**React JS**

* **Babel**
* Babel is a JavaScript compiler that converts modern JavaScript code into a version compatible with all browsers. Babel enables React developers to use the latest JavaScript syntax in their components. Babel transpiles modern JavaScript for use in React components and all browsers.
* **What is Component**
* Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
* Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components. Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.
* Components are used to divide our pages UI into small piece and when we want to isolate our code and it does not effect on other component this is called component.
* **Function Component Router in React JS**
* **MainRouter.jsx**

import React, { Suspense } from "react";

import { createBrowserRouter } from "react-router-dom";

// import { Link } from "react-router-dom";

import Main from "../Pages/Main";

import Contact from "../Pages/Contact";

import About from "../Pages/About";

import HeaderFile from "../Component/HeaderFile";

import Example from "../Pages/Example";

const Functionalconporoute = React.lazy(() => { return import('./Functionalconporoute.jsx') })

const Classcomporoute = React.lazy(() => { return import('./Classcomporoute.jsx') })

const MainRouter = createBrowserRouter([

  {

    path: "/",

    element: (

      <>

        <HeaderFile />

        <Main />

        {/\* <Link to="/"></Link> \*/}

      </>

    ),

  },

  {

    path: "/about",

    element: (

      <>

        <HeaderFile />

        <About />

        {/\* <Link to="/"></Link> \*/}

      </>

    ),

  },

  {

    path: "/contact",

    element: (

      <>

        <HeaderFile />

        <Contact />

        {/\* <Link to="/"></Link> \*/}

      </>

    ),

  },

  {

    path: "/example",

    element: (

      <>

        <HeaderFile />

        <Example />

        {/\* <Link to="/"></Link> \*/}

      </>

    ),

    children: [

      {

        path: "/example/functionalcomponent/\*",

        element: <Suspense fallback={<>Loading....</>} ><Functionalconporoute /></Suspense>

      },

    ]

  },

  {

    path: "/example",

    element: (

      <>

        <HeaderFile />

        <Example />

        {/\* <Link to="/"></Link> \*/}

      </>

    ),

    children: [

      {

        path: "/example/classcomponent/\*",

        element: <Suspense fallback={<>Loading....</>} ><Classcomporoute /></Suspense>

      },

    ]

  }

]);

export default MainRouter;

* **Functionalconporoute.jsx**

import React from "react";

import { useRoutes } from "react-router-dom";

import Functioncompomenu from "./../Component/FunctionComponent/01Functioncompomenu.jsx";

import Functioncompointro from "./../Component/FunctionComponent/02Functioncompointro.jsx";

const Functionalconporoute = () => {

    const routes = useRoutes([

        {

            path:"/",

            element:<Functioncompomenu/>,

            children:[

                {

                    path:"functioncompointro",

                    element:<Functioncompointro/>

                }

            ]

        }

    ])

    return routes;

}

export default Functionalconporoute;

* **Functioncompomenu.jsx**

import React from "react";

import { Link, Outlet } from "react-router-dom";

const Functioncompomenu = () => {

    return (<>

        <div className="row">

            <div className="col offset-6">

                <ol>

                    <li><Link to="functioncompointro">Functional Components Intro</Link> </li>

                </ol>

            </div>

        </div>

        <div className="row">

            <div className="col">

                <Outlet></Outlet>

            </div>

        </div>

    </>);

}

export default Functioncompomenu;

* When we want to print child component then we use Outlet.
* **Functioncompointro.jsx**

import React from "react";

const Functioncompointro = () => {

    return ( <>

        <div className="row">

            <div className="col offset-6">

                <h2>Functioncompointro</h2>

            </div>

        </div>

    </> );

}

export default Functioncompointro;

* **What is class component**
* Class component defined with class keyword. It extends the Component. In class component we provide render(),
* We can’t create object here so if we want to access some functionality we use render.
* Render() basically return functionality it always written JSX.
* Here render means when we use class component if we want to return anything we write render function. This is class component so class component doesn’t return function expect function component but when we write render it allows returning in class component. Without render it gives error.
* **Class component Router**
* Main router file is same as function component route.
* After lazy loading of classcomporoute file it is same as function component route.
* **Classcompomenu.jsx**

import React from "react";

import { Link, Outlet } from "react-router-dom";

import { Component } from "react";

class Classcompomenu extends Component {

    // state = {}

    render() {

        return (<>

            <div className="row">

                <div className="col">

                    <ol>

                        <li><Link to="classcompointro">Class Components Intro</Link> </li>

                    </ol>

                </div>

            </div>

            <div className="row">

                <div className="col">

                    <Outlet></Outlet>

                </div>

            </div>

        </>);

