***USECONTEXT:***

***Need of UseContext:***

* *Suppose there is a data present at a particular component A and we need that data in a particular component E, then we have to pass data to our top level App component, from when we pass the data to our child component E using the technique of* ***prop drilling(passing datas from parent to the lowest level of component using props).***
* *So, this isn’t a proper way of passing data between 2 components separated by a huge distance, as here the components which donot require that particular data are also receiving it.*
* ***So, the solution of this is usecontext or a way to store data in a external storage from where every component can get that very particular data.***

***How to Use usecontext():***

***usecontext.jsx file:***

import { createContext, useState } from "react";

***// Storage part stores the default value which we wanna access*.**

***SNIPPET:***

*export const UserContext =createContext({*

*currentUser: null,*

*setcurrentUser: ()=>null,*

*})*

***Acutal component used for wrapping all the components who needs the***

***External storage values.***

***Properties:***

* ***{Children}*** *: It refers to the child component.*
* ***UserContext****.****Provider****: IT wraps the components where we need our*

*storage values*

* ***value:*** *It is the value{state,UseState} here which our chidren component can*

*access to.*

***SNIPPET:***

*export const UserProvider =({children})=>{*

*const [currentUser, setcurrentUser] = useState(null);*

*const value ={currentUser,setcurrentUser}*

*return <UserContext.Provider value={value}>{children}</UserContext.Provider>*

*}*

***Index.js File:***

*i****mport { UserProvider } from './context/user.context';***

*const root = ReactDOM.createRoot(document.getElementById('root'));*

*root.render(*

*<React.StrictMode>*

***<UserProvider>***

*<BrowserRouter>*

*<App/>*

*</BrowserRouter>*

***</UserProvider>***

*</React.StrictMode>*

*);*

***Component A:***

***Snippet:***

*const A = () => {*

*const user=[{random valuesA}];*

***const {setcurrentUser} = useContext(UserContext);***

***setcurrentUser(user);***

*return(*

*<></>*

*)*

*}*

***Component B:***

***Snippet:***

*const A = () => {*

*const user=[{random valuesA}];*

***const {currentUser} = useContext(UserContext);***

***console.log(currentUser); //Prints [{random valuesA}]***

*return(*

*<></>*

*)*

*}*

***NOTE:***

***whenever we change the external storage value through component A, then the***

***whole component B gets re-render again with new set of values if we use that external storage value/ our that functional component re runs without rerendering the dom if we aren’t using the external storage value in our jsx,***

***as ultimately the state gets changed here.***