Pre-requiste for React

HTML:elements,tags,attributes

css:general Properties,selectors,flext,grid

JS: Varibales,Data types,Function,Operators,selection statements:if else switch

object,array,DOM and Dom manipulation,Events

Advance Js:ES6,Promises,oops,let var const,Module scope,clousers

What is React or React JS ?

-React JS is a JavaScript Library

-React Js is not a framework

What is Library?

-Library is a collection of pre-defined functions,classes or Objects

-React is a collection of pre-defined function,classes and Objects

What is the purpose of ReactJS?

-React JS is used to create UserInterface

-If we have to create frontend application then it wont possible by react we need other libraries like ReactDOM,Redux,Axios,ReactRouter

React==>UserInterface==>+==>Other Libraries==>FrontEnd application(Frontend Engineers)

What is UserInterface?

-Collection of UI Components

We can create UI with HTML,JS but with React JS?

-rendering,siglepage,etc...

Create a simple Button Using HTML

Button Element-----Button Tag



Create a simple button using JS

Pre-defined object and functions

DOM Button Element

DOM

Creating UI(Button) is easy on HTML In Html we cannot reuse the UI

Creating UI in JS looks little complex In js we can reuse the UI

So both html and js have adv and dis to overcome both we use reactjs

React

In React, creating UI is easy and simple like html

In react, we can reuse as UI multiple times

Why react is used to create user interface?

1.Creating UI is simple in react

2. We can re-use the UI in the react

Creating simple using React

1.Create a html file

2. Add basic code of html

3.add script tag inside body so that we can write react code

4.Integrate reactjs with html document

HTML-------------HTML elements-------------------UI Comp

JS---------Predefined object and function--------DOM element------add in the dom-------UI

React-------Predefined object and functions----------react element add in the dom-------ui

5.Create React Element

Var reactele=.createElement(<tagName>,{--},children)

Children means html children is body and body children is tittle etc

6.Add react element inside the DOM

ReactDOM is a library which is used to add React Elements inside the DOM

Integrate ReactDOM with HTML Document

ReactDOM.render(<React element>,<DomElementReference>);

Creating UI in reactjs is simple or complex?

-Creating UI components is complex because of createElement same as js so JSX (same as HTML) came into picture

ReactDev----JSX Element-----babel compilers----ReactElements---ReactDOM---DOM---UI

HTML JSX

<h2>----</h2> var h2=<h2>---</h2>

<button>--</button> var button=<button>----</button>

How to configure attributes in react?

A)Second argument in React.createElement({----}); is used for configuring attributes

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Task1</title>

    <script src="https://unpkg.com/react@18/umd/react.development.js"></script>

    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

</head>

<body>

    <div id="root"></div>

    <script>

        let div=React.createElement('div',null,

        React.createElement('div',null,

        React.createElement('div',null,

            [React.createElement('h2',{},"React Elements"),

             React.createElement('p',{},"Reacts look very complex now")

            ]

        )

        ));

        ReactDOM.render(div,document.getElementById('root'));

    </script>

</body>

</html>

Complex div inside div inside div inside p and h2 tags

Dev---React---ReactElements—DOM using ReactDOM---UI

Dev---JSX Elements----babel Compiler(convert jsx to react elements) ----ReactElements---ReactDOM---DOM—UI

-Use of Babel compiler is converting jsx to react elements

Note: We need to add script tag of babel and need to add  <script type="text/babel">.

* **JSX** → Lets you write HTML-like syntax inside JavaScript (developer-friendly).
* **Babel** → Converts JSX & modern JavaScript into plain JavaScript (browser-friendly).

How to add Javascript code inside the jsx element?

Syntax: <jsx-element>{javascript code}</jsx-element>

{} is mandatory

Eg: let container=(

            <div>

            {card}

            {card}

            </div>

        );

If we want to add js code inside HTML(JSX) we need to add {}.

-Create functions for passing arguments and easy reusing of component

In react, whenever we want to create any UI then, that UI we will always create in a javascript function.

Why we have to create UI inside the javascript function?

1. We can re-use the UI
2. We can pass dynamic data to the UI

Function createCard(){

Retrun <div></div>

}

createCard();

Instead of calling functions like this In react js we can call like element tag

Eg: <CreateCard></CreateCard> or <CreateCard/>

But the first letter should be captical.

Normal calling:

function multipleContainer(){

            return (

                <div>

                    {createContainer("React JS")}

                    {createContainer("Angular")}

                    {createContainer("Node JS")}

                </div>

            );

        }

Element tag based function:

 function MultipleContainer(){   //first letter should be capital

            return (

                <div>

                    <CreateContainer/>

                    <CreateContainer/>

                    <CreateContainer/>

                </div>

            );

        }

Functional Component:

-It is a simple javascript function whose name starts with uppercase and which returns UserInterface(JSX or React Elements).

There are two types of Component in React

1. Functional Component (Mostly used)
2. Class Component (Not recommended to use)

Components represents view on the screen or on the webpage

React application is a collection of components.

Check TechContainer.html for example

Note:

Any component which we assign to ReactDOM, that component is considered as a parent component for entire application.

Eg: ReactDOM.render(<App/>,document.getElementById('root'));

Rendering = calling

Rendering=calls+converted to react elements +added in the dom

What is the difference between normal js and react functional component?

