**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 containing a render which helps to return a JSX is called Class component
* **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 component 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;

* **Class List and Keys**
* List always works in an array.
* Keys always work in an object.
* Map is a method which allows us to return something and it always returns a new array.

import { Component } from "react";

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

class ClassListandKeys extends Component {

    render() {

        const numbers = [1, 2, 3, 4, 5];

        const ResponsefromMap = numbers.map((data) => {

            console.log(data);

            return data \* 2;

        }) console.log(ResponsefromMap);

        // const Menu = ["Home", "About", "Contact", "Login"];

        // const MenuReturn = Menu.map((menu, index) => {

        //     console.log(menu, index);

        //     return menu;

        // })

        // console.log(MenuReturn);

        const Menu = {

            "homepage": "home",

            "about": "About us",

            "contact": "Contact US",

            "Login": "Login Page"

        };

        // console.log(Menu);

        // console.log(Object.keys(Menu));

        // console.log(Object.values(Menu));

        // console.log(Object.entries(Menu));

        const ResponseFromForEach = numbers.forEach(element => {

            console.log(element);

            // return element \* 2;

        });

        console.log(ResponseFromForEach);

        const DymenicMenu = Object.entries(Menu).map(([key, val], index) => {

            console.log(key, val);

            // return val;

            return <li key={index}> <Link to={key}>{val}</Link></li>

        })

        // console.log(DymenicMenuResturant);

        return (<>

            <ul>

                {DymenicMenu}

                {/\* <a href="https://codepen.io/mingshin430/pen/RwPoKy?editors=1010">Slider Reference for json object</a> \*/}

            </ul>

        </>);

    }

}

export default ClassListandKeys;

* Here I create an array for iterate I use map and data print one by one if we want to return a value then in map it is possible to return the value.
* In keys we create an Object if we want to iterate we use map because of an object we use Object.entries to print both key and value.
* **Dynamic Sub Menu**

import { MDBDropdown, MDBDropdownItem, MDBDropdownMenu, MDBDropdownToggle, MDBNavbarItem } from "mdb-react-ui-kit";

import { Component } from "react";

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

class DynamicSubmenu extends Component {

    render() {

        const menuItems = [

            {

                title: 'Services',

                url: '/services',

                submenu: [

                    {

                        title: 'web design',

                        url: 'web-design',

                    }, {

                        title: 'web development',

                        url: 'web-dev',

                    }, {

                        title: 'SEO',

                        url: 'seo',

                    },

                ],

            },

        ];

        const MenuData = menuItems.map((data) => {

            console.log(data);

            let submenudata = data.submenu.map((submenu) => {

                console.log("called inner loop", submenu);

                return <MDBDropdownItem><Link to={submenu.url}>{submenu.title}</Link></MDBDropdownItem>

            })

            return <MDBNavbarItem>

                <MDBDropdown>

                    <MDBDropdownToggle tag='a' className='nav-link' role='button'>

                        {data.title}

                    </MDBDropdownToggle>

                    <MDBDropdownMenu>

                        {submenudata}

                    </MDBDropdownMenu>

                </MDBDropdown>

            </MDBNavbarItem>

        })

        return (<>

            <div>

                {MenuData}

            </div>

        </>);

    }

}

export default DynamicSubmenu;

* Here we have array inside array so, we have one loop for outer menu print and one for inner menu.
* In outer menu(parent) I have just print data but inside data I have again map for sub menu.
* I use MDBDropdownItem for dropdown and print title and url of submenu.
* MDBNavbaritem for navbar creation.
* MDBDropdown allow us to drop down the menu.
* MDBDropdownToggle for toggle the drop down,
* MDBDropdownMenu for display menu after click on parent.
* **Class Component Spread and Rest**

import { Component } from "react";

class ClassCompoSpreadRest extends Component {

    render() {

        const addition = (a, ...b) => {

            console.log("called addition value of a : ", a, "value of b ", b);

        }

        addition(50, 60, 80, 90)

        addition(50, 60)

        const additionSpread = (a, b, c, d) => {

            console.log("called addition value of a : ", a, b, c, d);

        }

        additionSpread(50, 60, 80, 90)

        const numbers = [1, 3, 5, 7]

        additionSpread(...numbers)

        additionSpread(50, 60)

        return (

            <>

                <>

                    function myBio(firstName, lastName, ...otherInfo) &#123; <br />

                    &nbsp;&nbsp;&nbsp; return otherInfo;<br />

                    &#125;<br />

                // Invoke myBio function while passing five arguments to its parameters: <br />

                    myBio("Oluwatobi", "Sofela", "CodeSweetly", "Web Developer", "Male");

                    <br />

                    const myName = ["Sofela", "is", "my"];

                    <br />

                    const aboutMe = ["Oluwatobi", ...myName, "name."];

                    <br />

                    <p>About me : aboutMe</p>

                    const numbers = [1, 3, 5, 7];<br />

                    function addNumbers(a, b, c, d)&#123;<br />

                    &nbsp;&nbsp;&nbsp;return a + b + c + d;<br />

                    &#125;<br />

                    console.log(addNumbers(...numbers));

                    <br />

                    console.log(aboutMe);

                    {/\* {this.getDataSpread(arr)} \*/}

                </>

            </>

        );