    }

}

export default Classcompomenu;

* **Classcompointro.jsx**

import React from "react";

import { Component } from "react";

class Classcompointro extends Component {

    // state = {}

    render() {

        return (<>

            <div className="row">

                <div className="col">

                    <h2>Classcompointro</h2>

                </div>

            </div>

        </>);

    }

}

export default Classcompointro;

* **JSX**
* JSX stands for JavaScript XML
* JSX allows us to write your html within JavaScript in very easy manner.
* JSX is an extension of JavaScript that allows you to write HTML in JavaScript file.
* JSX syntax is not understand by browser and so we need transpiler like babel to convert the code from JSX to JavaScript.
* Example of JSX
* When we want to print something in JSX we use { }
* <input type="text" name="" id="" value={name}></input>
* Username: 
* In above Example we pass value in input tag because of Virtual dom we can't be able to edit or delete input tag value.
* **JSX.jsx**

import React from "react";

import { Component } from "react";

class JSX extends Component {

    render() {

        let name = "dishank"

        let input = '<input type="text" name="" id="" value={name}></input>';

        let inputtag = <input type="text" value={name} name="" id=""></input>;

        return (

            <>

                <div className="row">

                    <div className="col-6">

                        <p>JSX stands for JavaScript XML</p>

                        <p>JSX allows us to write your html within JavaScript in very easy manner.</p>

                        <p>JSX is an extension of JavaScript that allows you to write HTML in JavaScript file.</p>

                        <p>JSX syntax is not understand by browser and so we need transpiler like babel to convert the code from JSX to JavaScript.</p>

                        Example of JSX

                        <p>When we want to print something in JSX we use &#123; &#125; </p>

                        <p>{input}</p>

                        <p>Username: {inputtag}</p>

                        <p>In above Example we pass value in input tag because of Virtual dom we can't be able to edit or delete input tag value.</p>

                        <p>6+6 = {6 + 6}</p>

                    </div>

                </div>

            </>

        );

    }

}

export default JSX;

* **Event Syntax**
* Wrong <button onClick={this.event()}>onClick={this.function\_name()}>.
* If we want to call the function then this.functionname() these is a wrong way to call the function in class component.
* {this.functionname} is only right way to call function.

import { Component } from "react";

class Event extends Component {

    event() {

        alert("called");

    }

    render() {

        return (

            <>

                <div className="row">

                    <div className="col-6">

                    </div>

                </div>

                <p>

                    react-dom.development.js:86 Warning: You provided a `value` prop to a form field without an `onChange` handler. This will render a read-only field. If the field should be mutable use `defaultValue`. Otherwise, set either `onChange` or `readOnly <br /><br />

                    &lt;button onclick="event()"&gt;Click onclick="event()"&lt;/button&gt;

                    {/\* <button onclick="kaiPanEvent()">Click onclick="kaiPanEvent()"</button>   \*/}

                </p>

                <br />

                {/\* <button onclick={kaiPanEvent()}>Click onclick=&#123;kaiPanEvent()&#125;</button> \*/}

                {/\* <button onClick={kaiPanEvent}>onclick=&#123;kaiPanEvent&#125; Wrong</button>   \*/}

                Wrong

                &lt;button onClick=&#123;this.event()&#125;&gt;onClick=&#123;this.function\_name()&#125;&gt;

                <p>If we want to call the function then this.functionname() these is a wrong way to call the function in class component</p>

                <br />

                Correct

                <button onClick={this.event}>onClick=&#123;this.function\_name&#125;</button>

                <p>&#123;this.functionname&#125; is only right way to call function</p>

                <br />

                <br />

            </>

        );

    }

}

export default Event;

