1. Hooks are the new features addition in react version 16.8.
2. They allow you to use react features without having to write a class.
3. Avoid the Whole confusion with “this” keyword.
4. Allow you to reuse stateful logic.
5. Organize the logic inside a component into reusable insolated units.

--------------------------------------------------------------------------------------------------

**Class Components**

import React, { Component } from 'react'

export class ClassCounter extends Component {

    constructor(props) {

      super(props)

     this.state = {

        count : 0

      }

    }

    IncrementCounter = () => {

        this.setState({  // ensures that the component has been updated and calls for re-rendering of the component. setState is asynchronous call means if synchronous call get called it may not get updated at right time like to know current value of object after update using setState it may not get give current updated value on console.

            count : this.state.count + 1

        })

       }

  render() {

    return (

      <div>

        <button onClick={this.IncrementCounter}>Count {this.state.count}</button>

      </div>

    )

  }

}

export default ClassCounter

This is the class component for increasing the counter.

---------------------------------------------------------------------------------------------------------------------

**Function Component**

import React, { useState } from 'react'

function HooksCounter() {

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

  return (

    <div>

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

    </div>

  )

}

export default HooksCounter

here we are using UseState in order to get the counter.

**useState :** The React useState Hook allows us to track state in a function component.

State generally refers to data or properties that need to be tracking in an application.

**Rules Of Hooks:**

1. Only call hooks on the top level.
2. Don’t Call hooks inside loops, Conditions or nested Functions.
3. Only call hooks from the react function.
4. Call them from within the react functional components and not just any regular javascript functions.

-----------------------------------------------------------------------------------------------------------

**HooksCounter2.js**

import React, { useState } from 'react'

function HooksCounter2() {

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

  return (

    <div>

      <h2>Count {counter}</h2>

      <div>

        <button style={{margin: "10px"}} onClick={()=> setCounter(0)}>Reset</button>

        <button style={{margin: "10px"}} onClick={() => setCounter(counter + 1)}>Increment</button>

        <button style={{margin: "10px"}} onClick={() => setCounter(counter - 1)}>Decrement</button>

      </div>

    </div>

  )

}

export default HooksCounter2

Here we are using 3 buttons reset, increment and decrement.

-------------------------------------------------------------------------------------------------------------------------

State Variable can be any datatype, It can be object, interger, string, Boolean and array.

import React, { useState } from 'react'

function HooksCounter3() {

    const [name, setName] = useState({

        firstName : "",

        lastName : ""

    })

  return (

    <div>

        <div>

            <input type="text"

            value={name.firstName}

            onChange={e => setName({...name , firstName: e.target.value })}/>

            <input type="text"

            value={name.lastName}

            onChange={e => setName({...name , lastName: e.target.value })}/>

        </div>

        <div>

            <h2>

                First Name : {name.firstName}

            </h2>

            <h2>

                Last Name : {name.lastName}

            </h2>

            <h2>

                {JSON.stringify(name)}

            </h2>

        </div>

    </div>  //JavaScript allows us to listen to an input’s change in value by providing the attribute onchange. React’s version of the onchange event handler is the same, but camel-cased.

    //We can access the target input’s value inside of the handleChange by accessing e.target.value. Therefore, to log the name of the input field, we can log e.target.name.

    //The JSON.stringify() static method converts a JavaScript value to a JSON string, optionally replacing values if a replacer function is specified or optionally including only the specified properties if a replacer array is specified.

  )

}

export default HooksCounter3

**HooksCounter4.js (ARRAY MAPPING)**

import React, { useState } from 'react'

function HooksCounter4() {

    const [items, setItems] = useState([])

    const addItems = () => {

        setItems([... items, {

            id : items.length,

            value : Math.floor(Math.random() \* 10) + 1 //using math module to get random number in to array.

        }])

    }

    return (

    <div>

      <button onClick={addItems} id={items.id}>Add a number</button>

      <ul>

        {

            items.map(item =>{

                return (

                    <li key={item.id}>{item.value}</li>

                )

            }

            )

        }

      </ul>

    </div>

  )

}

export default HooksCounter4

1. The useState hooks lets you add state to functional component.
2. In classes, the state is always an object.
3. With the useState hook, the state doesn’t have to be an object.
4. The useState hook returns as array with 2 elements.
5. The first element is the current value of the state, and the second element is a state setter functions.
6. New State value depends on the previous state value? You can pass a function to the setter function.
7. When dealing with object or array, always make sure to spread your state variable and then call the setter function.

**USEEFFECT Hook**

The Effect hook let you perform side effect in functional components.

