ReactJS useReducer Hook

The useReducer Hook is the better alternative to the useState hook and is generally more preferred over the useState hook when you have complex state-building logic or when the next state value depends upon its previous value or when the components are needed to be optimized.

The useReducer hook takes three arguments including reducer, initial state, and the function to load the initial state lazily.

**Syntax:**

**const [state, dispatch] = useReducer(reducer, initialArgs, init);**

Example: Here reducer is the user-defined function that pairs the current state with the dispatch method to handle the state, initialArgs refers to the initial arguments and init is the function to initialize the state lazily.

Example program

import React, { useReducer } from "react";

// Defining the initial state and the reducer

const initialState = 0;

const reducer = (state, action) => {

switch (action) {

case "add":

return state + 1;

case "subtract":

return state - 1;

case "reset":

return 0;

default:

throw new Error("Unexpected action");

}

};

const App = () => {

// Initialising useReducer hook

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

return (

<div>

<h2>{count}</h2>

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

add

</button>

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

subtract

</button>

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

reset

</button>

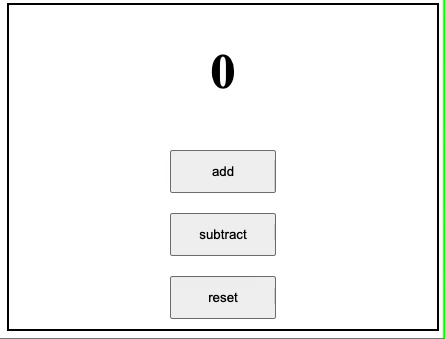
</div>

);

};

export default App;

Output:



**ReactJS useCallback Hook**

The useCallback hook is used when you have a component in which the child is rerendering again and again without need.

Pass an inline callback and an array of dependencies. useCallback will return a memoized version of the callback that only changes if one of the dependencies has changed. This is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

**Syntax:**

const memoizedCallback = useCallback(

() => {

doSomething(a, b);

},

[a, b],

);

With useCallback hook: To solve this problem we can use the useCallback hook. Here, the filename is App.js.

**Javascript**

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

var funccount = new Set();

const App = () => {

const [count, setCount] = useState(0)

const [number, setNumber] = useState(0)

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);

alert(funccount.size);

return (

<div>

Count: {count}

<button onClick={incrementCounter}>

Increase counter

</button>

<button onClick={decrementCounter}>

Decrease Counter

</button>

<button onClick={incrementNumber}>

increase number

</button>

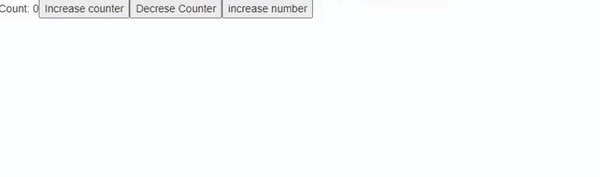
</div>

)

}

export default App;

**Output:**

**  
React JS useMemo Hook**

The useMemo is a hook used in the functional component of react that returns a memoized value. In Computer Science, memoization is a concept used in general when we don’t need to recompute the function with a given argument for the next time as it returns the cached result.

A memoized function remembers the results of output for a given set of inputs. For example, if there is a function to add two numbers, and we give the parameter as 1 and 2 for the first time the function will add these two numbers and return 3, but if the same inputs come again then we will return the cached value i.e 3 and not compute with the add function again.

**Syntax:**

const memoizedValue = useMemo(functionThatReturnsValue, arrayDepencies)

**Example:**

When we use useMemo Hook

import React, {useState} from 'react';

function App() {

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);

}

return (

<div className="App">

<h1>Welcome to Geeksforgeeks</h1>

<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>

</div>

);