* **State in Class component**
* Constuctor : constructor is method that invokes by default when object are created.
* Super: Super() function is to call the constructor of the parent class. It is used when we need to access a few variables in the parent class.
* State: State is a React JS variable. It is mutable it means when we want to change the data into runtime then we use state.
* In normal variable we cannot change the value in react so state comes into picture Using the setState we are able to change the value any time.
* two ways to create a function with arrow function we don't require to bind the function whenever we want to change the state value
* If we create a normal function then we need to bind using below syntax:
* this.changeStateData2 = this.changeStateData2.bind(this);

import { Component } from "react";

class StateinClass extends Component {

    // usernamedm = "Data Member"

    constructor() {

        super(); //Must call super constructor in derived class before accessing 'this' or returning from derived constructor

        // this.usernamedm = "Something from DM"

        // this.state = { usernamedm: "Something from DM" }

        // console.log(this.usernamedm);

        // console.log("Constuctor calling");

        this.state = { data: "Test" }

        this.state = { data2: "Test2" }

        // this.clickHandle = this.clickHandle.bind(this);

        this.changeStateData2 = this.changeStateData2.bind(this);

    }

    // databuilding(){ let variable = "some variable" }

    // datachanging(){}

    // var username = "Something"

    // clickHandle = () => {

    //     // this.setState({ usernamedm: "Dishank" })

    //     // console.log(this.usernamedm);

    //     // console.log("checking");

    //     // username = "change"

    // }

    // changeStateData() {

    //     this.setState({ DataMariMarji: "Testing from changeStateData" })

    // }

    changeStateData = () => {

        this.setState({ data: "Change data without bind" })

        console.log(this.state.data);

    }

    changeStateData2() {

        this.setState({ data2: "Change data with bind" })

        console.log(this.state.data2);

    }

    render() {

        // console.log("From Render",this.usernamedm);

        // function clickHandle() {

        //     console.log(this.usernamedm);

        //     // console.log("checking");

        //     // username = "change"

        // }

        // var username = "Something"

        return (

            <>

                <div className="row">

                    <div className="col-6">

                        <p>Constuctor : constructor is method that invokes by default when object are created</p>

                        <p>Super: Super() function is to call the constructor of the parent class. It is used when we need to access a few variables in the parent class. </p>

                        <p>State: State is a React JS variable. It is mutable it means when we want to change the data into runtime then we use state.</p>

                        <p>In normal variable we cannot change the value in react so state comes into picture Using the setState we are able to change the value any time.</p>

                        <p>two ways to create a function with arrow function we don't require to bind the function whenever we want to change the state value</p>

                        <p>If we create a normal function then we need to bind using below syntax:</p>

                        <p>this.changeStateData2 = this.changeStateData2.bind(this);</p>

                        {/\* <button onClick={this.clickHandle}>Click</button>

                <button onClick={() => { console.log("btn Clicked"); }}>Click</button> \*/}

                        {/\* <button onClick={() => { this.setState({ DataMariMarji: "change" }) }}>State change{this.state.DataMariMarji}</button> \*/}

                        <button onClick={this.changeStateData}>Change data without bind</button>

                        <button onClick={this.changeStateData2}>Change data with bind</button>

                    </div>

                </div>

            </>

        );

    }

}

export default StateinClass;

* **Props in Class component**
* Props stand for "**Properties**." They are **read-only** components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
* It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.
* Props are used to pass the data from one component to another (Parent component to child component).
* **PropsChild.jsx**

import { Component } from "react";

import {

    MDBCard,

    MDBCardBody,

    MDBCardTitle,

    MDBCardText,

    MDBCardImage,

    MDBBtn

} from 'mdb-react-ui-kit';

class PropsChild extends Component {

    render() {

        return (<>

            <MDBCard>

                <MDBCardImage src={this.props.image} position='top' alt='React JS' />

                <MDBCardBody>

                    <MDBCardTitle className="text-center">{this.props.title}</MDBCardTitle>

                    <MDBCardText>

                        Some quick example text to build on the card title and make up the bulk of the card's content.