It is a close replacement for **componentDidMount, componentDidUpdate** and **componentWillUnmount.**

**ClassCounteruseEffect.js**

import React, { Component } from 'react'

export default class ClassCounteruseEffect extends Component {

    constructor(props) {

      super(props)

      this.state = {

         count : 0

      }

    }

    componentDidMount(){

        document.title = `Clicked ${this.state.count} Times`

    }

    componentDidUpdate(){

        document.title = `Clicked ${this.state.count} Times`

    }

  render() {

    return (

      <div>

        <button onClick={()=>

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

        }>You Clicked {this.state.count} Times</button>

      </div>

    )

  }

}

Above is the class code for useEffect in class component…..

**HooksCounteruseEffect.js**

import React, { useEffect, useState } from 'react'

function HooksCounteruseEffect() {

    const [counter , setCounter] = useState(0)

    useEffect(()=>{

        document.title = `You Clicked ${counter} times`

    })

  return (

    <div>

      <button onClick={() => setCounter(counter + 1)}>You Clicked {counter}</button>

    </div>

  )

}

export default HooksCounteruseEffect

Note: useEffect is called after every single render. But in some cases useEffect applying after every single render might create a performance problem. To overcome this problem we use conditionally run effect.

**CONDITIONAL RUN EFFECT RENDERING**

**ClassCounteruseEffectCdn.js**

import React, { Component } from 'react'

export default class ClassCounteruseEffectCdn extends Component {

    constructor(props) {

      super(props)

      this.state = {

         count : 0 , //Variable count initialize to 0.

         name : ""

      }

    }

    componentDidMount(){

        document.title = `Clicked ${this.state.count} Times`

    }

    componentDidUpdate(prevProps, prevState){

        if (prevState.count !== this.state.count) {

            console.log("Updating the document...");

            document.title = `Clicked ${this.state.count} Times`

        }

    }

  render() {

    return (

      <div>

        <input type="text" value={this.state.name} onChange={e => {

            this.setState({name : e.target.value})

        }} />

        <button onClick={()=>

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

        }>You Clicked {this.state.count} Times</button>

      </div>

    )

  }

}

Above is the component for class Counter useEffect for Conditional Rendering Run Effect.

**HooksCounteruseEffectCdn.js**

import React, { useEffect, useState } from 'react'

function HooksCounteruseEffectCdn() {

    const [counter , setCounter] = useState(0)

    const [name , setName] = useState("")

    useEffect((()=>{

        console.log("Document is Getting Updated..........")

        document.title = `You Clicked ${counter} times`

    }),[counter])

    // For Conditionally executiong an effect we are passing second parameter as array, In this Array we need specify either props or state that we need to watch for, Only if those props and state that specify in this array where to change, the effect will be excecuted.

    // For our example we only need effect to be changed when the counter changes its state or we can say value here.

  return (

    <div>

        <input type="text" value={name} onChange={e => setName(e.target.value)} />

      <button onClick={() => setCounter(counter + 1)}>You Clicked {counter}</button>

    </div>

  )

}

export default HooksCounteruseEffectCdn

Above is the function component that shows the useEffect in for some Conditions  
-----------------------------------------------------------------------------------------------------------

**RUN EFFECT ONLY ONCE**

**ClassMouse.js**

import React, { Component } from 'react'

export default class ClassMouse extends Component {

    constructor(props) {

      super(props)

      this.state = {

         x: 0,

         y: 0

      }

    }

    logMousePosition = e => {

        this.setState({x: e.clientX , y: e.clientY})

    }

    componentDidMount() {

        window.addEventListener("mousemove" , this.logMousePosition)

    }

  render() {

    return (

      <div>

        <div style={{fontSize : "50px" , fontWeight : "bold"}}>

            X - {this.state.x}   Y - {this.state.y}

        </div>

      </div>

    )

  }

}

**Here we saw How to make a component using componentDidMount.**

**HooksMouse.js**

import React, { useEffect, useState } from 'react'

function HooksMouse() {

    const [x , setX] = useState(0);

    const [y , setY] = useState(0);

    const logMouseMovement = e => {

        console.log('Mouse Event id called.........')

        setX(e.clientX);

        setY(e.clientY);

    }

    useEffect((() => {

        console.log("UseEffect Called");

        window.addEventListener("mousemove" , logMouseMovement);

    }), [])

    // To Only call useEffect Once we define the second parameter in the useEffect function.

  return (

    <div>

      <div>

        <h2>

            X : {x}     Y : {y}

        </h2>

      </div>

    </div>

  )

}

export default HooksMouse

**USEEFFECT WITH CLEANUP.**

Here We are gonna see how we are gonna use componentWillUnmount.

Lets Check why we need componentWillUnmount.