    }

}

export default ClassCompoSpreadRest;

* **Controlled Component**
* Controlled component is handle by react. When we want to get the input data then we use controlled component.
* It is managed by state.
* There are four ways to set our state data and get the data from input using controlled component.

import { Component } from "react";

class Controlledcomponent extends Component {

    constructor(props) {

        super(props);

        this.state = { username: "", formdata: {} }

    }

    inputHandler = (e) => {

        console.log(e);

        console.log(e.target);

        console.log(e.target.value);

        this.setState({ username: e.target.value });

    }

    render() {

        return (<>

            {JSON.stringify(this.state)};

            <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="usernameonchangelog" className="form-control" placeholder="Print inside console" id="" onChange={(e) => console.log("called")} />

                </div>

            </div>

            <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="usernameeventhandle" id="" className="form-control" placeholder="Print using event handler" onChange={this.inputHandler} />

                </div>

            </div>

            <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="usernamesetstate" className='form-control' placeholder='Enter direct arrow func set state' onChange={(e) => { this.setState({ username: e.target.value }) }} id="username" />

                </div>

            </div>

            <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="mobile" className='form-control' placeholder='Dynamically fetch using set State' onChange={(e) => { this.setState((prevValue) => ({ formdata: { ...prevValue.formdata, [e.target.id]: e.target.value } })) }} id="mobile" /> <br />

                </div>

            </div>

            <div className="row my-3">

                <div className="col-4">

                    <input type="email" name="email" className='form-control' placeholder='Email' onChange={(e) => { this.setState((prevValue) => ({ formdata: { ...prevValue.formdata, [e.target.name]: e.target.value } })) }} id="email" />

                </div>

            </div>

        </>);

    }

}

export default Controlledcomponent;

1. Using Console log

            <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="usernameonchangelog" className="form-control" placeholder="Print inside console" id="" onChange={(e) => console.log("called")} />

                </div>

            </div>

1. Set state using event Handler

            <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="usernameeventhandle" id="" className="form-control" placeholder="Print using event handler" onChange={this.inputHandler} />

                </div>

            </div>

inputHandler = (e) => {

        console.log(e);

        console.log(e.target);

        console.log(e.target.value);

        this.setState({ username: e.target.value });

    }

1. Print set State using Arrow function Direct

            <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="usernamesetstate" className='form-control' placeholder='Enter direct arrow func set state' onChange={(e) => { this.setState({ username: e.target.value }) }} id="username" />

                </div>

            </div>

1. Dynamically save the key and value

           <div className="row my-3">

                <div className="col-4">

                    <input type="text" name="mobile" className='form-control' placeholder='Dynamically fetch using set State' onChange={(e) => { this.setState((prevValue) => ({ formdata: { ...prevValue.formdata, [e.target.id]: e.target.value } })) }} id="mobile" /> <br />

                </div>

            </div>

* Here we set the state and save the previous data using destructing and gets the id that we give and change the value according the type of input and print in {JSON.string(this,state)}.
* **Dynamic Menu create without MDB**

import React, { Component } from "react";

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

class DynamicSubMenuex extends Component {

    constructor(props) {

        super(props);

        this.state = { open: true };

    }

    inputHandle = () => {

        this.setState({ open: !this.state.open })

    }

    render() {

        const menuItems = [

            {

                title: 'Services',

                url: '/services',

                submenu: [

                    {

                        title: 'web design',

                        url: 'web-design',

                    }, {

                        title: 'web development',

                        url: 'web-dev',

                    }, {

                        title: 'SEO',

                        url: 'seo',

                    },

                ],

            },

        ];

        const Menudata = menuItems.map((data) => {

            let submenudata = data.submenu.map((submenu) => {

                return <>

                    <li><Link to={submenu.url}>{submenu.title}</Link></li>

                </>

            })

            return <>

                {/\* <ol>{submenudata}</ol> \*/}

                {this.state.open ? <>

                    <button onClick={this.inputHandle}>Services</button>

                </> : <>

                    <button onClick={this.inputHandle}>Services</button>

                    <ol>{submenudata}</ol>

                </>}

            </>

        })

        return (<>

            <ol>{Menudata}</ol>

        </>)

    }

}

export default DynamicSubMenuex;

* **Uncontrolled Component**
* Uncontrolled component is handling by JavaScript or DOM.

import React from "react";

import { Component } from "react";

import "../ClassComponent/table.css";

class Uncontrolledcompo extends Component {

    constructor(props) {

        super(props);

        this.firstName = React.createRef();

        this.lastName = React.createRef();

        this.email = React.createRef();

        this.password = React.createRef();

        this.contact = React.createRef();