1. RFC should always starts its name with uppercase
2. Normal js function can start names with any case
3. RFC will always return jsx code
4. NJF can return any type of value
5. RFC contains JSX code
6. NJF does not contain any jsx code

Note:

Using Attribute Concept we can pass arguments to the components if it is rendered in the element or Tag format

Props:

Props are the properties of component

Props are used to pass dynamic data or input data to the components

Props are used to share the data from parent comp to child compoent

Eg:

 function CreateCard(props){  //data={title:" ",image:" "} props stored

            //JSX code to create a card

            return (<div class="card">

                    <img src={props.image} width="100%" height="210px"/>

                    <h3>{props.title}</h3>

                    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Aspernatur dolore, obcaecati totam veritatis quo maiores. Voluptate quidem deleniti rem iste sint sunt facilis culpa ab, iusto nostrum iure provident. Atque.</p>

                    <button>Profile Details </button>

                </div>);

        }

Always first parameter is object it will store all the parameters passed through the function

eg:

<CreateCard title="Nikhil Thula" image="https://media.istockphoto.com/id/1391718981/photo/portrait-of-a-confident-young-businessman-standing-with-his-arms-crossed-in-an-office.jpg?s=612x612&w=0&k=20&c=eF\_0QCtw-Y8Q2c4\_xQe6KTkcSPiGCT6qBf6nuavE2Dg="/>

We need to pass function CreateCard(props){  //data={title:" ",image:" "} props stored like this

<img src={props.image} width="100%" height="210px"/>

Note:

Functional component first parameter will be always object, this object stores props or this object contains set of props as a properties.

Tools used to develop ReactApplication

1.CRA – it takes to time to create application slow

2. Vite🡪Basic React Application

Installed

Pre-Configured

Easy and simple to install configure other libraries

Optimised code for production

Development server

Start,stopand loading appp or project inside this dev server

url—to access our application in any browser

Create a basic React application using vite

1. Create a folder in any drive and open it in the terminal
2. Add the following command to react application

npm create vite@latest

1. Give the Name for the project(ex:appdemo)
2. Select React Options
3. Change the Directory to the React Application

cd <reactapp-name>

1. Add the following command

npm install

1. Open the react app in vs code

code.(to open vscode)

1. Open the terminal and add the following command to run development server

npm run dev(to run in normal cmd in vscode)

1. Open the Browser and the following url to access react application
2. <http://localhost:5173/>

Rules to be followed in project

1. Every Component should have there own file
2. Component name should be same as file name
3. All .jsx,.js or .css files should be created inside src folder only

React Component------------ReactDOM--------DOM

-If we want to create any file we need to create folder any side that only we need create jsx and css files

Flow is, first browser--🡪 index.html-🡪App.jsx(parent and it is common for every app)-🡪header.jsx(changes accordingly)

Devloper ----------🡪 API’s(pre-defined functions and objects)---------🡪DOM (This is normal HTML code)

Eg: This is simple js code for dom manipulation

var count=0;

function increaseCount(){

    count=count+1;

   let pref= document.querySelector('p');

   pref.innerText="Count value is: "+(count);

}

//DOM Manipulation

Devloper -----------🡪React -----------🡪 API’s(pre-defined functions and objects)-----------🡪DOM Manipulation (This is in React)

-React know very well how to manipulate dom instead we manipulate we tell react to manipulate dom.

-In react we use useState() for dynamic value change (dom manipulation) and rerendering the component.

-If data from UI then it should come from “State” only.

State

Important

-State is a special variable in React, where we can store any type of data.

-State Data we can bind with JSX element

-Whenever state gets updated, internally react take care updating UI with new State Data.

Q) How to create or get a State variable?

-Using useState(), we can get a state variable

Var[state,setState]=useState(); (state=variable,setState=setter function)

Q) How to update the state variables?

-Using setter function i.e setState() we can update the data of state variable.

setState(data);

eg for both state and setState:

var [state,setState]=useState(0); //here 0 is stored in state

function increaseCount(){

        setState(state+1);

   }

Destructuring

var [state, setState] = useState(0);

useState(0) returns an array: [currentStateValue, functionToUpdateIt]

[state, setState] uses array destructuring to assign:

state → the first item in the array (initially 0)

setState → the second item (the function to update it)

Can we update the state variable without using setState()

Yes (in console we can update but not on UI)

But the problem is UI will not get updated.

Bcz of setState() it does

1.It will update the state

2. Re-render(re-call) the

 function increaseCount(){

        /\* state=state+1;

        console.log(state); \*/

        setState(state+1); //1-re-render

        setState(state+1); //1

        setState(state+1); //1

        //Last will execute and render what ever it is like setState(state+3)

    }

setState(state+3); if it is last then this will execute and the result will be this line only

Internally setState uses Queue DS that is first in first out last request only rendered before it wont.

=> setState(state+1); //1-re-render

        console.log(state);//0

This will print 0 bcz setState is Asynchronous function until setState() get the result before it self console.log(state) executes and print the value as 0.

Function that is passed as a argument to setState() is called updater function

Hook:

Any function whose name starts with the word use, all those functions are called React Hook.

Eg: useState()

Rules of React hook:

1. React Hooks we can use only inside the functional component
2. React Hooks should be called at top level the FC.
3. Hooks cannot be called outside the component or inside JSX or inside any Event Handler.

What is React Hook?

React Hook is a simple JS Function with some rules.

Rules are above.

Routing in React

Routing is a technique to switch from one component to another component

Routing cannot be implemented using react library.

ReactRouter is a library which implements routing in React Application

ReactRouter provides predefined react component and hooks, by using which we can implement routing in react application.

Some of the react components and hooks

BrowserRouter useNavigate()

Route useParams()

Link useSearchParams()

Outlet

Navigate

NavLink

Routes

Router setup

1. Installation of Library

npm install react-router-dom

NPM-Node Package Manager

-It helps in

-Track the version of each dependency

-Let others install all dependencies by just running,

Configuration:

BrowserRouter is a component created by react-router-dom

1. Wrap parent component (App) of the application inside the BrowserRouter Component. (in main.jsx)
2. const root = ReactDOM.createRoot(document.getElementById('root'));
3. root.render(
4. <BrowserRouter>
5. <App/>
6. </BrowserRouter>
7. ); //parent component
8. Change the browser Url path on click of any link.

