Function Component

❖ Function Component

- Functional component ek JavaScript function ki tarah kam karta hai. Ek single argument ko as a props accept karta hai jo react element ko return karta hai.

- Functional component is just a simple javascript function; it accepts the data in the form of props and returns the react element.

❖ Function Component Props

- Normally we have use props to pass the data from parent component to child component.
- In function component it is same but when we want to receive the data into child component we pass the parameter as a props and receive the data.

❖ Function Component State

- In function component there is a default hook is available that is called useState.
- useState accepts two values first is the variable name that is given the default state value and second is updated variable value that we want to update into the runtime.
- For example

```
const [firstVariablename, secondVariableruntimeChange] = useState(“Default value”);
```

❖ useEffect

- When we want to display the data after component render then we use useEffect hook.
- useEffect works asynchronously.

```
import React, { useEffect, useState } from "react";
const FunctioncompouseEffect = () => {
    const [api, setApi] = useState(false);
    const [api2, setApi2] = useState(false)
    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/albums").then((res) =>
res.json()).then((response) => { console.log(response); })
    }, [])
    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/todos").then((res) =>
res.json()).then((response) => { console.log(response); })
    }, [api])
    useEffect(() => {
        fetch("https://jsonplaceholder.typicode.com/posts").then((res) =>
res.json()).then((response) => { console.log(response); })
        return (() => {
            console.log("unmount called");
        })
    }, [api2])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <button onClick={() => { setApi(!api) }}>Click</button>
                <button onClick={() => { setApi2(!api2) }}>Click</button>
            </div>
        </div>
    </>);
}
export default FunctioncompouseEffect;
```

- There are two ways to get the data after render the page.
- First is to give [] dependency it means when our component is render then only our data is displayed. It is same like a constructor.
- Second give the dependency of state [api] and setState on button click event once the button is click then the data is show otherwise when our component render or re rendered then our data is show at that time but once the button is click no of times the data is show.

- useEffect also display the data when our component is closed we write the data on return statement inside useEffect once the component is closed the data is display it is also called the clean up.

❖ useLayoutEffect

- useLayoutEffect is used when we want to print some data when component is render before it means it works synchronously. The data is always display first.
- The useLayoutEffect works similarly to useEffect but rather working asynchronously like useEffect hook, it fires synchronously after all DOM loading is done loading. This is useful for synchronously re-rendering the DOM and also to read the layout from the DOM. But to prevent blocking the page loading, we should always use useEffect hook.

```
import React, { useEffect } from "react";
import { useLayoutEffect } from "react";
const FunctionCompoUseEffectUseLayoutEffect = () => {
    useEffect(() => {
        // getData();
        async function data() {
            await fetch("https://jsonplaceholder.typicode.com/posts").then((res)
=> res.json()).then((response) => { console.log(response); })
        }
        data();
    })

    // async function getData(){
    //     let Alldata = await
fetch("https://jsonplaceholder.typicode.com/posts").then((res)=>res.json()).then((
response)=>{console.log(response);})
    //     return Alldata;
    // }
    useLayoutEffect(() => {
        console.log("called useLayoutEffect");

        fetch("https://jsonplaceholder.typicode.com/todos/").then((res) =>
res.json()).then((response) => { console.log(response); })
        return (() => {
            console.log("called");
            // fetch("https://jsonplaceholder.typicode.com/todos/").then((res) =>
res.json()).then((response) => { console.log(response); })
        })
    }, [])
    // useLayoutEffect(async () => {
    //     console.log("called useLayoutEffect");
```

```
    //     // Warning: useLayoutEffect must not return anything besides a
function, which is used for clean-up.

    //     // It looks like you wrote useLayoutEffect(async () => ...) or returned
a Promise. Instead, write the async function inside your effect and call it
immediately:


    //     await fetch("https://jsonplaceholder.typicode.com/todos/").then((res)
=> res.json()).then((response) => { console.log(response); })
    //     return (() => {
    //         console.log("called");
    //         // fetch("https://jsonplaceholder.typicode.com/todos/").then((res)
=> res.json()).then((response) => { console.log(response); })
    //     })
    // }, [])
    return (<>
        <div className="row">
            <div className="col-6 offset-6">
                <h3>difference bten useEffect vs useLayoutEffect</h3>
                <p>useLayoutEffect is used when we want to print some data when
component is render before it means it works synchronously. The data is always
display first.</p>
                <p>The useLayoutEffect works similarly to useEffect but rather
working asynchronously like useEffect hook, it fires synchronously after all DOM
loading is done loading. This is useful for synchronously re-rendering the DOM and
also to read the layout from the DOM. But to prevent blocking the page loading, we
should always use useEffect hook.</p>
            </div>
        </div>
    </>);
}

export default FunctionCompoUseEffectUseLayoutEffect;
```

- Here use fetch api in both the useEffect and useLayoutEffect first display useLayoutEffect Api data then display useEffect API Data.