                    </MDBCardText>

                    <MDBBtn href='#'>Button</MDBBtn>

                </MDBCardBody>

            </MDBCard>

        </>);

    }

}

export default PropsChild;

* **Props.jsx**

import { Component } from "react";

import PropsChild from "./08PropsChild";

import ReactJS from "./ReactJS.png"

class Props extends Component {

    render() {

        return (<>

            <div className="row">

                <div className="col-3">

                    <PropsChild title="React js Live Url image" image="load-image-Url” />

                </div>

                <div className="col-3">

                    <PropsChild title="React js Component load image" image={ReactJS} />

                </div>

                <div className="col-3">

                    <PropsChild title="Public folder image(Relative path of Image)" image="/ReactJS.png" />

                </div>

                <div className="col-3">

                    <PropsChild title="Absolute Path" image={`${process.env.PUBLIC\_URL}/ReactJS2.jpg`} />

                </div>

                <div className="col-6">

                    <p>There are four ways to pass the image</p>

                    <ol>

                        <li>Live Url image Load</li>

                        <p>We load the image from internet copy the url and paste whenever our child component is loaded and receive using &#123;this.props.nameofParameter&#125;</p>

                        <li>Using Load image</li>

                        <p>We download the image and save it whenever component is there then import it load it and use this image to pass. </p>

                        <li>Using Relative Path</li>

                        <p>We save the image inside public folder whenever we want to get the image we load image using /imagename.extension of the image</p>

                        <li>Using Absolute Path</li>

                        <p>We use &#123;`$&#123;process.env.PUBLIC\_URL&#125;/imagename to pass the image </p>

                    </ol>

                </div>

            </div>

        </>);

    }

}

export default Props;

* We load Child file in parent.
* **There are four ways to pass the image**

1. **Live Url image Load**

We load the image from internet copy the url and paste whenever our child component is loaded and receive using {this.props.nameofParameter}

1. **Using Load image**

We download the image and save it whenever component is there then import it load it and use this image to pass.

1. **Using Relative Path**

We save the image inside public folder whenever we want to get the image we load image using /imagename.extension of the image

1. **Using Absolute Path**

We use {`${process.env.PUBLIC\_URL}/imagename to pass the image

* **State Life Cycle**

****

* **StateLifeCycleClassCompo.jsx**

import { Component } from "react";

class StateLifeCycleClassCompo extends Component {

    constructor(props) {

        super(props);

        console.log("constructor");

        this.state = {

            count: 0

        }

    }

    // componentWillMount() {

    //     console.log("componentWillMount");

    // }

    componentDidMount() {

        this.setState({ count: this.state.count + 1 });

        console.log("componentDidMount");

    }

    // componentWillReceiveProps(nextProps) {

    // }

    // getSnapshotBeforeUpdate() {

    //     // console.log("getSnapshotBeforeUpdate");

    //     // return { count: props.count }

    //     return document.getElementById("p1").innerHTML = "Before the update" + this.state.count;

    // }

    shouldComponentUpdate(nextProps, nextState) {

        console.log("shouldComponentUpdate");

        // return false

        return true

    }

    // componentWillUpdate(nextProps, nextState) {

    // }

    componentDidUpdate(prevProps, prevState) {

        console.log("componentDidUpdate");

    }

    componentWillUnmount() {

        console.log("componentWillUnmount");

    }

    updateCount = () => {

        this.setState({ count: this.state.count + 1 });

    }

    render() {

        console.log("render");

        return (

            <>

                <div className="row">

                    <div className="col-6">

                        <h3>ComponentWillMount</h3>

                        <p>Before the component rendered ComponentWillMount will call. It is not necessary to use because constructor do the same thing what ComponentWillMount has done</p>

                        <h3>componentDidMount</h3>

                        <p>When the component is created after these method load</p>

                        <p>The componentDidMount() method is called after the component is rendered.</p>

                        <h3>componentWillReceiveProps</h3>

                        <p>This method is used during the updating phase of the React lifecycle. This function is generally called if the props passed to the component change. It is used to update the state in response with the new received props. setState() method doesn’t generally call this method again.</p>

                        <p>When we recive the props before receive the props when we want to print something then we use componentWillReceiveProps.</p>

                        <h3>shouldComponentUpdate</h3>

                        <p>This method makes the component to re-render only when there is a change in state or props of a component and that change will affect the output.</p>

                        <p>When we want to update something in our component then shouldComponentUpdate checks weather return true or false if return false then it doesn't allow to update if return true then update successfully. By default the method always return true.</p>

                        <p>once the props or state are written for updating it is forcefully updated.</p>

                        <h4>The main difference between componentWillReceiveProps and shouldComponentUpdate is that componentWillReceiveProps it allows us to update for nextProps (Props) and shouldComponentUpdate it allows us to update both state and props values.</h4>