Add url paths to the href attribute of the anchor tag

 <div class="rightPanel flexContainer">

                <a href="/">Home</a>

                <a href="/products">Products</a>

                <a href="/counter">Counter</a>

                <a href="/imagecomp">ImageComp</a>

            </div>

1. Configure the Route.
2. <Route path={"/"} element={<Home/>}/>
3. <Route path={"/products"} element={<Products/>}/>
4. <Route path={"/counter"} element={<Counter/>}/>
5. <Route path={"/imagecomp"} element={<ImageComp/>}/
6. Wrap all <Route/> Component inside the <Routes></Routes> component.
7. <Routes>
8. <Route path={"/"} element={<Home/>}/>
9. <Route path={"/products"} element={<Products/>}/>
10. <Route path={"/counter"} element={<Counter/>}/>
11. <Route path={"/imagecomp"} element={<ImageComp/>}/>
12. </Routes>
13. If we click on link in nav bar like home products then the page is reloading which is not correct bcz single page wont reload to over that we have to replace all the anchor tag with link component and replace href attribute with to props
14. <div class="rightPanel flexContainer">
15. <Link to="/">Home</Link>
16. <Link to="/products">Products</Link>
17. <Link to="/counter">Counter</Link>
18. <Link to="/imagecomp">ImageComp</Link>
19. </div>

What is Single Page Application?

Ans) Any application which loads a single html document (index.html) in the browser and its content is changed dynamically using any JavaScript API’s like react, angular or plain js.

Eg: Gmail,Instagram,Airbnd,Netfilx, gitlabs etc

Nested Routing

Products Comp

|  |  |
| --- | --- |
| Electronis  Jewelery  Mens Clothing  Womens Clothing |  |

If we want to load any component inside other use like this

In Product.jsx

<div>

            <h2 style={{color:"green"}}>Products Components</h2>

            <div className='productsContainer'>

                <div className="linksContainer">

                      <Link to="electronics" style={{color:"white",textDecoration:"none",fontSize:"21px"}}>Electronics</Link>

                        <Link to="jewellery" style={{color:"white",textDecoration:"none",fontSize:"21px"}}>Jewellery</Link>

                        <Link to="mensclothing" style={{color:"white",textDecoration:"none",fontSize:"21px"}}>MensClothing</Link>

                        <Link to="womensclothing" style={{color:"white",textDecoration:"none",fontSize:"21px"}}>WomesClothing</Link>

                </div>

                <div className="componentConatainer">

                        <Outlet/>

                </div>

            </div>

        </div>

Note->In above we remove / from to if we want to render in other component

In app.jsx

<Route path={"/products"} element={<Products/>}>

        <Route path={"electronics"} element={<Electronics/>}/>

        <Route path={"jewellery"} element={<Jewellery/>}/>

        <Route path={"mensclothing"} element={<MensClothing/>}/>

        <Route path={"womensclothing"} element={<WomesClothing/>}/>

      </Route>

Note: In App.jsx also we need to remove / from path and keep all the sub components inside main component check closing tags

Use <Outlet/> tag for rendering the output on existing screen.

<div className="componentConatainer">

                        <Outlet/>

                </div>

Check all the imports from react-router-dom

To map error page we use this in App.jsx always keep this at the end

<Route path={"\*"} element={<PageNotFound/>}/>

To get the data from DB i.e from api backend we need to use

1. Axios
2. Fetch()
3. Ajax
4. Install Axios

Npm install axios

1. Import axios

Import axios from “axios”

1. Use required functions from axios to send the request

Flow

 req

Component🡪 axios🡪server🡪DataBase

 res

Here after coming data from server to axios data will be stored in promise object

Axios:

Get Request

Installation

Import

Get function from axios to send a request to a backend app(fakestore)

If we writing js inside jsx then we to write it inside {}

Eg:

 <div className="allPrdouctsData">

               {

                allProducts.map((ele)=>{

                    return <Cards image={ele.image} title={ele.title} price={ele.price}/>

                })

               }

            </div>

-If something is repeating for multiple times use map

<div style={{textAlign:"left",padding:"50px"}}>

            <h2>UserNames</h2>

            <ul>

                 {

                    names.map((ele)=>{

                    return <li>{ele}</li>

                })

                }

            </ul>

        </div>

Whenever certain UI need to be created based on the data then go for map.

Eg:

In this way if we replace objects instead of arrays then can we use map function then ans is no.

let [names,setNames]=useState({“name1:nikhil”,”name2”:”Ajay”,..});

Can we use this map function on Object?

Ans)No

1. Relationship between the components
2. Parent Child
3. Sibiling

Function A(){

Return <div>

<B/>

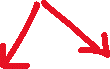
</div>

}

<Route></Route> and <Card/> if double use / staring like route but if card only single then use / at end.

Here A is parent and B is child component

App



Nav Home



Siblings

App--🡪 A---🡪A1---🡪A2

A is direct child for app and A1 is indirect child likewise A1 is direct child for A and A2 is indirect child

Sharing the Data between the components

Parent-child

Child-parent

Sibling1-sibling2

How to share the data from parent to child?

Using props concept

To set State when one function is clicked we have to pass one callback function in that we need to set that state.

<button onClick={()=>{

                setState("Nikhil Thula");

            }}>Pass Data to B</button>

How to share the data from parent com to child comp using props

If we are sharing data from A comp to E comp then that is called Props Drilling.

Problems of passing data from one component other component like A to E.

1. Things will become complex as more components are nested
2. If any comp fails to receive the data then destination component will never receive the data
3. If intermediate does not require the data, then we should make data available to them.

Context API

To overcome above problems we use this topic

What happens is Parent component directly store data in Space(Context) and that is used by whom ever they want like if it want by B comp then it will take from space or E want then it will take from there.

For that we need to found out

1. How to create Context
2. How to store the data in the context
3. How to access the data from the context

Context Concept is suitable for sharing the data from parent comp to child at any level.