- In useEffect we use async await function but it is not use in useLayoutEffect when we want to use it gives the warning because useLayoutEffect works synchronously async await returns the promise.

```
// Warning: useLayoutEffect must not return anything besides a function, which is
used for clean-up.

    //     // It looks like you wrote useLayoutEffect(async () => ...) or returned
a Promise. Instead, write the async function inside your effect and call it
immediately:
```

❖ Use Memo

- Use memo returns the memorized value. When give dependency it always works in specific dependency.

- In general terms when we have multiple of data in Array or something else when data is count always it starts from first index or value, but when we use useMemo hook it stores the previous value data and accordingly the calculation or task is perform. So basically if we imporove our code performance and if we have bunch of data at that time we use useMemo.

- Use memo returns the memorized value. When give dependency it always works in specific dependency.

- Jb hamare pass bahot sare data hote hai array me ya koi or jagah jb hamara component rerender hota hai tab hamari value or index first se start hoti hai but jab hum useMemo ka use karte hai tab vo previous value ke basis pe next value ko calculate karta hai. So basically hum code ke performance ko improve karne ke liye jab hamare pass bunch of data ho tub hum uska use karte hai.

```jsx
import React, { useMemo, useState } from "react";
const FactorialusingUseMemo = () => {
    const [number, setNumber] = useState(1);
    // const [inc, setInc] = useState(0);
    const factorial = useMemo(() => factorialOf(number), [number]);
    const onChange = event => {
        setNumber(Number(event.target.value));
    };
    // const onClick = () => setInc(i => i + 1);
    return (
        <div>
            Factorial of
            <input type="number" value={number} onChange={onChange} />
            is {factorial}
            {/* <button onClick={onClick}>Re-render</button> */}
        </div>
    );
}

function factorialOf(n) {
    console.log('factorialOf(n) called!');
    return n <= 0 ? 1 : n * factorialOf(n - 1);
}

export default FactorialusingUseMemo;
```

❖ Use callback

- Use callback always returns a memorized function.
- For example we have multiple functions and each function have some different functionality perform so when we want to perform some functionality on some click event into single component then we use useCallback hook.

- Jab bhi hum callback function use krate tab memorized function returns karta.
- For example Jab hum Multiple function banate hai but particular function button click pe call karva na ho tab hum dependency dete hai but jab hame Multiple function and multiple button pe call karvana ho tab hu callback me same dependency deke use kar sakte hai.

❖ Use Context

- Use Context provide three functionalities
    1. Create context
        - Create context is used to create a function.
    2. Provider
        - When create a function in using use Context the provider helps to pass the data from one component to another component.
    3. Consumer or useContext
        - In old react version if we want to receive the data use consumer but in latest version we use hook called use Context it is used to receive the data from whenever our context is created.
- The main difference between props and useContext is that props is a react functionality to receive the data but useContext is a by default hook it is used to receive the data whenever the create context is created.

❖ Use Reducer

Use reducer is used when we have perform multiple operation into single function then we use Use reducer.

❖ Use Imparative Handle

- useImperativeHandle is a React Hook that lets you customize the handle exposed as a ref.
- In React, data is passed from parent to child components via props, in what is known as unidirectional data flow. The parent component cannot directly call a function defined in the child component or reach down to grab a value for itself.
- In certain circumstances, we want our parent component to reach down to the child component, getting data that originates in the child component for its own use. We can achieve this type of data flow with the useImperativeHandle Hook, which allows us to expose a value, state, or function inside a child component to the parent component through ref. You can also decide which properties the parent component can access, thereby maintaining the private scoping of the child component.
- This hook is used when we want to call the function from child to parent then we use useImparative.
- Jab hame child component mese function ko pass karna parent me or usko call karvana hai tab hum Use imperative handle ka use karte hai.

❖ Use Ref

- Use ref hook is used to get the reference value that is not needed for rendering.
- Jab hame koi value get karvani hai reference ke through state or props ke bina tub hum useRef ka use karte hai.

❖ API

- API stands for Application Programming Interface. It is a medium that allows different applications to communicate programmatically with one another and return a response in real time.

❖ How to make API Calls in React JS

- There are three ways to call API in React JS.

1. XMLHTTPREQUEST

- In JavaScript, the XMLHttpRequest object is an API for sending HTTP requests from a web page to a server. It's a low-level API because it only provides a basic mechanism for making HTTP requests and leaves it up to the developer to parse the response, handle errors and manage the request's state.

```
import React, { useState } from 'react';

function Example() {
  const [data, setData] = useState(null);

  function handleClick() {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', 'https://api.example.com/data');
    xhr.onload = function() {
      if (xhr.status === 200) {
        setData(JSON.parse(xhr.responseText));
      }
    };
    xhr.send();
  }

  return (
    <div>
      <button onClick={handleClick}>Get Data</button>
      {data ? <div>{JSON.stringify(data)}</div> : <div>Loading...</div>}
    </div>
  );
}
```

2. Fetch API

- Fetch API is a modern, promise-based API for making HTTP requests in JavaScript. It provides a simple and flexible interface for making GET, POST, PUT, PATCH and DELETE requests and handling the response from the server.
- The use of Put and Patch is that when update the record then use Put and Patch.
- Difference between Put and Patch is that Patch update specific record but put request all records to update so Patch is more reliable to use.