    }

    submitdata = (e) => {

        e.preventDefault();

        const result = document.getElementById("result");

        const tr = document.createElement("tr");

        const td1 = document.createElement("td");

        const td1Class = td1.classList.add("table1");

        td1.append(td1Class);

        const td2 = document.createElement("td");

        const td2Class = td2.classList.add("table2");

        td2.append(td2Class);

        const td3 = document.createElement("td");

        const td3Class = td3.classList.add("table3");

        td3.append(td3Class);

        const td4 = document.createElement("td");

        const td4Class = td4.classList.add("table4");

        td4.append(td4Class);

        const td5 = document.createElement("td");

        const td5Class = td5.classList.add("table5");

        td5.append(td5Class);

        const fname = this.firstName.current.value;

        const lName = this.lastName.current.value;

        const email = this.email.current.value;

        const password = this.password.current.value;

        const contact = this.contact.current.value;

        td1.textContent = fname;

        td2.textContent = lName;

        td3.textContent = email;

        td4.textContent = password;

        td5.textContent = contact;

        tr.append(td1);

        tr.append(td2);

        tr.append(td3);

        tr.append(td4);

        tr.append(td5);

        this.firstName.current.value = "";

        this.lastName.current.value = "";

        this.email.current.value = "";

        this.password.current.value = "";

        this.contact.current.value = "";

        result.append(tr);

        // console.log("called");

        // console.log(this.username);

        // console.log(this.username.current.value);

    }

    render() {

        return (<>

            <h3>Uncontrolled Component</h3>

            <div className="row">

                <div className="col-6">

                    <form onSubmit={this.submitdata} method="post">

                        <div className="mb-3">

                            <label htmlFor="firstName" className="form-label">First Name</label>

                            <input type="text" className="form-control" ref={this.firstName} id="" aria-describedby="emailHelp" required />

                        </div>

                        <div className="mb-3">

                            <label htmlFor="lastName" className="form-label">Last Name</label>

                            <input type="text" className="form-control" ref={this.lastName} id="" aria-describedby="emailHelp" />

                        </div>

                        <div className="mb-3">

                            <label htmlFor="email" className="form-label">Email Id</label>

                            <input type="email" className="form-control" ref={this.email} id="" />

                        </div>

                        <div className="mb-3">

                            <label htmlFor="password" className="form-label">Password</label>

                            <input type="password" className="form-control" ref={this.password} id="" />

                        </div>

                        <div className="mb-3">

                            <label htmlFor="contact" className="form-label">Mobile</label>

                            <input type="text" className="form-control" ref={this.contact} id="" />

                        </div>

                        <button type="submit" className="btn btn-primary">Submit</button>

                    </form>

                    <div>

                        <table border="2" id="result">

                            <thead>

                                <tr>

                                    <th>Registration</th>

                                </tr>

                                <tr>

                                    <th className="title">First Name</th>

                                    <th className="title">Last Name</th>

                                    <th className="title">Email Id</th>

                                    <th className="title">Password</th>

                                    <th className="title">Contact</th>

                                </tr>

                            </thead>

                        </table>

                    </div>

                </div>

            </div>

        </>);

    }

}

export default Uncontrolledcompo;

* **Table.css**

.table1 {

    color: black;

    background-color: blue;

    padding: 10px;

    font-size: 1.2rem;

    font-style: italic;

}

.table2 {

    color: black;

    background-color: rgba(149, 52, 57, 0.422);

    padding: 10px;

    font-size: 1.2rem;

    font-style: italic;

}

.table3 {

    color: black;

    background-color: rgba(122, 132, 34, 0.505);

    padding: 10px;

    font-size: 1.2rem;

    font-style: italic;

}

.table4 {

    color: black;

    background-color: rgba(171, 50, 177, 0.388);

    padding: 10px;

    font-size: 1.2rem;

    font-style: italic;

}

.table5 {

    color: black;

    background-color: rgba(35, 132, 130, 0.352);

    padding: 10px;

    font-size: 1.2rem;

    font-style: italic;

}

.title {

    padding: 20px;

    text-align: center;

}

* First we use createRef for take a reference value of input.
* Form onSubmit event we call the function we create a function and give preventDefault for stop refreshing the function.
* Inside input tag we give ref{this.variableName} of value to take the value.
* When submit the form table is created automatically using id and createElement we create tr and td. Number of td is created depended upon the column.
* Then we assign the current reference value to each td using text Content.
* Then append all td values to tr and after appending for new row creation we append tr to result (table Id).
* **Composition and Inheritance**
* Normally we pass the data using props it is working but when we want to pass the child element data it is not possible using single props so we want to pass all child data then we use {this.props.children}.
* **CompositionandInheritance.jsx**

import { Component } from "react";

import ChildCompoInheritance from "./ChildCompoInheritance";

class CompositionandInheritance extends Component {

    constructor(props) {

        super(props);

        this.state = {

            ParentCompoData: "Something from ParentCompoData"

        }

    }

    render() {

        return (<>

            <ChildCompoInheritance message="Message from Parent" title={this.state.ParentCompoData} imgMsg="This is a image that is passed by props" image="/ReactJS.png">

                <h1 className="Dialog-title">

                    Welcome to the world of React.

                </h1>

                <p className="Dialog-message">

                    This is a child data of ChildCompoInheritance.