                        <h3>componentWillUpdate</h3>

                        <p>This function is generally called before the component is updated or when the state or props passed to the component changes. Don't call setState() method in this function. This method will not be invoked if shouldComponentUpdate() methods return false.</p>

                        <h3>componentDidUpdate</h3>

                        <p>This method work when the state and props are changed.</p>

                        <h3>componentWillUnmount</h3>

                        <p>This method works when we move to one component to other or close the component.</p>

                        <button onClick={this.updateCount}>Click {this.state.count}</button>

                        <p id="p1"></p>

                    </div>

                </div>

            </>

        );

    }

}

export default StateLifeCycleClassCompo;

* **ComponentWillMount**
* Before the component is created ComponentWillMount will call. It is not necessary to use because constructor do the same thing what ComponentWillMount has done
* **componentDidMount**
* When the component is created after these method load
* The componentDidMount() method is called after the component is rendered.
* **componentWillReceiveProps**
* This method is used during the updating phase of the React lifecycle. This function is generally called if the props passed to the component change. It is used to update the state in response with the new received props. setState() method doesn’t generally call this method again.
* When we receive the props before receive the props when we want to print something then we use componentWillReceiveProps.
* **shouldComponentUpdate**
* This method makes the component to re-render only when there is a change in state or props of a component and that change will affect the output.
* When we want to update something in our component then shouldComponentUpdate checks weather return true or false if return false then it doesn't allow updating if return true then update successfully.
* Once the props or state are written for updating it is forcefully updated.
* The main difference between componentWillReceiveProps and shouldComponentUpdate is that componentWillReceiveProps it allows us to update for nextProps (Props) and shouldComponentUpdate it allows us to update both state and props values.
* **componentWillUpdate**
* This function is generally called before the component is updated or when the state or props passed to the component changes. Don't call setState() method in this function. This method will not be invoked if shouldComponentUpdate() methods return false.
* **componentDidUpdate**
* This method work when the state and props are changed.
* **componentWillUnmount**
* This method works when we move from one component to other or close the component.
* **StateLifeCycleLoader.jsx**

import { Component } from "react";

import LoaderCompo from "./11LoaderCompo";

class StateLifeCycleLoader extends Component {

    constructor(props) {

        console.log("called constructor");

        super(props);

        // Declare State with default value for loader enable disable functionalities START

        this.state = { data: "Something", loader: false }

        // Declare State with default value for loader enable disable functionalities END

        // Using setTimeOut function make chnages in state after 2sec START

        // Using setTimeOut function make chnages in state after 2sec END

    }

    componentDidMount() {

        console.log("componentDidMount");

        setTimeout(()=>{

            this.setState({loader:true})

        },2000)

    }

    // changeDataFunc = () => {

    //     if (!this.state.loader) {

    //         <LoaderCompo />

    //         setTimeout(() => {

    //             // console.log("called");

    //             this.setState({ loader: true })

    //         }, 2000)

    //     }

    //     else {

    //         <p>Data</p>

    //     }

    // }

    // shouldComponentUpdate(nextProps, nextState) {

    //     console.log("shouldComponentUpdate");

    //     // return false

    //     return true

    // }

    // componentDidUpdate(prevProps, prevState) {

    //     console.log("componentDidUpdate");

    //     setTimeout(() => {

    //         console.log("componentDidUpdate time out");

    //         this.setState({loader:false})

    //     }, 4000);

    // }

    // componentWillUnmount() {

    //     console.log("componentWillUnmount");

    // }

    render() {

        console.log("called render");

        // variable declaration with diff default values and its impact

        // let HtmlDisp = null;  //variable with null value will be adopt any data with its type

        // const HtmlDisp = "";  //variable with empty value will be adopt any value with only string type

        // const HtmlDisp = 0;

        // const HtmlDisp = {};

        // const HtmlDisp = [];

        // if (this.state.loader) {

        //     HtmlDisp = <img src="https://miro.medium.com/v2/resize:fit:1400/1\*CsJ05WEGfunYMLGfsT2sXA.gif" alt="" />

        // } else {

        //     HtmlDisp = "inside else"

        // }

        // console.log("render");

        console.log(this.state.loader);

        return (

            <div>

                <p>Change Data</p>

                {JSON.stringify(this.state.loader)}

                {this.state.loader ? <p>Data</p> : <LoaderCompo/>}

                {/\* {this.state.loader ? <img src="https://miro.medium.com/v2/resize:fit:1400/1\*CsJ05WEGfunYMLGfsT2sXA.gif" alt="" /> : <p>Data</p>}  \*/}