**HooksMouse2.js :-**  This file contains the changes made in HooksMouse.js file and added the componentWillUnmount code in it.

import React, { useEffect, useState } from 'react'

function HooksMouse2() {

    const [x , setX] = useState(0);

    const [y , setY] = useState(0);

    const logMouseMovement = e => {

        console.log('Mouse Event id called.........')

        setX(e.clientX);

        setY(e.clientY);

    }

    useEffect((() => {

        console.log("UseEffect Called");

        window.addEventListener("mousemove" , logMouseMovement);

        return () => {

            console.log("Component is getting unmounted");

            window.removeEventListener("mousemove" , logMouseMovement);

        }

    }), [])

    // To Only call useEffect Once we define the second parameter in the useEffect function.

  return (

    <div>

      <div>

        <h2>

            X : {x}     Y : {y}

        </h2>

      </div>

    </div>

  )

}

export default HooksMouse2

**MouseContainer.js**

import React, {useState} from 'react'

import HooksMouse2 from './HooksMouse2'

function MouseContainer() {

    const [display , setDisplay] = useState(true)

  return (

    <div>

      <button onClick={() => setDisplay(!display)}>Toggle Display</button>

      {display && <HooksMouse2></HooksMouse2> }

    </div>

    // display && HookMouse2 line explains if the display is true make the hookmouse2 file visible but if the diaplay is false we can't see the hookmouse2 file.

    // Above code we got the hooksmouse output and it is displaying and undisplaying correctly but in the console when the display is false in concole we are getting the error and after error the log statement is getting executed again.

    // And to overcome this problem we use componentWillUnmount method in class component but in the functional component and useEffect hook we just write the componentWillUnmount the code in return statment in useEffect Function.

    // The Changes are made in HooksMouse2.js

  )

}

export default MouseContainer

The Above is the cleanup code.

**USEEFFECT WITH INCORRECT DEPENDENCIES>:**

**ClassCounterInterval.js**

import React, { Component } from 'react'

export default class ClassCounterInterval extends Component {

    constructor(props) {

      super(props)

      this.state = {

         count : 0

      }

    }

    componentDidMount(){

        this.interval = setInterval(this.tick , 1000)

    }

    componentWillUnmount(){

        clearInterval(this.interval)

    }

    tick = () => {

        this.setState({

            count : this.state.count + 1

        })

    }

  render() {

    return (

      <div>

        <h1>{this.state.count}</h1>

      </div>

    )

  }

}

**HooksCounterInterval.js**

import React, { useEffect, useState } from 'react'

function HooksCounterInterval() {

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

    const tick = () => {

        setCount(count + 1);

    }

    useEffect((() => {

        const interval = setInterval(tick, 1000);

        return ()=>{

            clearInterval(interval);

        }

    }), [count])

    // Above in second parameter we had to give the count as an array if we wouldnt have given that the interval would have shopped at the 1 only.

  return (

    <div>

      <h1>{count}</h1>

    </div>

  )

}

export default HooksCounterInterval

**FETCHING DATA WITH USEEFFECT**

To perform fetching of data we have to install a module named **Axios.**

**DataFetching.js**

import React, { useEffect, useState } from 'react'

import axios from 'axios'

function DataFetching() {

    const [post , setPost] = useState([])

    useEffect(() => {

        axios.get('https://jsonplaceholder.typicode.com/posts')

        .then(res => {

            console.log(res)

            setPost(res.data)

        })

        .catch(err => {

            console.log(err)

        })

    }, []) // Second parameter means specifying empty dependency list, If we doesn't have a empty array the effect goes on infinite loop.

  return (

    <div>

      <ul>

        {

            post.map(post => (

                <li key={post.id}>{post.title}</li>

            ))

        }

      </ul>

    </div>

  )

}

export default DataFetching

The JSON we are getting from is a website <https://jsonplaceholder.typicode.com>

**Fetching Individual post by passing post id into it.**

Getting an individual post using useEffect, useState and axios.

**DataFetching2.js**

import React, { useEffect, useState } from 'react'

import axios from 'axios'

function DataFetchingTwo() {

    const [post , setPost] = useState({})

    const [id , setId] = useState(1)

    useEffect(() => {

        axios.get(`https://jsonplaceholder.typicode.com/posts/${id}`)

        .then(res => {

            console.log(res)

            setPost(res.data)

        })

        .catch(err => {

            console.log(err)

        })

    }, [id])

  return (

    <div>

        <input type="text" value={id} onChange={e => setId(e.target.value)} />

        <div>

            {post.title}

        </div>

    </div>

  )

}

export default DataFetchingTwo

**Fetching the Data By Clicking the Button**

**DataFetchingThree.js**

import React, { useEffect, useState } from 'react'

import axios from 'axios'

function DataFetchingThree() {

    const [post , setPost] = useState({})

    const [id , setId] = useState(1)

    const [IdFromClickingbtn , setIdFromClickingbtn] = useState(1)

    const HandleClick = () => {

        setIdFromClickingbtn(id)

    }

    useEffect(() => {

        axios.get(`https://jsonplaceholder.typicode.com/posts/${IdFromClickingbtn}`)

        .then(res => {

            console.log(res)

            setPost(res.data)

        })

        .catch(err => {

            console.log(err)

        })