                </p>

            </ChildCompoInheritance>

        </>);

    }

}

export default CompositionandInheritance;

* **ChildCompoInheritance.jsx**

import { Component } from "react";

class ChildCompoInheritance extends Component {

    render() {

        return (<>

            <div>

                <h1>

                    {this.props.title}

                </h1>

                {this.props.message}

                <h3>{this.props.imgMsg}</h3>

                <img src={this.props.image} alt="" />

                {this.props.children}

            </div>

        </>);

    }

}

export default ChildCompoInheritance;

* **State Lifting**
* Suppose when we want to pass the data from child component to parent component then we use State lifting.
* **StateLifting.jsx**

import { Component } from "react";

import Secondcompo from "./Secondcompo";

class StateLifting extends Component {

    constructor(props) {

        super(props);

        this.state = {

            data: "default"

        }

    }

    inputChange = (fromChild) => {

        console.log("called", fromChild);

        this.setState({ data: fromChild })

    }

    render() {

        return (<>

            <div className="row p-3">

                <div className="col">

                    <Secondcompo parentMethod={this.inputChange} />

                    <p>in parent compo :{this.state.data}</p>

                </div>

            </div>

        </>);

    }

}

export default StateLifting;

* **SecondCompo.jsx**

import { Component } from "react";

class Secondcompo extends Component {

    inputInsideChildCompo = (e) => {

        console.log("called inputInsideChildCompo", e);

        console.log("called inputInsideChildCompo", e.target.value);

        this.props.parentMethod(e.target.value)

    }

    render() {

        return (<>

            <input type="text" onChange={this.inputInsideChildCompo} />

        </>);

    }

}

export default Secondcompo;

* First we define the state.
* In parent component we load child component and we create attribute and we define function.
* Then in child component create one input text inside textbox pass the onChange event we create arrow function inside child component and receive the state data.
* Now inside the parent component we receive the data as a parameter and update the data using setState whenever we change the input our data will also change inside the p tag.
* **Higher Order Component**
* **MainCompo.jsx**

import { Component } from "react";

import Enhancedcompo from "./Enhancedcompo";

class Maincompo extends Component {

    render() {

        console.log("Main Compo called");

        const { clickHandle } = this.props;

        return (<>

            <button onClick={clickHandle}>Btn Click Main Compo</button>

        </>);

    }

}

export default Enhancedcompo(Maincompo);

* **Enhancedcompo.jsx**

import { Component } from "react";

const Enhancedcompo = (OriginalComponent) => {

    class NewComponent extends Component {

        clickHandle = () => {

            console.log("called inside enhanced");

        }

        render() {

            console.log("Enhanced Compo called");

            return <>

                <OriginalComponent clickHandle={this.clickHandle} />

            </>

        }

    }

    return NewComponent;

}

export default Enhancedcompo;

* When we pass the function from one component to other component then use Higher Order Component.
* First create a button onclick event we call the function.
* Destructing of a function that is in Main compo file.
* Import the Enhanced compo file and wrap the Maincompo file.
* export default Enhancedcompo(Maincompo) also known as custom hook because we merge two components.
* Enhanced component is created with class compo and entire class is will be wrapped by function.
* Pass the argument in Enhanced component function and create a class.
* Inside the class load the argument as a component then call the same function which is call from the Maincompo file.
* Inside the class create a function and after create the function whatever action you want to perform.
* Then return the class that is created inside the Enhanced component.
* **Loan calculator using controlled component**

import { Component } from "react";

class UncontrolledTask extends Component {

    constructor(props) {

        super(props);

        this.state = {

            amount: 0,

            roi: 0,

            year: 0,

            emi: 0,

            totalInterest: 0,

            totalAmount: 0,

        }

    }

    //EMI = [P x R x (1+R)^N]/[(1+R)^N-1]

    calculationLoan = () => {

        const { amount, roi, year } = this.state;

        const monthlyInterestrate = (roi / 12) / 100;

        const numberofPayments = year \* 12;

        const emi = (amount \* monthlyInterestrate \* Math.pow(1 + monthlyInterestrate, numberofPayments)) / (Math.pow(1 + monthlyInterestrate, numberofPayments) - 1);

        const total = emi \* numberofPayments;

        const totalInterest = total - amount;

        const totalAmount = amount + totalInterest;

        this.setState({ emi })

        this.setState({ totalInterest })

        this.setState({ totalAmount })

    }

    render() {

        return (<>

            {JSON.stringify(this.state)};

            <div className="mb-3">

                <label for="amount" className="form-label">Amount</label><br />

                <input type="text" className="w-10" id="amount" onChange={(e) => { this.setState({ amount: e.target.value }, this.calculationLoan) }} />

            </div>

            <div className="mb-3">

                <label for="roi" className="form-label">Rate of Interest</label><br />

                <input type="text" className="w-10" id="roi" onChange={(e) => { this.setState({ roi: e.target.value }, this.calculationLoan) }} />

            </div>

            <div className="mb-3">

                <label for="year" className="form-label">Years</label><br />

                <input type="text" className="w-10" id="year" onChange={(e) => { this.setState({ year: e.target.value }, this.calculationLoan) }} />

            </div>

            <p>EMI: {this.state.emi.toFixed(2)}</p>

            <p>Total Interest: {this.state.totalInterest}</p>

            <p>Total Amount: {this.state.totalAmount}</p>

        </>);

    }

}

export default UncontrolledTask;

* **HOC API Task**
* Main compo is same as above.
* **Enhancedcompo.jsx**

import { Component } from "react";

const Enhancedcompo = (OriginalComponent) => {

    class NewComponent extends Component {

        clickHandle = () => {

            fetch('https://fakestoreapi.com/products/category/jewelery')

                .then((response) => response.json())

                .then((result) => {

                    let data = result;

                    let div = document.getElementById("div");

                    data.map((e) => {

                        const id = `${e.id}`;