                {/\* <button onClick={this.changeDataFunc}>Click</button> \*/}

                {/\* {HtmlDisp} \*/}

                <button onClick={() => this.setState({ loader: !this.state.loader })}>Click</button>

                <p><strong>State</strong>{this.state.data}</p>

            </div>

        );

    }

}export default StateLifeCycleLoader;

* First declare the state the default value is false
* compomenetDidMount we set the timeout in 2 seconds it is automatically true.
* If the condition is true then print the data otherwise print <LoaderCompo/> data here I pass the image gif.
* On button click event we set the setState if the value is true then it becomes false and false it becomes true it is like toggle menu.
* **LoaderCompo.jsx**

import { Component } from "react";

class LoaderCompo extends Component {

    render() {

        return (<>

            <img src="https://miro.medium.com/v2/resize:fit:1400/1\*CsJ05WEGfunYMLGfsT2sXA.gif" alt="" />

        </>);

    }

}

export default LoaderCompo;

* **Conditional Rendering**

import { Component } from "react";

class ConditionalRendering extends Component {

    constructor(props) {

        super(props);

        this.state = { isLogin: true }

    }

    render() {

        return(

            <>

            {/\* {(this.state.isLogin)  ?<> <button onClick={()=>{ this.setState({ isLogin:!this.state.isLogin }) }}>Logout</button></>:<> <button onClick={()=>{ this.setState({ isLogin:!this.state.isLogin }) }}>Login</button></>} \*/}

            <> <button onClick={()=>{ this.setState({ isLogin:!this.state.isLogin }) }}>{(this.state.isLogin)?"Login" :"Logout"}</button></>

            </>

        )

    // render() {

    //     if (this.state.isLogin) {

    //         return (<>

    //             {/\* <button onClick={()=>{ this.setState({ isLogin:false }) }}>Login</button> \*/}

    //             <button onClick={() => { this.setState({ isLogin: !this.state.isLogin }) }}>Logout</button>

    //         </>);

    //     }

    //     else {

    //         return (<>

    //             {/\* <button onClick={()=>{ this.setState({ isLogin:false }) }}>Login</button> \*/}

    //             <button onClick={() => { this.setState({ isLogin: !this.state.isLogin }) }}>Login</button>

    //         </>);

    //     }

    }

}

export default ConditionalRendering;

* There are two ways for conditional rendering either return state for each condition on use ternary operator we use ternary operator.
* **StateLifeCycleLoaderTask.jsx**

import { Component } from "react";

import LoaderTheme1 from "./13LoaderTheme1";

import LoaderTheme2 from "./14LoaderTheme2";

class StateLifeCycleLoaderTask extends Component {

    constructor(props) {

        super(props);

        this.state = { loader: true }

    }

    render() {

        return (

            <>

                <button onClick={() => this.setState({ loader: !this.state.loader })}>{this.state.loader ? <LoaderTheme1 /> : <LoaderTheme2 />}</button>

            </>);

    }

}

export default StateLifeCycleLoaderTask;

* First we create a state and default value of state is true.
* On button click event we set the setState to opposite true to false or false to true.
* We check the condition if state is true then LoaderTheme1 will load otherwise LoaderTheme2 will load.
* **LoaderTheme1.jsx**

import { Component } from "react";

class LoaderTheme1 extends Component {

    render() {

        return (<>

            <div className="spinner-grow text-primary" role="status">

                <span class="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow text-secondary" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow text-success" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow text-danger" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow text-warning" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow text-info" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow text-light" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow text-dark" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

        </>);

    }

}

export default LoaderTheme1;

* **LoaderTheme2.jsx**

import { Component } from "react";

class LoaderTheme2 extends Component {

    render() {

        return (<>

            <div className="spinner-border spinner-border-sm" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow spinner-grow-sm text-primary" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-border spinner-border-sm text-secondary" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow spinner-grow-sm text-danger" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-border spinner-border-sm" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow spinner-grow-sm" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-border spinner-border-sm" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

            <div className="spinner-grow spinner-grow-sm" role="status">

                <span className="visually-hidden">Loading...</span>

            </div>

        </>);

    }

}

export default LoaderTheme2;