    }, [IdFromClickingbtn])

  return (

    <div>

      <input type="text" value={id} onChange={e => setId(e.target.value)} />

      <button type='button' onClick={HandleClick}>Fetch Post</button>

        <div>

            {post.title}

        </div>

    </div>

  )

}

export default DataFetchingThree

**USECONTEXT**

Context Provides a way to pass data through the component tree without having to pass props down manually at every level

**App.js**

import ComponentC from './components/ComponentC';

import React, { useContext } from 'react';

export const UserContext = React.createContext()

export const ChannelContext = React.createContext()

function App() {

  return (

    <div className="App">

      <UserContext.Provider value={'Syed'}>

        <ChannelContext.Provider value={'Youtube Channel'}>

          <ComponentC/>

        </ChannelContext.Provider>

      </UserContext.Provider>

    </div>

  );

}

export default App;

**ComponentC.js**

import React from 'react'

import ComponentE from './ComponentE'

function ComponentC() {

  return (

    <div>

      <ComponentE></ComponentE>

    </div>

  )

}

export default ComponentC

**ComponentE.js**

import React from 'react'

import ComponentF from './ComponentF'

function ComponentE() {

  return (

    <div>

      <ComponentF></ComponentF>

    </div>

  )

}

export default ComponentE

**ComponentF.js**

import React from 'react'

import { ChannelContext, UserContext } from '../App'

function ComponentF() {

  return (

    <div>

      <UserContext.Consumer>

        {

            user => {

                return (

                    <ChannelContext.Consumer>

                        {

                            channel => {

                                return <div>User Context value: <b>{user}</b>, Channel Context value: <b>{channel}</b></div>

                            }

                        }

                    </ChannelContext.Consumer>

                )

            }

        }

      </UserContext.Consumer>

    </div>

  )

}

export default ComponentF

In the Above components componentC is imported in App.js and in the same way componentE is imported in componentC and ComponentF is in componentE. And the content values are defined in App.js and the values are accessed in ComponentF.js

Above we Saw the creating and using context values in different components.

**Steps For creating and using context value**

Step 1. We created a context using create context value.

Step 2. We created a context value at the high level in the component tree.

Step 3. We consumed the context value using the render props pattern

Now lets use the hook useContext in the ComponentE.js.

The first two steps are the same in using the useContext hook .

In the third step we just do like it is shown below.

**ComponentE.js**

import React, { useContext } from 'react'

import ComponentF from './ComponentF'

import { ChannelContext, UserContext } from '../App';

function ComponentE() {

    const user = useContext(UserContext);

    const channel = useContext(ChannelContext);

  return (

    <div>

        <h2>

            ------------------

            useContext Hook

            ------------------

        </h2>

        <b>

            {user} - {channel}

        </b>

    </div>

  )

}

export default ComponentE

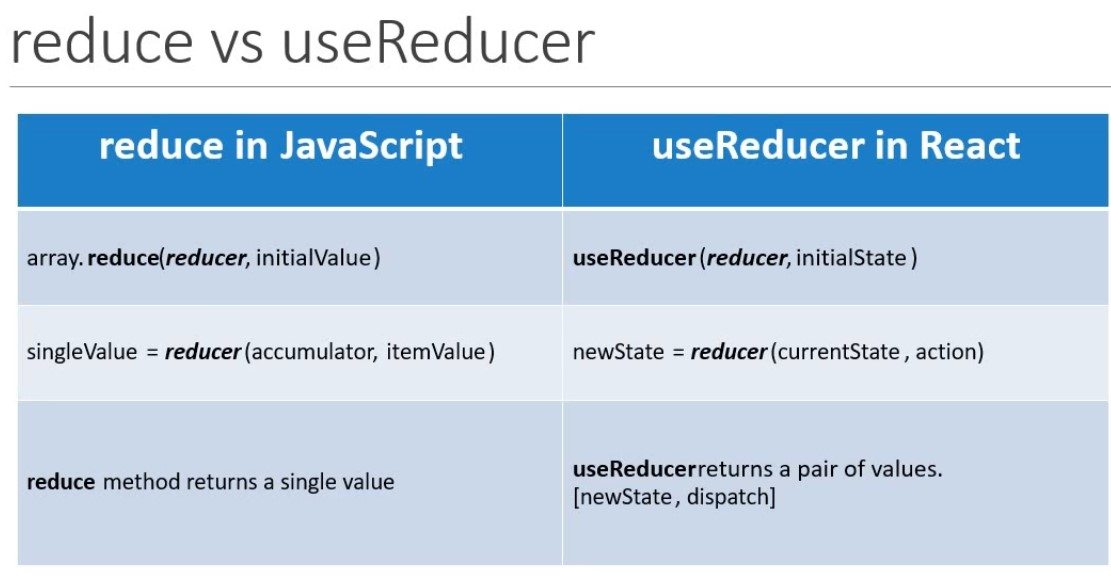
**USEREDUCER**

useReducer is a hook that is used for state management.