                        console.log(id);

                        const image = `${e.image}`;

                        console.log(image);

                        const p = document.createElement("p");

                        p.textContent = id;

                        div.append(p);

                        console.log(div);

                        console.log(p);

                        const img = document.createElement("img");

                        img.setAttribute("src", image)

                        div.append(img)

                        console.log(img);

                        return data;

                    })

                })

        }

        render() {

            console.log("Enhanced Compo called");

            return <>

                <OriginalComponent clickHandle={this.clickHandle} />

                <div id="div"></div>

            </>

        }

    }

    return NewComponent;

}

export default Enhancedcompo;

* **Functional Component**
* Functional component is just a simple JavaScript function; it accepts the data in the form of props and returns the react element.
* When we want to receive the data we use props as an argument and receive the data.
* Function component provides some advanced features hooks.
* **Hook**
* Hooks are the new additional features in React.
* When we saw use it means hooks.
* **Functional Component State**
* When we use state in function component that is the hook available called useState.
* **useState**
* useState is React Hook that allows you to add state to a function component. It returns an array with two values: the current state and a function to update it. The Hook takes an initial state value as an argument and returns an update state value whenever the setter function is called.
* Here setter function are onClick, onChange, onFocus, onLoad, onChange.

import React, { useState } from "react";

import ReactJS from "./ReactJS.png"

import {

    MDBCard,

    MDBCardBody,

    MDBCardTitle,

    MDBCardText,

    MDBCardImage,

    MDBBtn

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

const Functionompostate = () => {

    const [Data, setData] = useState("Default");

    const [count, setCount] = useState(0);

    const [info, setInfo] = useState(false);

    const [show, setShow] = useState(false);

    let variable = "Data";

    const Handle = () => {

        console.log("called");

        variable = "test";

        setData(count % 2 === 0 ? "odd" : "even");

        setCount(count - 1);

        setInfo((current) => !current);

    }

    const card = () => {

        setShow((visible) => !visible);

    }

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <p>Simple variable : {variable}</p>

                <p>Variable threaw state : {Data}</p>

                <p>{count}</p>

                <p>{info ? "New value" : "Old value"}</p>

                <button className="btn btn-primary" onClick={Handle}>Even and Odd</button>

                <button className="btn btn-primary" onClick={() => { setCount(count + 1); setData(count % 2 === 0 ? "odd" : "even") }}>

                    Click </button>

                <button className="btn btn-primary" onClick={card}>Card</button>

                <div className="row">

                    <div className="col-6">

                        {show ? <>

                            <MDBCard>

                                <MDBCardImage src={ReactJS} position='top' alt='React JS' />

                                <MDBCardBody>

                                    <MDBCardTitle className="text-center">React JS</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>

                        </> : ""}

                    </div>

                </div>

            </div>

        </div>

    </>);

}

export default Functionompostate;

* **Functional Component Props**
* When we want pass the data from parent component to child component then we use props.
* We load child component inside parent component and gave attribute name.
* Normally we have use props to pass the data from parent component to child component.
* In function component it is same but when we want to receive the data into child component we pass the parameter as props and receive the data.
* **Functioncompoprops.jsx**

import React from "react";

import PropsChild from './05Propschild.jsx';

import ReactJS from './ReactJS.png';

const FunctionCompoProps = () => {

    return (<>

        <div className="row">

            <div className="col-3">

                <PropsChild title="React js Live Url image" image="LiveUrl" />

            </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;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 FunctionCompoProps;

* **PropsChild.jsx**

import {

    MDBCard,

    MDBCardBody,

    MDBCardTitle,

    MDBCardText,

    MDBCardImage,

    MDBBtn

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

const PropsChild = (props) => {

    return (<>

        <MDBCard>

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

            <MDBCardBody>

                <MDBCardTitle className="text-center">{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;

* **useEffect**
* When we want to display the data after component render then we use useEffect hook.
* useEffect works asynchronously.

import React, { useEffect, useState } from "react";

const FunctioncompouseEffect = () => {

    const [api, setApi] = useState(false);

    const [api2, setApi2] = useState(false)

    useEffect(() => {

        fetch("https://jsonplaceholder.typicode.com/albums").then((res) => res.json()).then((response) => { console.log(response); })

    }, [])

    useEffect(() => {

        fetch("https://jsonplaceholder.typicode.com/todos").then((res) => res.json()).then((response) => { console.log(response); })

    }, [api])

    useEffect(() => {

        fetch("https://jsonplaceholder.typicode.com/posts").then((res) => res.json()).then((response) => { console.log(response); })

        return (() => {

            console.log("unmount called");

        })

    }, [api2])

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <button onClick={() => { setApi(!api) }}>Click</button>

                <button onClick={() => { setApi2(!api2) }}>Click</button>

            </div>

        </div>

    </>);

}

export default FunctioncompouseEffect;

* There are two ways to get the data after render the page.
* First is to give [] dependency it means when our component is render then only our data is displayed. It is same like a constructor.
* Second give the dependency of state [api] and setState on button click event once the button is click then the data is show otherwise when our component render or re rendered then our data is show at that time but once the button is click no of times the data is show.
* useEffect also display the data when our component is closed we write the data on return statement inside useEffect once the component is closed the data is display it is also called the clean up.

