state có thể chứa nhiều các thuộc tính khác nhau.
* Ví dụ:

class Name extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      firstName: "kai",

      lastName: "tran"

    };

  }

  changeName = () => {

    this.setState({firstName: "sena"});

  this.setState({lastName: "nguyen"});

  }

  render() {

    return (

      <div>

        <p>Hello: {this.state.firstName} {this.state.lastName}</p>

    <br/>

        <button type="button" onClick={this.changeName}>change name</button>

      </div>

    );

  }

}

const rootElement = document.getElementById('root')

ReactDOM.render(

    <Name />, rootElement

)

* + Trong đó:
    - Tạo state:

constructor(props) {

    super(props);

    this.state = {

      firstName: "kai",

      lastName: "tran"

    };

  }

* + - Thay đổi state:

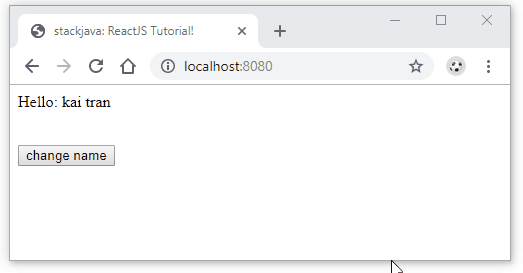
changeName = () => {

    this.setState({firstName: "sena"});

  this.setState({lastName: "nguyen"});

  }

* + - Kết quả: click vào button tên kai tran => sena nguyen



4.2. Lifecycle

* Trong React Component, một chu kì cũng xuất hiện, components được khởi tạo (hiển thị ra DOM), update, và kết thúc (unmount),..đó được gọi là một component life cycle. React cho phép chúng ta tham gia vào các giai đoạn của mỗi component bằng cách sử dụng các phương thức được xây dựng sẵn trong mỗi giai đoạn đó. Khi một components được khởi chạy nó sẽ phải trải qua 4 giai đoạn chính:
  + Initialization: Đây là giai đoạn mà thành phần sẽ bắt đầu hành trình của mình bằng cách khởi tạo state và props. Điều này thường được thực hiện bên trong phương thức constructor.
  + Mounting: Giai đoạn này được thực hiện sau khi quá trình initialization(khởi tạo) được hoàn thành. Nó thực hiện nhiệm vụ chuyển virtual DOM (DOM ảo) trong React thành DOM và hiển thị trên trình duyệt. Component sẽ được render lần đầu tiên, ở đây chúng ta có 2 phương thức để tham gia vào giai đoạn này: componentWillMount(), componentDidMount().
  + Updating: Trong giai đoạn này, dữ liệu của các phần (props & state) sẽ được cập nhật để đáp ứng với các sự kiện của người dùng như click, gõ, v.v. Điều này dẫn đến việc re-render lại component, ở trong giai đoạn này chúng ta sẽ có 3 phương thức chính: shouldComponentUpdate(),componentWillUpdate(),ComponentDidUpdate().
  + Unmounting: Đây là bước cuối cùng trong mỗi component, khi tất cả các tác vụ hoàn thành và bạn tiến hành unmount DOM. Quá trình này chỉ có duy nhất 1 phương thức đó là componentWillUnmount().

**5. Handling Events**

* Xử lý các sự kiện trong React rất giống với xử lý các sự kiện trên các phần tử DOM. Có một số khác biệt về cú pháp:
  + Các sự kiện React được đặt tên bằng camelCase, thay vì chữ thường. Ví dụ: onclick -> onClick, onchange -> onChange
  + Với JSX, bạn truyền một hàm để bắt sự kiện, thay vì một chuỗi như HTML thông thường.
* Một điểm khác biệt nữa là bạn không thể quay lại falseđể ngăn hành vi mặc định trong React. Bạn phải gọi preventDefaultmột cách rõ ràng. Ví dụ: với HTML thuần túy, để ngăn hành vi gửi biểu mẫu mặc định, bạn có thể viết:

<form onsubmit="console.log('You clicked submit.'); return false">

    <button type="submit">Submit</button>

  </form>

Trong React, thay vào đó có thể là:

function Form() {

    function handleSubmit(e) {

      e.preventDefault();

      console.log('You clicked submit.');

    }

    return (

      <form onSubmit={handleSubmit}>

        <button type="submit">Submit</button>

      </form>

    );

  }

Trong đó, e là một sự kiện tổng hợp. React xác định các sự kiện tổng hợp này theo W3C spec, vì vậy bạn không cần phải lo lắng về khả năng tương thích giữa các trình duyệt.

* Ví dụ: Toggle thành phần này hiển thị một nút cho phép người dùng chuyển đổi giữa trạng thái “BẬT” và “TẮT”:

class Toggle extends React.Component {

    constructor(props) {

      super(props);

      this.state = {isToggleOn: true};

      // This binding is necessary to make `this` work in the callback

      this.handleClick = this.handleClick.bind(this);

    }

    handleClick() {

      this.setState(prevState => ({

        isToggleOn: !prevState.isToggleOn

      }));

    }

    render() {

      return (

        <button onClick={this.handleClick}>

          {this.state.isToggleOn ? 'ON' : 'OFF'}

        </button>

      );

    }

  }

  ReactDOM.render(

    <Toggle />,

    document.getElementById('root')

  );

* Cần phải thận trọng với việc sử dụng this trong việc JSX callbacks. Trong Javascript, các method của class không bị ràng buộc bởi mặc định. Nếu bạn quên không gọi this.handleClick mà gọi thằng onClick khi đó this sẽ bị lỗi undefined khi mà function được gọi.
* Để truyền tham số vào event handlers, có thể dùng theo 1 trong số cách sau:

<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>

<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>

Trong cả hai trường hợp, tham số e đại diện cho React Event sẽ được truyền như là tham số thứ 2 sau id.