It is an alternative to useState.

useState 🡪 State useEffect 🡪 Side-Effects

useContext 🡪 Context API useReducer 🡪 Reducers



useReducer is a hook that is used for state management in react.

useReducer is related to reducer function.

useReducer( reducer, initialState)

reducer(currentState, action).

**useReducer (Simple State & Action)**

**HooksCounteruseReducer.js**

import React, { useReducer } from 'react'

const initialState = 0;

const reducer = (state, action) => {

    switch (action) {

        case 'increment':

            return state + 1;

        case 'decrement':

            return state - 1;

        case 'reset':

            return initialState

        default:

            return state

    }

}

function HooksCounteruseReducer() {

    const [count, dispatch] = useReducer(reducer, initialState)  //useReducer also needs array destructuring.

    // Above count refers to the initialState and dispatch refers to the reducer function.

  return (

    <div>

        <h1>{count}</h1>

      <button onClick={() => dispatch("increment")}>Increment</button>

      <button onClick={() => dispatch("decrement")}>Decrement</button>

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

    </div>// onClick only accepts the function or a arrow function.

  )

}

export default HooksCounteruseReducer

Above code is the simple example for reducer functions.

**useReducer (Complex State & Action)**

**HooksCounteruseReducertwo.js**

import React, { useReducer } from 'react'

const initialState = {

    firstCounter : 0 ,

    secondCounter : 7

};// this is the state Object

const reducer = (state, action) => {  //Here also state refers to the intialState

    switch (action.top) {

        case 'increment':

            return {...state ,firstCounter : state.firstCounter + action.value};

        case 'decrement':

            return {...state ,firstCounter : state.firstCounter - action.value};

        case 'incrementCounter2':

            return {...state ,secondCounter : state.secondCounter + action.value};

        case 'decrementCounter2':

            return {...state ,secondCounter : state.secondCounter - action.value};

        case 'increment5':

            return {...state ,firstCounter : state.firstCounter + action.value};

        case 'decrement5':

            return {...state ,firstCounter : state.firstCounter - action.value};

        case 'reset':

            return initialState

    // Above action.value is given because action should be performed on value not on the given number

        default:

            return state

    }

}

function HooksCounteruseReducertwo() {

    const [count, dispatch] = useReducer(reducer, initialState)  //useReducer also needs array destructuring.

    // Above count refers to the initialState and dispatch refers to the switch function.

  return (

    <div>

        <h1>count1 - {count.firstCounter}</h1>

        <h1>count2 - {count.secondCounter}</h1>

      <button onClick={() => dispatch({top : "increment", value: 1})}>Increment</button>

      <button onClick={() => dispatch({top : "decrement", value: 1})}>Decrement</button>

      <button onClick={() => dispatch({top : "increment5", value: 5})}>Increment 5</button>

      <button onClick={() => dispatch({top : "decrement5", value: 5})}>Decrement 5</button>

      <div>

        <button onClick={() => dispatch({top : "incrementCounter2", value: 7})}>Increment</button>

        <button onClick={() => dispatch({top : "decrementCounter2",value: 7})}>Decrement</button>

      </div>

      <button onClick={() => dispatch({top : "reset"})}>Reset</button>

    </div>// onClick only accepts the function or a arrow function.

  )

}

export default HooksCounteruseReducertwo

**HooksCounteruseReducerThree.js**

import React, { useReducer } from 'react'

const initialState = 0;

const reducer = (state, action) => {

    switch (action) {

        case 'increment':

            return state + 1;

        case 'decrement':

            return state - 1;

        case 'reset':

            return initialState

        default:

            return state

    }

}

function HooksCounteruseReducerThree() {

    const [count, dispatch] = useReducer(reducer, initialState)

    const [counttwo, dispatchtwo] = useReducer(reducer, initialState)

  return (

    <div>

        <div>

            <h1>Count1 - {count}</h1>

            <button onClick={() => dispatch("increment")}>Increment</button>

            <button onClick={() => dispatch("decrement")}>Decrement</button>

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

        </div>

        <div>

            <h1>Count2 - {counttwo}</h1>

            <button onClick={() => dispatchtwo("increment")}>Increment</button>

            <button onClick={() => dispatchtwo("decrement")}>Decrement</button>

            <button onClick={() => dispatchtwo("reset")}>Reset</button>

        </div>

    </div>

  )