* **useLayoutEffect**
* useLayoutEffect is used when we want to print some data when component is render before it means it works synchronously. The data is always display first.
* The [useLayoutEffect](https://www.geeksforgeeks.org/reactjs-uselayouteffect-hook/) works similarly to useEffect but rather working asynchronously like useEffect hook, it fires synchronously after all DOM loading is done loading. This is useful for synchronously re-rendering the DOM and also to read the layout from the DOM. But to prevent blocking the page loading, we should always use useEffect hook.

import React, { useEffect } from "react";

import { useLayoutEffect } from "react";

const FunctionCompoUseEffectUseLayoutEffect = () => {

    useEffect(() => {

        // getData();

        async function data() {

            await fetch("https://jsonplaceholder.typicode.com/posts").then((res) => res.json()).then((response) => { console.log(response); })

        }

        data();

    })

    // async function getData(){

    //     let Alldata = await fetch("https://jsonplaceholder.typicode.com/posts").then((res)=>res.json()).then((response)=>{console.log(response);})

    //     return Alldata;

    // }

    useLayoutEffect(() => {

        console.log("called useLayoutEffect");

        fetch("https://jsonplaceholder.typicode.com/todos/").then((res) => res.json()).then((response) => { console.log(response); })

        return (() => {

            console.log("called");

            // fetch("https://jsonplaceholder.typicode.com/todos/").then((res) => res.json()).then((response) => { console.log(response); })

        })

    }, [])

    // useLayoutEffect(async () => {

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

    //     // Warning: useLayoutEffect must not return anything besides a function, which is used for clean-up.

    //     // It looks like you wrote useLayoutEffect(async () => ...) or returned a Promise. Instead, write the async function inside your effect and call it immediately:

    //     await fetch("https://jsonplaceholder.typicode.com/todos/").then((res) => res.json()).then((response) => { console.log(response); })

    //     return (() => {

    //         console.log("called");

    //         // fetch("https://jsonplaceholder.typicode.com/todos/").then((res) => res.json()).then((response) => { console.log(response); })

    //     })

    // }, [])

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h3>difference bten useEffect vs useLayoutEffect</h3>

                <p>useLayoutEffect is used when we want to print some data when component is render before it means it works synchronously. The data is always display first.</p>

                <p>The useLayoutEffect works similarly to useEffect but rather working asynchronously like useEffect hook, it fires synchronously after all DOM loading is done loading. This is useful for synchronously re-rendering the DOM and also to read the layout from the DOM. But to prevent blocking the page loading, we should always use useEffect hook.</p>

            </div>

        </div>

    </>);

}

export default FunctionCompoUseEffectUseLayoutEffect;

* Here use fetch api in both the useEffect and useLayoutEffect first display useLayoutEffect Api data then display useEffect API Data.
* In useEffect we use async await function but it is not use in useLayoutEffect when we want to use it gives the warning because useLayoutEffect works synchronously async await returns the promise.

// Warning: useLayoutEffect must not return anything besides a function, which is used for clean-up.

    //     // It looks like you wrote useLayoutEffect(async () => ...) or returned a Promise. Instead, write the async function inside your effect and call it immediately:

* **Use Memo**
* Use memo returns the memorized value. When give dependency it always works in specific dependency.
* In general terms when we have multiple of data in Array or something else when data is count always it starts from first index or value, but when we use useMemo hook it stores the previous value data and accordingly the calculation or task is perform. So basically if we imporove our code performance and if we have bunch of data at that time we use useMemo.
* **Functioncompousememo.jsx**

import React, { useMemo, useState } from "react";

const Functioncompousememo = () => {

    const [number, setNumber] = useState(0)

    // Using useMemo

    const squaredNum = useMemo(() => {

        return squareNum(number);

    }, [number])

    const [counter, setCounter] = useState(0);

    // Change the state to the input

    const onChangeHandler = (e) => {

        setNumber(e.target.value);

    }

    // Increases the counter by 1

    const counterHander = () => {

        setCounter(counter + 1);

    }

    // const [count, setCount] = useState(0);

    // const [item, setItem] = useState(0);

    // const [text, setText] = useState("sachin");

    // const Usememo = useMemo(function multipleCount() {

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

    //     return count \* 2

    // }, [count])

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <input type="number" placeholder="Enter a number"

                    value={number} onChange={onChangeHandler}>

                </input>

                <div>OUTPUT: {squaredNum}</div>

                <button onClick={counterHander}>Counter ++</button>

                <div>Counter : {counter}</div>

                {/\* <h2>Usememo</h2>

                <h2>{count}</h2>

                <h2>{item}</h2>

                <h2>{text}</h2>

                <h2>{Usememo}</h2>

                <button onClick={() => setCount(count + 1)}>Update Count</button>

                <button onClick={() => setItem(item + 5)}>Update Item</button>

                <button onClick={() => setText("Virat")}>Update Text</button> \*/}

            </div>

        </div>

    </>);

    function squareNum(number) {

        console.log("Squaring will be done!");

        return Math.pow(number, 2);

    }

}

export default Functioncompousememo;