**6. Router**

* React Router là một thư viện cho việc điều hướng URL tiêu chuẩn trong React, Nó cho phép chúng ta có thể đồng bộ UI với URL. Được thiết kế với API đơn giản, từ đó cho phép giải quyết các vấn đề về URL một cách nhanh chóng.
* Để sử dụng React Router chúng ta cần phải cài đặt thư viện này vào trong dự án React bằng cách sử dụng NPM: npm install react-router-dom
* Sau khi cài đặt thành công, trong trường hợp cần dùng đến React Router bạn chỉ cần import nó component đó:

import { BrowserRouter, Route, Switch } from 'react-router-dom';

* Các thành phần trong React-Router:
  + **BrowserRouter vs HashRouter**: React-Router cung cấp cho chúng 2 thành phần hay sử dụng đó là BrowserRouter & HashRouter. Hai thành phần này khác nhau ở kiểu URL mà chúng sẽ tạo ra và đồng bộ:
    - BrowserRouter: Được sử dụng phổ biến hơn, nó sử dụng History API có trong HTML5 để theo dõi lịch sử bộ định tuyến của bạn.
    - HashRouter: Sử dụng hash của URL (window.location.hash) để ghi nhớ mọi thứ.
* **Route**: Định nghĩa một ánh xạ (mapping) giữa một URL và một Component. Điều đó có nghĩa là khi người dùng truy cập theo một URL trên trình duyệt, một Component tương ứng sẽ được render trên giao diện.

Trong đó:

* + path: Là đường dẫn trên URL.
  + exact: Giúp cho route này này chỉ hoạt động nếu URL trên trình duyệt phù hợp tuyệt đối với giá trị của thuộc tính path của nó.
  + component: Là component sẽ đươc load ra tương ứng với Route đó.
* **Link**: Trong HTML thì cặp thẻ để chuyển hướng đó là thẻ <a></a> thì trong react chúng ta sẽ dử dụng cặp thẻ <Link></Link> được import từ React-Router. Ví dụ: <Link to="/about">About</Link> , trong đó: **to** giống như thuộc tính href trong thẻ a.
* **NavLink**: rất giống với Link về cách sử dụng, nhưng NavLink tốt hơn vì nó hỗ trợ thêm một số thuộc tính như là activeClassName và activeStyle 2 thuộc tính này giúp cho khi mà nó trùng khớp thì nó sẽ được active lên và chúng ta có thể style cho nó. Ví dụ:

<NavLink exact activeStyle={{

    backgroundColor : 'white',

    color : 'red'

}} to="/" className="my-link">Trang Chu</NavLink>

* **Custom Link**: ở trên ta có thẻ NavLink giúp chúng ta có thêm một thuộc tính nhưng giả sử khi bạn không muốn activeClassName hoặc activeStyle tại thẻ NavLink mà nó lại nằm ở một thẻ bao nó ví dụ như thẻ div hay thẻ li thì sao? sau đây mình sẽ custom lại để có thể sử dụng các class hoặc style ở thẻ bao ngoài của nó. Ví dụ:

const MenuLink = ({

    label, // nội dung trong thẻ

    to, // giống như href trong thẻ a

    activeOnlyWhenExact

}) => {

    return (

        <Route

            path={to}

            exact={activeOnlyWhenExact}

            children={ ({ match }) => { //match la doi tuong xac dinh su trung khop cua URL

                var active = match ? 'active abc' : '';

                return (

                    <li className={`my-li ${active}`}>

                        <Link  to={to} className="my-link">{label}</Link>

                    </li>

                );

            }}

        />

    );

}

* Đối tượng Match: Khi bạn muốn lấy một số thông tin ở trên URL thì bạn có thể dùng đối tượng match để lấy dữ liệu về. Tại cấu hình Router ta chỉ cần truyền thêm đối tượng match vào component mà cần sử dụng đối tượng match.

{

    path : '/products',

    exact : false,

    main : ({match}) => <Products match={match} />

}

* Đối tượng prompt - Xác nhận trước khi chuyển trang: Giả sử khi bạn đang nhập liệu ở form nào đó mà không may click nút back hay chuyển trang thì thôi xong dữ liệu bạn nhập sẽ mất hết để khác phục điều đó ta có đối tượng prompt nó sẽ giúp chúng ta trước khi back hay chuyển trang nó sẽ xác nhận xem là chúng ta có chắc chắn muốn back hay chuyển trang không!
  + Khi muốn sử dụng đối tượng prompt thì chúng ta chỉ cần import nó từ react-router

import {Prompt} from 'react-router-dom';

<Prompt

    when={true} // true | false

    message={ (location) => (`Ban chac chan muon di den ${location.pathname}`) }

/>

* Redirect
  + Chức năng dùng để chuyển trang.
  + Có thể truy xuất thông tin trang trước đó thông qua đối tượng location. Để sử dụng Redirect ta chỉ cần import nó từ react-router.

import { Redirect } from 'react-router-dom';

* + Khi bạn muốn sử dụng location thì tại cấu hình Router ta chỉ cần truyền thêm đối tượng location vào component mà cần sử dụng đối tượng location.

{

    path : '/login',

    exact : false,

    main : ({location}) => <Login location={location} />

}