}

export default HooksCounteruseReducerThree

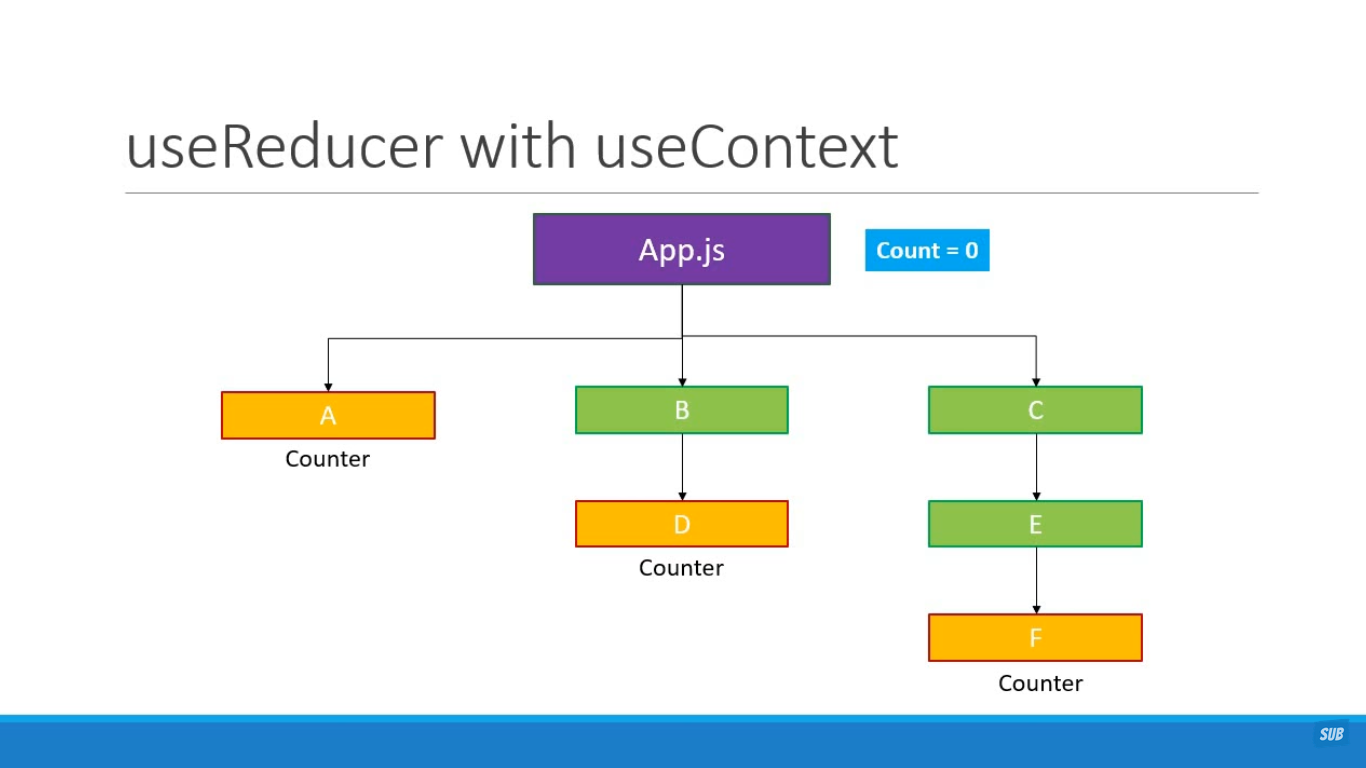
In the Above code we saw different count buttons with different count value but perform the mutual operation i.e., same operation of counter +1, counter -1 and reset

**USEREDUCER WITH USECONTEXT**

useReducer – Local State Management.

Share State between components – Global State Management.

In order to manage globally we have to do **useReducer + useContext**



**App.js**

import React, {useReducer } from 'react';

import UeWithUrA from './components/useContext+useReducer/UeWithUrA';

import UeWithUrB from './components/useContext+useReducer/UeWithUrB';

import UeWithUrC from './components/useContext+useReducer/UeWithUrC';

export const CountContext = React.createContext()

const initialState = 0;

const reducer = (state, action) => {

    switch (action) {

        case 'increment':

            return state + 1;

        case 'decrement':

            return state - 1;

        case 'reset':

            return initialState

        default:

            return state

    }

}

function App() {

  const [count, dispatch] = useReducer(reducer, initialState)

  return (

    <div className="App">

      <CountContext.Provider value={{countState: count, countDispatch: dispatch}}>

        <div>

          Count - {count}

        <UeWithUrA/>

        <UeWithUrB/>

        <UeWithUrC/>

        </div>

      </CountContext.Provider>

    </div>

  );

}

export default App;

**UeWithUrA.js**

import React, { useContext } from 'react'

import { CountContext } from '../../App'

function UeWithUrA() {

    const countContext = useContext(CountContext)

  return (

    <div>

      Component A - {countContext.countState}

        <button onClick={() => countContext.countDispatch("increment")}>Increment</button>

        <button onClick={() => countContext.countDispatch("decrement")}>Decrement</button>

        <button onClick={() => countContext.countDispatch("reset")}>Reset</button>

    </div>

  )

}

export default UeWithUrA

**UeWithUrB**

import React from 'react'

import UeWithUrD from './UeWithUrD'

function UeWithUrB() {

  return (

    <div>

      <UeWithUrD/>

    </div>

  )

