React JS:-

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

useState is a react hook which is used to store variable state, it can store the variable state in the form of number, string, object, array, Boolean etc.,

example:-

import React, { useState } from 'react'

const App=()=>{

const [data,setData]= useState(0)

return(

<div>

<h1>Count:{data} </h1>

<button onClick={()=>setData(data+1)}>Add</button>

</div>

)

}

export default App;

useEffect:-

The useEffect Hook allows you to perform side effects in your components.Some examples of side effects are: fetching data, directly updating the DOM, and timers.

syntax: useEffect(()=>{

//actual code

},[dependencies])

In the above syntax "dependencies" is optional parameters, if you want the code to be executed once make it empty.

If any value is specified in the dependency the code inside the first parameter gets executed whenever the dependency gets changed.

Example:-

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

const App = () => {

const [data, setData] = useState(0);

useEffect(() => {

setTimeout(() => {

setData((data) => data + 1);

}, 1000);

});

return (

<div>

<h1>Count:{data} </h1>

{*/\* <button>Add</button> \*/*}

</div>

);

};

export default App;

**useCallback hook:**

The React useCallback Hook returns a memoized callback function. Think of memoization as caching a value so that it does not need to be recalculated.

This allows us to isolate resource intensive functions so that they will not automatically run on every render. The useCallback Hook only runs when one of its dependencies update.

This can improve performance. The useCallback and useMemo Hooks are similar. The main difference is that useMemo returns a memoized value and useCallback returns a memoized function.

syntax: useCallback(()=>{

//actual code

},[dependencies]);

Example:-

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

import Task from "../task";

const App = () => {

const [data, setData] = useState(0);

const [tas, setTas] = useState([]);

const task = useCallback(() => {

setTas([...tas, "hello"]);

});

return (

<div>

<h1>Count:{data} </h1>

<button onClick={() => setData(data + 1)}>Add</button>

<br />

<Task task={task} tas={tas} />

</div>

);

};

export default App;

**useContext hook:**

useContext hook is a way to manage state globally.It can be used

together with the useState Hook to share state between deeply nested components more easily than with useState alone.

Example:- same component

import React, { useState, useContext, createContext } from "react";

const usercontext = createContext();

const App = () => {

const [data, setData] = useState("hello javascript");

return (

<div>

<usercontext.Provider value={data}>

<App2 />

</usercontext.Provider>

</div>

);

};

export default App;

function App2() {

const user = useContext(usercontext);

return (

<div>

<h1>hello</h1>

<p>{user}</p>

</div>

);

}

Example 2:- separatecomponent

import React, { useState } from "react";

import App2 from "../app2";

import UserContext from "../context";

const App = () => {

const [data, setData] = useState("hello javascript");

return (

<div>

<UserContext.Provider value={{ data, setData }}>

<App2 />

</UserContext.Provider>

<h1>hello</h1>

</div>

);

};

export default App;

context.js

import { createContext } from "react";

*// import ReactDOM from "react-dom/client";*

const UserContext = createContext();

export default UserContext;

app2

import React, { useContext } from "react";

import UserContext from "./context";

function App2() {

const { data, setData } = useContext(UserContext);

return (

<div>

<h1>{data}</h1>

</div>

);

}

export default App2;

**use ref :-**

The useRef Hook allows you to persist values between renders.

It can be used to store a mutable value that does not cause a re-render when updated.

It can be used to access a DOM element directly.

Example:-