* First I create a state number. Then I create a function it performs square of number.
* Create another onChangehandler function and get change value of state.
* Then I create input onChange event I pass onChangehandler function and pass the current value as well.
* I create usememo and inside useMemo return the squareNum function and give dependency when number is change on input it print log value it continuously return memorized value onChange event.
* I create second state counter on button click event setCounter increment by one. When button click the log value is not display on console but write on Input the log value is display.
* **Word count using Use Memo**

import React, { useState } from "react";

const WordCountusingUseMemo = () => {

    const [counter, setCounter] = useState(0);

    const [wordIndex, setwordIndex] = useState(0);

    const words = ["Hi", "I", "am", "Dishank"];

    const word = words[wordIndex];

    const computeletterCount = (word) => {

        // let i = 0;

        // while (i < 10000) i++;

        console.log("ComputeLetterCount");

        return word.length;

    }

    // const letterCount = useMemo(() => { computeletterCount(word) });

    // console.log(letterCount);

    // const letterCount = (() => computeletterCount(word))

    const letterCount = computeletterCount(word);

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <p>useMemo is a React Hook that lets you cache the result of a calculation between re-renders</p>

                {/\* <p>{words[0]}</p>

                <p>{words[1]}</p>

                <p>{words[2]}</p>

                <p>{words[3]}</p> \*/}

                <p>words = ["Hi","I","am","Dishank"]</p>

                <p>Words Array Element of {wordIndex} index: {word}</p>

                <p>

                    "{word}" has {letterCount} letters

                </p>

                State for word count: {wordIndex}

                <button onClick={() => {

                    if (wordIndex + 1 === words.length) {

                        setwordIndex(0);

                    }

                    else {

                        setwordIndex(wordIndex + 1);

                    }

                }}>Click Next Word Count</button>

                &nbsp; <label htmlFor="">{counter}</label>&nbsp;

                <button className='btn btn-danger' onClick={() => { setCounter(counter + 1) }}>+</button>

            </div>

        </div>

    </>);

}

export default WordCountusingUseMemo;

* **Use callback**
* Use callback always returns a memorized function.
* For example we have multiple functions and each function have some different functionality perform so when we want to perform some functionality on some click event into single component then we use useCallback hook.

import React, { useState, useCallback } from "react";

const funccount = new Set();

const UseCallMultifunctiondependency = () => {

    const [count, setCount] = useState(0);

    const [number, setNumber] = useState(0);

    // const incrementCounter = () => {

    //     setCount(count + 1);

    // };

    // const decrementCounter = () => {

    //     setCount(count - 1);

    // };

    // const incrementNumber = () => {

    //     setNumber(number + 1);

    // };

    const incrementCounter = useCallback(() => {

        setCount(count + 1);

    }, [count]);

    const decrementCounter = useCallback(() => {

        setCount(count - 1);

    }, [count]);

    const incrementNumber = useCallback(() => {

        setNumber(number + 1);

    }, [number]);

    funccount.add(incrementCounter);

    funccount.add(decrementCounter);

    funccount.add(incrementNumber);

    console.log(funccount);

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                Count: {count} <br />

                Number: {number} <br />

                <button onClick={incrementCounter}>Increase counter</button>

                <button onClick={decrementCounter}>Decrease Counter</button>

                <button onClick={incrementNumber}>increase number</button>

            </div>

        </div>

    </>);

}

export default UseCallMultifunctiondependency;

* First create a Set. Set objects are collections of values. A value in the set **may only occur once**; it is unique in the set's collection.
* Create two states Increment, Decrement and Number Increment. I give two dependencies at same for increment and decrement counter and I give number increment second dependency.
* Add three functions into Set using add method and I console.log my Set without write useMemo once click on any button it calls three time but using useMemo it calls depend upon the dependency whatever we give.
* **Use Context**
* Use Context provide three functionalities

1. Create context
   * + Create context is used to create a function.
2. Provider
   * + When create a function in using use Context the provider helps to pass the data from one component to another component.
3. Consumer or useContext
   * + In old react version if we want to receive the data use consumer but in latest version we use hook called use Context it is used to receive the data from whenever our context is created.

* The main difference between props and useContext is that props is a react functionality to receive the data but useContext is a by default hook it is used to receive the data whenever the create context is created.
* **FunctionuseContext.jsx**

import { createContext, useState } from "react";

import ChildCompo from "./ChildCompo";

import Login from "./Login";

import { useContext } from "react";

export const AppContext = createContext(null);

export const AppContext2 = createContext(null);

const ThemeContext = createContext();

const FunctionuseContext = () => {

    const [userName, setUserName] = useState("");

    const [loginid, setLoginid] = useState("");

    const [password, setPassword] = useState("");

    const [theme, setTheme] = useState("");

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <AppContext.Provider value={{ userName, setUserName }}>

                    <ChildCompo />

                </AppContext.Provider>

                <AppContext2.Provider value={{ loginid, setLoginid, password, setPassword }}>

                    <Login />

                </AppContext2.Provider><br />

                <ThemeContext.Provider value={{ theme, setTheme }}>

                    Use context Parent Data

                    <Formcompo />

                </ThemeContext.Provider>

            </div>

        </div>

    </>);

}

function Formcompo() {

    return (<>

        Child data

        <PanelForDisp>

        </PanelForDisp>

    </>)