We can use prop drilling also but is some complex mostly uses context api.

Context API is simple technique to share data from parent comp to child comp.

1. Create the context .
2. import { createContext } from "react";
3. let myfirstContext=createContext();
4. export default myfirstContext;

2. myfirstContext provides predefined component called Provider

So that we can store the data inside the context it takes one props name value fixed name.

import myfirstContext from "./context";

<div style={aStyles}>

            <h2>A Contex-API</h2>

            <myfirstContext.Provider value={100}>

<B/>

            </myfirstContext.Provider>

                   </div>

We can use any name inplace of myfirstContext but provideris fixed

=>Render Provider Component inside the parent component and pass the data to the provider component as a value props.

3. Make context available to all the child component of parent component by wrapping direct child inside the provider component.

<myfirstContext.Provider value={100}>

                <B/>

    </myfirstContext.Provider>

Here not only child like B C,D,E also access indirect child can also access.

1. Access the data from the context using useContext() hook.
2. import { useContext } from "react";
3. import myfirstContext from "./context";

function E(){

    let contextData=useContext(myfirstContext);

    return (

        <div style={eStyles}>

            <h2>E</h2>

            <p>Data: {contextData}</p>

        </div>

    );

}

If it is type input then use onChange, event,event.target.value etc

<div>

            <h2>HomeContext</h2>

            <input type="text" palceholder="Enter Text" onChange={(event)=>{

                setState(event.target.value);

            }}/>

            <button onClick={submitData}>Submit</button>

            <hr>

            </hr>

            <homeContext.Provider value={data}>

            <Details/>

            </homeContext.Provider>

        </div>

<homeContext.Provider value={data}>

            <Details/>

            <Navbar/>

     </homeContext.Provider>

What ever it is if we want to transfer data then we should wrap inside this provider tag like navbar and navbar import useContext ( let info=useContext(homeContext)).

If we are passing data from one component to multiple components then we wont use props concept we use context-api in case of parent to children

Here comes the children to parent

Parent

import Display from "./Display";

import { useState } from "react";

function Name(){

    let [state,setState]=useState();

    return (

        <div>

            <h2>Name-Parent</h2>

            <br>

            </br>

            <p>Child Data: {state}</p>

            <hr>

            </hr>

            <Display setData={setState}/> //here we can pass function also so we are passing setState in props for display in children when button clicked we are setting the setData to existing data.

        </div>

    );

}

export default Name;

Child

import { useState } from "react";

function Display({setData}){

    let [childData,setchildData]=useState("React JS");

    return (

        <div>

            <h2>Display-child</h2>

            <button onClick={()=>{

                setData(childData);

            }}>Pass Data To Parent</button>

        </div>

    );

}

export default Display;

Note: Props and context-api is not recommended for children to parent

For Children to parent and sbiling also redux concept is recommended.

Context-API

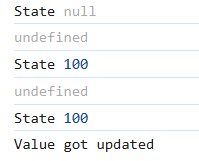
How to create context

How to store the data in the context

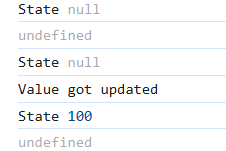
How to access the data from the context



1. First we update state then we updated normal variable



1. Here first we updated normal variable and then we updated state



When clicking on updateState we are setting the state to 100 and we are rendering it so var value is called again so first var value destroyed and again it is created as undefined this is the problem with storing values in normal variables.

Code:

function Ref(){

    let[state,setState]=useState(null);

    let value; //undefined

    let updateState=()=>{

        setState(100);

    }

    let updateValue=()=>{

        value="Value got updated";

    }

    let displayValue=()=>{

        console.log("State",state);

        console.log(value);

    }

    return(

        <div className='refContainer'>

            <h2>Ref Component: {state}</h2>

            <button onClick={updateState}>Update State</button>

            <button onClick={updateValue}>Update Value</button>

            <button onClick={displayValue}>Display</button>

        </div>

    );

}

Whenever components get re-rendered, normal variable will loose its current value.

State value will not be lost on re-rendering of component.

So now will create one more useState and try to store In that.

function Ref(){

    let[state,setState]=useState(null);

    let[data,setData]=useState(null);

    let value; //undefined

    let updateState=()=>{

        setState(100);

    }

    let updateData=()=>{

        setData(1000);

    }

    let updateValue=()=>{

        value="Value got updated";

    }

    let displayValue=()=>{

        console.log("State",state);

        console.log("Data",data);

        console.log(value);

    }

    return(

        <div className='refContainer'>

            <h2>Ref Component: {state}</h2>

            <button onClick={updateState}>Update State</button>

            <button onClick={updateData}>Update Data</button>

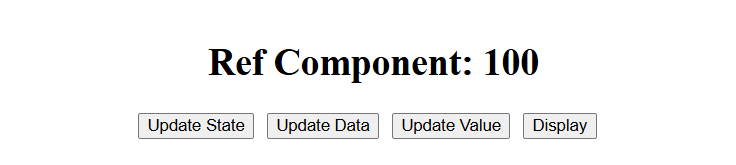
            <button onClick={updateValue}>Update Value</button>

            <button onClick={displayValue}>Display</button>

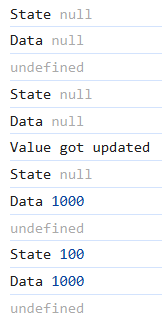
        </div>

    );

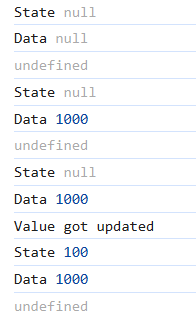
}



1. Clicking on display next update value next update data next update state



1. In this first Display next update data next update value next update state



Here also same value is changing to undefined so we have stored another state but which is not recommended because when we are using setState there we are rendering the data which decreases the productivity and we are not showing that value also any where on the screen which is unnecessary so not use state and variable.