}

// function to square the value

function squareNum(number){

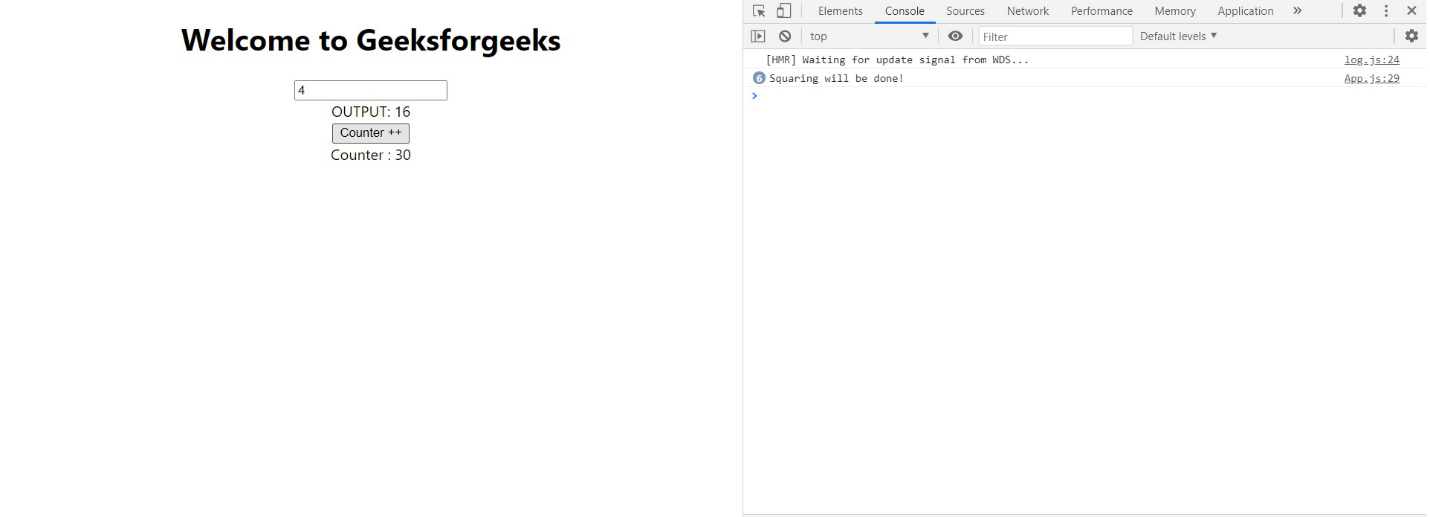
console.log("Squaring will be done!");

return Math.pow(number, 2);

}

export default App;

**Output:**



**ReactJS | Router**

React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.

After installing react-router-dom, add its components to your React application.

**Adding React Router Components**: The main Components of React Router are:

**BrowserRouter:**

BrowserRouter is a router implementation that uses the HTML5 history API(pushState, replaceState and the popstate event) to keep your UI in sync with the URL. It is the parent component that is used to store all of the other components.

**Route:**

Route is the conditionally shown component that renders some UI when its path matches the current URL.

**Link:**

Link component is used to create links to different routes and implement navigation around the application. It works like HTML anchor tag.

**Switch:**

Switch component is used to render only the first route that matches the location rather than rendering all matching routes. Although there is no defying functionality of SWITCH tag in our application because none of the LINK paths are ever going to coincide. But let’s say we have a route (Note that there is no EXACT in here), then all the Route tags are going to be processed which start with ‘/’ (all Routes start with /). This is where we need SWITCH statement to process only one of the statements.

To add React Router components in your application, open your project directory in the editor you use and go to app.js file. Now, add the below given code in app.js.

import {

BrowserRouter as Router,

Route,

Link,

Switch

} from 'react-router-dom';

**Using React Router:**

To use React Router, let us first create few components in the react application. In your project directory, create a folder named component inside the src folder and now add 3 files named home.js, about.js and contact.js to the component folder.

Let us add some code to our 3 components:

**Home.js:**

import React from 'react';

function Home (){

return <h1>Welcome to the world of Geeks!</h1>

}

export default Home;

**About.js:**

import React from 'react';

function About () {

return <div>

<h2>GeeksforGeeks is a computer science portal for geeks!</h2>

Read more about us at :

<a href="https://www.geeksforgeeks.org/about/">

https://www.geeksforgeeks.org/about/

</a>

</div>

}

export default About;

**Contact.js:**

import React from 'react';

function Contact (){

return <address>

You can find us here:<br />

GeeksforGeeks<br />

5th & 6th Floor, Royal Kapsons, A- 118, <br />

Sector- 136, Noida, Uttar Pradesh (201305)

</address>

}

Export default Contact;

.

**BrowserRouter:**

Add BrowserRouter aliased as Router to your app.js file in order to wrap all the other components. BrowserRouter is a parent component and can have only single child.

**Link:**

Let us now create links to our components. Link component uses the to prop to describe the location where the links should navigate to.

**Route:**

Route component will now help us to establish the link between component’s UI and the URL. To include routes to the application, add the code give below to your app.js.

**Switch:**

To render a single component, wrap all the routes inside the Switch Component.

After adding all the codes

import React, { Component } from 'react';

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

import Home from './component/home';

import About from './component/about';

import Contact from './component/contact';

import './App.css';

class App extends Component {

render() {

return (

<Router>

<div className="App">

<ul className="App-header">

<li>

<Link to="/">Home</Link>

</li>

<li>

<Link to="/about">About Us</Link>

</li>

<li>

<Link to="/contact">Contact Us</Link>

</li>

</ul>

<Switch>

<Route exact path='/' component={Home}></Route>

<Route exact path='/about' component={About}></Route>

<Route exact path='/contact' component={Contact}></Route>

</Switch>

</div>

</Router>

);

}

}

export default App;