}

export default UeWithUrB

**UeWithUrC.js**

import React from 'react'

import UeWithUrE from './UeWithUrE'

function UeWithUrC() {

  return (

    <div>

      <UeWithUrE/>

    </div>

  )

}

export default UeWithUrC

**UeWithUrD.js**

import React, { useContext } from 'react'

import { CountContext} from '../../App'

const UeWithUrD = () => {

    const countContext = useContext(CountContext)

  return (

    <div>

        Component D - {countContext.countState}

        <button onClick={() => countContext.countDispatch("increment")}>Increment</button>

        <button onClick={() => countContext.countDispatch("decrement")}>Decrement</button>

        <button onClick={() => countContext.countDispatch("reset")}>Reset</button>

    </div>

  )

}

export default UeWithUrD

**UeWithUrE.js**

import React from 'react'

import UeWithUrF from './UeWithUrF'

function UeWithUrE() {

  return (

    <div>

      <UeWithUrF/>

    </div>

  )

}

export default UeWithUrE

**UeWithUrF.js**

import React, { useContext } from 'react'

import { CountContext } from '../../App'

function UeWithUrF() {

    const countContext = useContext(CountContext)

    return (

    <div>

      Component F - {countContext.countState}

        <button onClick={() => countContext.countDispatch("increment")}>Increment</button>

        <button onClick={() => countContext.countDispatch("decrement")}>Decrement</button>

        <button onClick={() => countContext.countDispatch("reset")}>Reset</button>

    </div>

  )

}

export default UeWithUrF

**DataFeching using useEffect and useState**

**DataFetchingOne.js**

import React, { useEffect, useState } from 'react'

import axios from 'axios'

function DataFetchingOne() {

    const [loading, setLoading] = useState(true)

    const [error, setError] = useState("")

    const [post, setPost] = useState({})

    useEffect(() => {

        axios.get("https://jsonplaceholder.typicode.com/posts/1")

        .then(response => {

            setLoading(false)

            setPost(response.data)

            setError('')

        })

        .catch(error => {

            setLoading(false)

            setPost({})

            setError("Something Went Wrong")

        })

    }, []);

  return (

    <div>

      {loading ? 'loading' : post.title}

      {error ? error : null}

    </div>

  )

}

export default DataFetchingOne

**DataFetching using useReducer and useEffect**

**DataFetchinguseRuseE.js**

import React, { useEffect, useReducer } from 'react'

import axios from 'axios'

const initialState = {

  loading : true,

  post : {},

  error : ""

}

const reducer = (state, action) => {

  switch(action.type){

    case "FETCH-SUCCESS":

      return {

        loading: false,

        post : action.payload,

        error : ""

      }

    case "FETCH-ERROR":

      return {

        loading : false,

        post : {},

        error : "Something is Wrong"

      }

    default:

      return state

  }

}

function DataFetchinguseRuseE() {

  const [state, dispatch] = useReducer(reducer, initialState)

  useEffect(() => {

    axios.get("https://jsonplaceholder.typicode.com/posts/1")

    .then(response => {

        dispatch({

          type : "FETCH-SUCCESS",

          payload : response.data

        })

    })

    .catch(error => {

        dispatch({

          type : "FETCH-ERROR"

        })

    })

}, []);

  return (

    <div>

      {state.loading ? 'loading' : state.post.title}

      {state.error ? state.error : null}

    </div>

  )

}

export default DataFetchinguseRuseE

**USESTATE V/S USEREDUCER**

|  |  |  |
| --- | --- | --- |
| **Scenario** | **useState** | **useReducer** |
| **Type of State** | Number, String, Boolean | Object or Array |
| **Number of State Transitions** | One or Two | Too Many |
| **Related State Transitions** | No | Yes |
| **Business Logic** | No Business Logic | Complex Business Logic |
| **Local V/S Global** | Local | Global |

**Performance issue in react while rendering the code.**

Suppose we have a parent component and five children components is being rendered in the parent component. In five component the first component is of only a title component, two components are of increment buttons ( Increment is done using useState ) and two components are of showing the increment value buttons and every component has console.log in its code. When we are making use of the component like clicking an increment button then all the components is getting rerendered again, we used console.log statements to confirm. Inorder to avoid this kind of thing we use **React.memo(Component)** while exporting the react component. This doesn’t help us to overcome fully but it helps to differtiate between the components.

Using React.memo the component only gets rendered when there is a change in props or state.

**USECALLBACK HOOK**

useCallback is a hook that will return a memorized version of the callback function that only changes when one of the dependencies has changed.