Whenever you don’t want to show any data on the screen such data should not be stored in the state.

Re-rendered: application will have performance issues.

1. I want to a variable where I can store the data which I don’t want to show on the screen.
2. Data of that variable should not be loosed on re-rendering of the component.
3. If I update the variable value then mycomponent should not re-reneder.

Noramal variable state

| |

Rerender the component we should not store the data which we are not showing on screen

| |

Value will be loosed It will trigger re-rendering which we wont require.

Where to store the id value, so that we should not loose the id value on re-rendering component and whenever we update id value then component should not get re-render.

Use one hook called useRef();

Special reference🡪useRef() //having only one object in that only one property name,

{current:<data>}

Note: Rem this current

For that problems, use useRef() it is special reference given to store this kind of data.

useRef()

useRef is a hook in React

Whenever we call useRef() hook it returns one object and that object contains only one property I.e

{current: <data>}

Let myobj=useRef();

If we want we can update the value of current property any number of time, updation of current property value will not trigger re-rendering.

Eg code:

function Timer(){

    let [timer,setTimer]=useState(0);

    //var timerId;  //We cant store in this way due to rendering of setInterval.

    //let [timerId,setTimerId]=useState();  //This timerId is also should not be stored in this way as we not showing that id value on screen so use, useref();

    let timerId=useRef();   //timer={current:jundefined}

    //timerId.current=new value  (how to update timer value)

    const startTimer=()=>{

        timerId.current=(setInterval(()=>{

            //setTimer(timer+1);    //Dont use this due to multiple function before updating in UI it will render which gives problem so use functional argument.

            setTimer((prevTimer)=>{

                return prevTimer+1;

            })

        },1000));

    }

    const stopTimer=()=>{

        clearInterval(timerId.current); //usefull in stopping that sameid.

    }

    const resetTimer=()=>{

        setTimer(0);

    }

    return (

        <div style={{

            width:"500px",

            margin:"100px auto",

            textAlign:"center",

            boxShadow:"0 0 10px blue",

            padding: "15px"

            }}>

        <h2>Timer value is: {timer}</h2>

        <br></br>

        <button style={{ marginRight: "10px" }} onClick={startTimer}>Start</button>

        <button style={{ marginRight: "10px" }} onClick={stopTimer}>Stop</button>

        <button style={{ marginRight: "10px" }} onClick={resetTimer}>Clear</button>

        </div>

    );

}

useRef() is used to 1. store the data and other we can do 2.DOM manipulation.

DOM Manipulation using useRef()

1. Import useRef() hook
2. Call the useRef() hook in the component and assing null as default value
3. Add ref attribute to the JSX element and assign reference created by useRef() hook as a value to it.

Eg:

 let h2Ref=useRef(null);

<h2 ref={h2Ref}>DOM Manipulation using useRef() hook</h2>

<button ref={buttonRef} onClick={changeContentStyle}>Change Content and Style</button>

const changeContentStyle=()=>{

        h2Ref.current.innerText="Yes,DOM got Manipulated using useRef concept"

        h2Ref.current.style.color="blue";

        buttonRef.current.style.color="green";  //Every where current is common.

        buttonRef.current.style.padding="10px";  //Every where current is common.

    }

Types of hooks

1. useState() –to store data in state,setState

2. useRef()—to store data without rerendering and deleting and for DOM manipulation

3. useContext()—while transferring from parent to child component if two or more component is there then we use, useContent to store the data.

4. useEffect()—Youtube videos

5. useReducer()

useReducer() Hook

useReducer is same as useState but when there is huge data to store then we use useReducer mostly for less data storage use useState only.

state:store+manage the data

It is one of the hook in React

It is used to handle complex data of the component

It has same purpose of useState

How to store and manage the data using useReducer() hook

1. Import useReducer() hook
2. Call the useReducer() hook

useReducer() is also having two elements same as useState like (state,setState) those are state,dispatch.

let[state,dispatch]=useReducer();

1. useReducer() hook takes two argument and they are

1.function(reducer)

2.Initial Data

let[state,dispatch]=useReducer(reducerFn,initialData);

If we want to update state then we have to call reducerFn, we can not reducerFn directly for that we need to call dispatch i.e flow will be

Dispatch will call reducerFn then it updates state.

useState(null), it will take only one value but useReducer takes two useRducer(reduceFn,initialData).

We can create reduceFn any where like outside main fn or outside project also.

Eg:

import { useReducer } from "react";

const reducerFn=()=>{

    return "It is modified";

}

function UseReducer(){

    //let[state,dispatch]=useReducer(reducerFn,{city:"Delhi"});

    let[state,dispatch]=useReducer(reducerFn,100);  //reducerFn is userdefined we can create at any where outside fun or any where and 100=initialvalue which stores in state.

    let modifyState=()=>{

        dispatch();

    }

    return (

        <div style={{padding:"100px"}}>

             <h2>UseReducer() Hook: {state}</h2>

            {/\* <h2>UseReducer() Hook: {state.city}</h2> \*/}

            <button onClick={modifyState}>update State</button>

        </div>

    );

}

Reducer is user defined function, which is used to update the state.

Reducer function takes one parameter and that parameter is object type.

Reducer function stores a special object in its parameter and i.e action object.

Reducer function whatever it returns that data will be updated in the state.

Inside the reducer function we have to implement logic to update state.

Dispatch is a pre-defined function

Dispatch is used to call reducer function

Dispatch will re-render the component

Dispatch can take one argument and that is argument action object.

Action object is a special object in react

Action object should have one mandatory property and i.e type and whose value should be some unique string across the application.

{

type:”<unique string”>

data:””

}

Apart from type property, we can add any number of properties inside the action object.