}

function PanelForDisp() {

    const { theme, setTheme } = useContext(ThemeContext);

    const customClass = "panel-" + theme;

    return (<>

        Grand child Data

        <div>

            {customClass} <br />

            <button onClick={() => setTheme("dark")}>Signup</button>

            <button onClick={() => setTheme("light")}>Signin</button>

        </div>

    </>)

}

export default FunctionuseContext;

* **ChildCompo.jsx**

import { useContext } from "react";

import { AppContext } from "./01FunctionuseContext";

const ChildCompo = () => {

    const {userName, setUserName} = useContext(AppContext);

    return (<>

        <h1>User Compo:{userName}</h1>

        <input type="text" name="" onChange={(event) => {

            setUserName(event.target.value)

        }} id="" />

    </>);

}

export default ChildCompo;

* **Login.jsx**

import { useContext } from "react";

import { AppContext2 } from "./01FunctionuseContext";

const Login = () => {

    const { loginid, setLoginid } = useContext(AppContext2);

    const { password, setPassword } = useContext(AppContext2);

    return (<>

        <h1>Login Id :{loginid}</h1>

        <input type="text" name="" onChange={(event) => {

            setLoginid(event.target.value);

        }} id="" />

        <h1>Password : {password}</h1>

        <input type="password" name="" onChange={(event) => {

            setPassword(event.target.value)

        }} id="" />

    </>);

}

export default Login;

* **Use Reducer**
* Use reducer is used when we have perform multiple operation into single function then we use Use reducer.

import React, { useReducer } from "react";

const FunctionCompoUseReducer = () => {

    const initialState = 0;

    const reducer = (state, action) => { //State is takes intialState value and dispatch it takes a object

        switch (action) {

            case "add":

                return state + 1;

            case "subtract":

                return state - 1;

            case "reset":

                return 0;

            default:

                throw new Error("Unexpected action");

        }

    }

    const [count, dispatch] = useReducer(reducer, initialState); //reducer function and initialState value that define //

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h2>{count}</h2>

                <button onClick={() => dispatch("add")}>

                    add

                </button>

                <button onClick={() => dispatch("subtract")}>

                    subtract

                </button>

                <button onClick={() => dispatch("reset")}>

                    reset

                </button>

            </div>

        </div>

    </>);

}

export default FunctionCompoUseReducer;

* First define initial value that is 0. Then create a function inside function I have pass the current state and action that we want to perform.
* Then create a usereducer hook first define the couth state and dispatch (setState) that we want to check. Inside usereducer we pass reducer function and initial state value.
* On button click event check the dispatch(setState) action that we want to perform.
* **Use Imparative Handle**
* useImperativeHandle is a React Hook that lets you customize the handle exposed as a ref.
* In React, data is passed from parent to child components via props, in what is known as unidirectional data flow. The parent component cannot directly call a function defined in the child component or reach down to grab a value for itself.
* In certain circumstances, we want our parent component to reach down to the child component, getting data that originates in the child component for its own use. We can achieve this type of data flow with the useImperativeHandle Hook, which allows us to expose a value, state, or function inside a child component to the parent component through ref. You can also decide which properties the parent component can access, thereby maintaining the private scoping of the child component.
* **Useimparative.jsx**

import { useRef } from "react";

import ButtonCompo from "./ButtonCompo";

const UseImparative = () => {

    const buttonRef = useRef(null);

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <button onClick={() => { buttonRef.current.toggle() }}>Click for toggle</button>

                <ButtonCompo ref={buttonRef} data="this will contains props data" />

                <p>useImperativeHandle is a React Hook that lets you customize the handle exposed as a ref.</p>

                <p>In React, data is passed from parent to child components via props, in what is known as unidirectional data flow. The parent component cannot directly call a function defined in the child component or reach down to grab a value for itself.</p>

                <p> In certain circumstances, we want our parent component to reach down to the child component, getting data that originates in the child component for its own use. We can achieve this type of data flow with the useImperativeHandle Hook, which allows us to expose a value, state, or function inside a child component to the parent component through ref. You can also decide which properties the parent component can access, thereby maintaining the private scoping of the child component.</p>

            </div>

        </div>

    </>);

}

export default UseImparative;

* **Buttoncompo.jsx**

import React, { forwardRef, useImperativeHandle, useState } from 'react';

const ButtonCompo = forwardRef((props, ref) => {

    const [toggle, setToggle] = useState(false);

    useImperativeHandle(ref, () => ({

        toggle() {

            console.log("called inside child compo btn");

            setToggle(!toggle)

        },

    }));

    return (

        <>

            <p>Props : {props.data}</p>

            <p>this will contains child data : {toggle && <span>Toggle</span>} </p>

            {/\* <button onClick={()=>{

                console.log("anything");

            }}>Button From Child</button> \*/}

        </>  );

});

export default ButtonCompo;

* This hook is used when we want to call the function from child to parent then we use useImparative.
* **Use Ref**
* Use ref hook is used to get the reference value that is not needed for rendering.

import { useRef } from "react";

const Useref = () => {

    const inputRef = useRef(null);

    const onClick = () => {

        inputRef.current.value = "";

    }

    return (<>

        <div className="row">

            <div className="col-6 offset-6">

                <h2>Dishank</h2>

                <input type="text" name="" ref={inputRef} id="" />

                <button onClick={onClick}>Click</button>

            </div>

        </div>

    </>);

}

export default Useref;