It is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

**Parent.js**

import React, { useCallback, useState } from 'react'

import Count from './Count'

import Button from './Button'

import Title from './Title'

function Parent() {

    const [age, setAge] = useState(18)

    const [salary, setSalary] = useState(50000)

    const incrementAge =useCallback(() => {

        setAge(age + 1);

    },[age])

    const incrementSalary = useCallback(() => {

        setSalary(salary + 1000)

    },[salary])

  return (

    <div>

      <Title></Title>

      <Count text="Age" count={age}></Count>

      <Button handleClick={incrementAge}>Increment Age</Button>

      <Count text="salary" count={salary}></Count>

      <Button handleClick={incrementSalary}>Increment Salary</Button>

    </div>

  )

}

export default Parent

**Title.js**

import React from 'react'

function Title() {

  return (

    <div>

      USECALL HOOKS

    </div>

  )

}

export default React.memo(Title)

**Count.js**

import React from 'react'

function Count({text, count}) {

    console.log(`Rendering......${text}`);

  return (

    <div>

      {text} - {count}

    </div>

  )

}

export default React.memo(Count)

**Button.js**

import React from 'react'

function Button({handleClick, children}) {  // children refers to the value written in between the redering button component

    console.log(`Redendering.........${children}`);

  return (

    <button onClick={handleClick}>

      {children}

    </button>

  )

}

export default React.memo(Button)

**USEMEMO HOOK**

**Counter.js**

import React, { useMemo, useState } from 'react'

function Counter() {

    const [counterOne, setCounterOne] = useState(0)

    const [counterTwo, setCounterTwo] = useState(0)

    const incrementOne = () => {

        setCounterOne(counterOne + 1)

    }

    const incrementTwo = () => {

        let i = 0

        while (i<200000) {

            i++

        }

        setCounterTwo(counterTwo + 1)

    }

    const isEven = useMemo(() => {

        let i = 0;

        while(i<2000000000) i++

        return counterOne % 2 === 0

    },[counterOne])

  return (

    <div>

      <div>

        <button onClick={incrementOne}>Clicked - {counterOne}</button>

        <span>{isEven ? "Even" : "Odd"}</span>

      </div>

      <div>

        <button onClick={incrementTwo}>Clicked - {counterTwo}</button>

      </div>

    </div>

  )

}

export default Counter

useMemo is also Used for the code optimization same as useCallback is used for but there is a difference.

useCallback caches the provided function instance itself whereas useMemo invokes the provided function and caches its result. So, if we need to cache the function useCallback is used and we need to cache the result we use useMemo.

**USEREF HOOK**

useRef is a React Hook that lets you reference a value that's not needed for rendering. It is also used for focusing the an element in the component after loading.

**InputForm.js**

import React, { useEffect, useRef } from 'react'

function InputForm() {

    const inputReference = useRef(null)

    useEffect(() => {

        // Here we are wrting logic to focus the input area onloading of the component

        inputReference.current.focus()

    }, [])

  return (

    <div>

      <input ref={inputReference} type="text" />

    </div>

  )

}

export default InputForm

**ClassTimer.js**

import React, { Component } from 'react'

class ClassTimer extends Component {

    interval

    constructor(props) {

      super(props)

      this.state = {

         timer : 0

      }

    }

    componentDidMount() {

        this.interval = setInterval(()=>{

            this.setState(prevState => ({timer : prevState.timer + 1}))

        }, 1000)

    }

    componentWillUnmount(){

        clearInterval(this.interval)

    }

  render() {

    return (

      <div>

        Class Timer - {this.state.timer}

        <button onClick={() => clearInterval(this.interval)}>Clear Timer</button>

      </div>

    )

  }

}

export default ClassTimer

**HooksTimer.js**

import React, { useEffect, useRef, useState } from 'react'

function HooksTimer() {

    const [timer, setTimer] = useState(0)

    const intervalref = useRef();

    useEffect(()=>{

        intervalref.current = setInterval(() => {

            setTimer(prevTimer => prevTimer + 1)

        }, 1000);

        return () => {

            clearInterval(intervalref.current)

        }

    },[])

  return (

    <div>

      Hooks Timer - {timer}

        <button onClick={() => clearInterval(intervalref.current)}>Hooks Timer Clear</button>

    </div>

  )

}

export default HooksTimer

For clearing the timer we have to use useRef because the useEffect’s variable is not access in the JSX .

**CUSTOM HOOKS**

A custom hook is basically a Javascript function whose name starts with “use”.

A custom hook can also call other Hooks if Required.

Custom hooks are used to share Logic – Alternative to High Order Component and Render Props

**DocTitleCounter.js**

import React, { useState } from 'react'

import useDocCounter from './useDocCounter'

function DocTitleCounter() {

    const [counter, setCounter] = useState(0)

    useDocCounter(counter)

  return (

    <div>

      <button onClick={() => setCounter(counter + 1)}>Clicked {counter}</button>

    </div>

  )

}

export default DocTitleCounter

In the above code useDocCounter is the custom hook. And below is the program for initializing the hook.

**useDocCounter.js**

import React, { useEffect } from 'react'

function useDocCounter(count) {

    useEffect(() => {

        document.title = `Clicked ${count}`

    },[count])

}

export default useDocCounter