-Here type key is mandatory with that name only and it should contain unique string remaining we can take what ever we want.

In useReducer() we should understand:

1. state
2. dispatch
3. reducerFn
4. action object

=>Here reducerFunction takes two parameters i.e state and action obj where state gives old state value

const counterReducer=(state,action)=>{

    return "Hello";

}

export default counterReducer;

=>Hello will be updated in state and from state we can fetch old state value.

import { useReducer } from 'react';

import './CounterRed.css'

import counterReducer from './counterReducer';

function CounterRed(){

    let[state,dispatch]=useReducer(counterReducer,{count:0});

    const incCount=()=>{

        dispatch();

    }

    const decCount=()=>{

        dispatch();

    }

    return(

        <div className="reducerCounter">

            <h2>Counter App with useReducer() Hook: {state.count}</h2>

            <button onClick={incCount}>inCount</button>

            <button onClick={decCount}>deCount</button>

            <button>Clear</button>

        </div>

    );

}

export default CounterRed;

const counterReducer=(state,action)=>{

    //<h2>Counter App with useReducer() Hook: {state.count}</h2> //here state.count is used so should match state.count again rendering.

    //Here state.count=0

    return {count:(state.count+1)}; //here key count should be same name as it is used in CounterRed in fetching state.count, so should be same.

    //It return state.count at the end.

}

export default counterReducer;

Here when we click on increment and decrement both times we need to call dispatch it calls reducerfn, but in that we have written code for increment so here we have to use action object and by using It we need manage both increment and decrement logics.

Flow:

Here first data is getting from state, we need to call dispatch, then dispatch will call reducerFn and that updates the state again.

To manage both updated code is:

Inside CountRed function:

 const incCount=()=>{

        dispatch({

            type:"INC"

        });

    }

    const decCount=()=>{

        dispatch({

            type:"DEC"

        });

    }

We need to add type in dispatch I.e action object type is fixed.

Inside counterreducer.js file

const counterReducer=(state,action)=>{

    //<h2>Counter App with useReducer() Hook: {state.count}</h2> //here state.count is used so should match state.count again rendering.

    //Here state.count=0

    if(action.type=="INC"){

        state={

            count:state.count+1//here key count should be same name as it is used in CounterRed in fetching state.count, so should be same.

        //It return state.count at the end.

        }

    }else if(action.type=="DEC"){

        state={     //need to write state to access count.

            count:state.count-1

        }

    }

    return state;

}

export default counterReducer;

If we are using multiple operations then use useReducer() bcz we have action object we can manage multiple operations if we want to perform only one operation then use useState, eg: If we want to perform only increment use useState and if want to perform inc,dec use useReducer.

How to share the data btw the components using

Parent to child

1. props
2. context api

child to parent and sibling to sibling those are not recommended we use redux

=>In complex situations like multiple parenting and sibling it will be difficult so use redux.

Redux is separate library it is not from react.

Process of managing this state is called statemanagement or datamanagement we can use props, context api or redux for state management.

1. Create a redux store
2. Make redux store available to each and every component of react application
3. Any component can store the data in redux store
4. Any component can access the data from redux store.

How to implement Redux in react application?

1. Install redux library

npm install redux

1. Create a redux store.

Import and use that function

1. Make redux store available to each and every component of the application

React-redux is another library which is used to integrate react with redux.

Redux

----------

Redux is a javascript library and pattern

Purpose of Redux

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

To manage the state(data) of the application.

Using redux we can share the data between the components irrespective of the relationship.

Mostly redux is used for sibling to sibling and child to parent for parent to child it will be context api or props.

Steps:

1. Create the component

Created NameParent and DetailsChild component

1. Create a redux store.
2. Install redux
3. Import legacy\_createStore()
4. Call the above function
5. import {legacy\_createStore as createStore} from 'redux'
6. let myReduxStore=createStore();
7. export default myReduxStore;
8. Make redux store available to all the components of the application
9. Install react-redux
10. Import provider component react-redux library
11. Render app comp inside the provider component.(Main.jsx)
12. Pass store(fixed value same as contex api we use, myfirstcontext.Provider value={}) props to the provider component and assign redux store as a value to it.
13. import {Provider} from 'react-redux'
14. import myReduxStore from './redux/store/Store.js'
15. //ReactDOM.render(<App/>,document.createElement('root'));//upto 17
16. const root = ReactDOM.createRoot(document.getElementById('root'));
17. root.render(
18. <BrowserRouter>
19. <Provider store={myReduxStore}>
20. <App/>
21. </Provider>
22. </BrowserRouter>
23. ); //parent component
24. Store the data from component into the redux store.

passing action object







action

Component



Redux store

reducer

passing action object

dispatch



Redux Pattern

The flow is first data is sent to action object from component that action object is passed to dispatch function from there it will through the reducer fn and there it will store in redux.

Same as reducer function it has action object and dispatch will take care of reducerfn calling and it will update the state.

1. Create a reducer function.
2. Register reducer function with redux store.

import myReducer from '../reducers/myReducer';

let myReduxStore=createStore(myReducer);

Note: Whenever we start react application, internally redux will call reducer() function, at whatever data reducer will return it will store by default inside redux store as a initial data.

As in main.jsx we are rendering app inside provider component and passted store prop so it will that createStore in that store we are passing reducerfunction as argument so it will called when the react application starts and stores the data in state to store.

const initialData={

    name:""

}

const myReducer=(state=initialData,action)=>{

    return state;

}

export default myReducer;

Now data in store is {name:””} object.

applyMiddelware:

applyMiddleware is a helper function provided by Redux that lets you enhance the store with custom logic. It allows you to insert middleware functions into the Redux dispatch process, so that each action can be intercepted, modified, logged, delayed, or used for async operations **before** reaching the reducer.

Every time you dispatch an action, redux-logger will intercept it, print useful logs in the console, and then let it pass to the reducer.

**Without middleware:**

* Action → Reducer → Store updates state

**With middleware:**

* Action → **Middleware (can log, modify, block, or trigger async work)** → Reducer → Store updates state

To install redux logger npm install redux-logger

dispatch → store → reducer → state update

Upto now we have components, reducer function, redux store, we need to create action object and dispatch function.

Creation of action object

Onclick of submit

var actionObj={

type:”NAME,

nameComp:inputRef.current.value,

}

Dispatch(actionObj);

//actual code

import { useRef } from "react";

import { useDispatch } from "react-redux";

function NameParent(){

    let inputRef=useRef(null);

    let dispatch=useDispatch();

    const getData=()=>{

        //create a action object

        var actionObj={

            type:"NAME", //type is fixed here

            nameComp:inputRef.current.value,

        }

        /\* console.log(inputRef.current);

        console.log(actionObj.nameComp); \*/

        dispatch(actionObj);    //dispatch automatically calls stores(dispatch → store → reducer → state update)

    }

    return(

        <div style={{padding:"30px"}}>

            <h2>Name Component</h2>

            <input ref={inputRef} type='text' placeholder="Add some data"/>

            <button onClick={getData}>Submit</button>

        </div>

    );

}

export default NameParent;

<input ref={inputRef} type='text' placeholder="Add some data"/>

In this we are storing inputRef.current=complete input tag to get value we are using inputRef.current.value in action object.

this dispatch calls store there it calls reducer function in that if action.type==”NAME” then we have to store the action.nameComp to initialData i.e state.name=action.namecomp.

const initialData={

    name:"Nikhil"

}

const myReducer=(state=initialData,action)=>{

    if(action.type=="NAME"){

        state={

            name:action.nameComp,   //value stored from input field to store

        }

    }

    return state;

}

export default myReducer;

Now we have created everything left with accessing

1. Accessing the stored data from redux store.

We can access data inside state by useSelector which takes one parameter and returns complete state

Eg: { name:”Nikhil”} complete state it returns.

import { useSelector } from "react-redux";

function DetailsChild(){

    let data=useSelector((storedata)=>{

        return storedata;

    });

    return(

        <div style={{padding:"30px"}}>

            <h2>Details Component</h2>

            <p style={{color:"green",fontWeight:"bold", fontSize:"20px"}}>{data.name}</p>

        </div>

    );

}

export default DetailsChild;

=> we have passed sotredata that is React-Redux internally calls your function and passes the **current store state**. Eg:{name:””}

useSelector is coming from react-redux library.

Instead of directly creating action object now we are creating actionCreator which stores the data from actionCreater to action object.

Nothing but we are separately storing the action object in other file

const cityActionCreator=(cityValue)=>{

    return{

        type:"CITY",

        cityName:cityValue,

    }

}

export default cityActionCreator;

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

import { useState } from "react";

import { useDispatch } from "react-redux";

import cityActionCreator from "../actions/CityAction";

function City(){

    let [city,setCity]=useState("");

    let dispatch=useDispatch();

    const getCity=()=>{

        //This has been wrote separtely in CityAction.js File.

        /\* var cityAction={

            type:"CITY",

            cityComp:city,

        } \*/

        var cityAction=cityActionCreator(city);

        dispatch(cityAction);

    }

    return (

        <div style={{padding:"30px"}}>

            <h2>City Component</h2>

            <select onChange={(event)=>{

                setCity(event.target.value);

            }}>

                <option>Select the city</option>

                <option>Hyderabad</option>

                <option>Bangalore</option>

                <option>Chennai</option>

                <option>Gujarat</option>

                <option>Delhi</option>

            </select>

            <button onClick={getCity}>Submit</button>

        </div>

    );

}

export default City;

updated reducer function

const initialData={

    name:"Nikhil",

    city:"Warangal",

}

const myReducer=(state=initialData,action)=>{

    if(action.type=="NAME"){

        state={

...state,   //It will copy before state object by using spread operator.

            name:action.nameComp,   //value stored from input field to store

        }

    }else if(action.type=="CITY"){

        state={

            ...state,   //It will copy before state object by using spread operator.

            city:action.cityName,

        }

    }

    return state;

}

export default myReducer;

Note: We can create only one redux store for a application.

Whenever we refresh or reload application, redux data will be removed or it will be lost.

In products example always sending the request to server is not right idea for first time we need to send the request after getting the result we need to store that output in redux and from second time we can fetch from redux time saving and application works fast.

Redux is mostly used in child to parent and sibling to sibling and in parent to child we use mostly context api and props.

useParams, useSearchParams and Navigate, ProtectdRoutes

=>We can pass path parameters and query parameters to share the data between siblings.

=>In all products while we are fetching each card we can fetch each product detail by id, passing id through param.

Eg:

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

import './Cards.css'

function Cards({image,title,price,rating,id}){

    return (

        <div className="card">

                <img src={image} width="100%" height={200}/>

                <p>{id}</p>

                <h3>{title}</h3>

                <p>{price}</p>

                <p>{rating}</p>

                <Link to={`/proddetails/${id}`}>

                <button>Product Details</button>

                </Link>

        </div>

    );

}

export default Cards;

Here we are passing id as javascript code in link use {} and map that in app.js

      <Route path={'/proddetails/:id'} element={<ProductDetails/>}/>

To use that param in our component then we can use, useParams() hook, it returns one object.

Eg to use useParams;

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

function ProductDetails(){

    let {id}=useParams();

    return (

        <div>

            <h2>Product Details</h2>

            <p>Product Id: {id} </p>

        </div>

    );

}

export default ProductDetails;

That id has to match in app.jsx

useEffect(()=>{

        fetch(`https://fakestoreapi.com/products/${id}`)

        .then((res)=>{

            return res.json()

        })

        .then((data)=>{

            console.log(data);

        })

        .catch((error)=>{

            console.log(error);

        })

    },[]);

If we want to fetch any thing on mount stage i.e directly on app start use useEffect, and in fetch we need use res.json but in axios.get it will directly parse to json.

Query Params

If we want to pass parameters like query params

Eg: <http://localhost:3000/profile?name=nikhil>

We can get that query params in our component by using useSearchParams() it also give two arguments as output same as state.

Const[searchparams,setSearchParams]=useSearchParams()

Here searchparams is used for accessing all query params.

setSearchParams is used for adding query params and updating query params.

Eg:

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

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

function Profile(){

    let[searchparams,setSearchParams]=useSearchParams();

    const addQueryParams=()=>{

        setSearchParams({

            name:"nikhil",

            city:"Hyderabad",

        })

    }

    return (

        <div>

            <h2>Profile</h2>

            <button onClick={addQueryParams}>Add Query Params</button>

            <button onClick={()=>{

                setSearchParams({

                    name:"Vamshi",

                    city:"Mumabi",

                })

            }}>Update Query Params</button>

            <Link to="/assignment?name=vijay&city=delhi">

            <button>Switch to Assignment</button>

            </Link>

        </div>

    );

}

export default Profile;

Here setSearchParams is used for passing the query params as key value pair

If we want to use that passed query params then we have to use it by searchparams which is first arg In useSearch() we need to Import that useSearch where we want to use that query params

Eg’;

function Aassign(){

    let[state,setState]=useState("..........");

    let[searchparams]=useSearchParams();

    return(

        <div style={aStyles}>

            <h1>A</h1>

            <p>Data: {state}</p>

            <button onClick={()=>{

                setState("Nikhil Full Stack Developer");

            }}>To Transfer Data From A to E</button>

            <p>Query Parameters By useSearchParams: {searchparams.get("name")} {searchparams.get("city")} </p>

            <B info={state}/>

        </div>

    );

}

useNavigate()

If we want to change the path on onclick of any button like logout, login then we need to use hook called useNavigate() it will return one function called navigate.

Const navigate=useNavigate();

Navigate(“/”);

Eg:

function Logout(){

    const{logout}=useContext(AuthContext);

    const navigate=useNavigate();

    const handleLogout=()=>{

          logout();

          navigate("/");

    }

    return (

        <div>

            <h2>Are you sure you want to logout?</h2>

            <button onClick={handleLogout}>Logout</button>

        </div>

    );

}

<Navigate to=”/”/>

This Navigate tag is used for navigating i.e this will help in protecting routes directly we cant navigate to that link we can create one component and route inside or directly we can ex:

One fuction and use that in App.jsx

import { useContext } from "react";

import { AuthContext } from "./AuthProvider";

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

function ProtectedRoutes({children}){

    let {isLoggedIn}=useContext(AuthContext);

    if(isLoggedIn){

        return children;

    }else{

        return <Navigate to="/"/>

    }

}

export default ProtectedRoutes;

function App() {

  const {isLoggedIn}=useContext(AuthContext);

  return (

    <div className="app">

      <Navbar/>

      <Routes>

      <Route path={"/"} element={<Login/>}/>

      <Route path={"/home"} element={isLoggedIn?<Home/>:<Navigate to="/"/>}/>  {/\* we can write as below protected with separte fun or direct \*/}

      <Route path={"/products"} element={<ProtectedRoutes><Products/></ProtectedRoutes>}>

        <Route path={"electronics"} element={<ProtectedRoutes><Electronics/></ProtectedRoutes>}/>

        <Route path={"jewellery"} element={<ProtectedRoutes><Jewellery/></ProtectedRoutes>}/>

        <Route path={"mensclothing"} element={<ProtectedRoutes><MensClothing/></ProtectedRoutes>}/>

        <Route path={"womensclothing"} element={<ProtectedRoutes><WomesClothing/></ProtectedRoutes>}/>

      </Route>

      <Route path={"/counter"} element={<ProtectedRoutes><Counter/></ProtectedRoutes>}/>

      <Route path={"/imagecomp"} element={<ProtectedRoutes><ImageComp/></ProtectedRoutes>}/>

      <Route path={"/allproducts"} element={<ProtectedRoutes><AllProducts/></ProtectedRoutes>}/>

      {/\* <Route path={'/usernames'}  element={<UserNames/>}/> \*/}

      <Route path={"/reducerfn"} element={<ProtectedRoutes><CounterRed/></ProtectedRoutes>}/>

      <Route path={'/parenting'}  element={<ProtectedRoutes><A/></ProtectedRoutes>}/>

      <Route path={'/assignment'}  element={<ProtectedRoutes><Aassign/></ProtectedRoutes>}/>

      <Route path={'/context-api'}  element={<ProtectedRoutes><Acon/></ProtectedRoutes>}/>

      <Route path={'/redux'} element={<ProtectedRoutes><ReduxHome/></ProtectedRoutes>}/>

      <Route path={'/prodwithredux'} element={<ProtectedRoutes><AllProductsWithRedux/></ProtectedRoutes>}/>

      <Route path={'/proddetails/:id'} element={<ProtectedRoutes><ProductDetails/></ProtectedRoutes>}/>

      <Route path={"/profile"} element={<ProtectedRoutes><Profile/></ProtectedRoutes>}/>

      <Route path={"/logout"} element={<ProtectedRoutes><Logout/></ProtectedRoutes>}/>

      <Route path={"\*"} element={<PageNotFound/>}/>           {/\* This has to be last \*/}

      </Routes>

    </div>

  )

}

In this we need to route it inside protected component to check whether logedin or not or directly we can navigate like:

Element={isLoggedIn?<Home/>:<Navigate to="/"/>

Main goal is to restrict the opening of that url when we click on home,products,allproducts,counter links directly so that we are redirecting it to login page only.