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

function App() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(json => setData(json))
      .catch(error => console.error(error));
  }, []);

  return (
    <div>
      {data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}
    </div>
  );
}
export default App;
```

3. Axios API

- Axios is a popular JavaScript library for making HTTP requests, and it works great with React. Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations (create, read, update and delete), as well as handle the responses.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function App() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts')
      .then(response => {
        setPosts(response.data);
```

```
      })
      .catch(error => {
        console.error(error);
      });
  }, []);

  return (
    <ul>
      {posts.map(post => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
}

export default App;
```

❖ Redux

- Redux is an open-source JavaScript library for managing and centralizing application state.
- Basically if we want to pass the data from first component to last component then unnecessary we pass the data to all component that is not required this is called prop drilling.
- So when we don't want to send data in unnecessary component and if we send the data from one component to other component directly then we use Redux.
- There are two ways to redux we have : 1) React Redux and 2) Redux Toolkit

❖ React Redux

Steps to Perform React Redux

Index.js from src folder

```
import React from 'react';
import ReactDOM from 'react-dom/client';
// import './index.css';
```

```
import App from './App';
// import reportWebVitals from './reportWebVitals';
import "./custom.css"
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <App />
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
// reportWebVitals();
```

App,js from src folder

```
import './App.css';
import { Provider } from 'react-redux';
import store from './store';
import Cart from './component/Cart';


function App() {
  return (
    <Provider store={store}>
      <Cart />
    </Provider>
  );
}

export default App;
```

Create a Store file in src folder

```
import { legacy_createStore as createStore } from 'redux';
import { cartReducer } from "./reducers/cartReducer";

const store = createStore(cartReducer);

export default store;
```

Create a reducer folder inside reducer folder create cartReducer file

```
import { ADD_ITEM, DELETE_ITEM } from "../actionTypes/actionTypes";
const initialState = {
    numOfItems: 0,

}
export const cartReducer = (state = initialState, action) => {
    switch (action.type) {
        case ADD_ITEM:
            return {
                ...state,
                numOfItems: state.numOfItems + 1,
            };
        case DELETE_ITEM:
            return {
                ...state,
                numOfItems: state.numOfItems - 1,
            };
        default:
            return state;
    }
}
```

Create actionTypes folder and create actionTypes file

```
export const ADD_ITEM = "ADD_ITEM";
export const DELETE_ITEM = "DELETE_ITEM";
```

Now we move on to UI part create Cart.jsx inside Component folder

```
import {useDispatch, useSelector} from "react-redux";
import { addItem, deleteItem } from "../actions/cartAction";
const Cart = () => {
const state = useSelector((state)=>state);
const dispatch = useDispatch();
return ( <>
        <div className="cart">
            <h2>Number of items in cart: {state.numOfItems}</h2>
            <button onClick={()=>{dispatch(addItem());}} className="green">Add item in
cart</button>
            <button disabled={state.numOfItems>0 ? false:true}
onClick={()=>{dispatch(deleteItem());}} className="red">Remove Item in cart</button>
        </div>
    </> );
}

export default Cart;
```

Now create a Action folder inside action folder create cartAction file

```
import { ADD_ITEM, DELETE_ITEM } from "../actionTypes/actionTypes";
const addItem = () => {
    return {
        type: ADD_ITEM,
    };

};
const deleteItem = () => {
    return {
        type: DELETE_ITEM,
    };
};
export { addItem, deleteItem }
```

Now go to again reducer and calculate the state for increment or decrement

The value is come from Cart again and using useSelector hook we receive the current state value.

```
const state = useSelector((state)=>state);
```

❖ Redux toolkit

Steps to perform Redux toolkit

Index.js in src folder

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { Provider } from 'react-redux';
import store from './store';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store}>
    <App />
  </Provider>
```

```
);
```

store.js

```
import { configureStore } from "@reduxjs/toolkit";
import reducer from "./reducer";

const store = configureStore({
    reducer: {
        count: reducer
    }
})
export default store;
```

reducer.js

```
const initial=0;
const reducer = (state = initial, action) => {
    switch (action.type) {
        case "inc":
            return state + 1;
        case "insert":
            return state + 1;
        case "dec":
            return state - 1;
        default:
            return state;
    }

}

export default reducer;
```

counterSlice.js

```
import { createSlice } from '@reduxjs/toolkit'
import { PayloadAction } from '@reduxjs/toolkit'

// export const CounterState = ""

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
```

```
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

// Action creators are generated for each case reducer function
export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

- Either if use createSlice then no required to create reducer separate or create reducer if create configureStore.

App.js

```
// import logo from './logo.svg';
import './App.css';
import { useDispatch, useSelector } from 'react-redux';
import { decrement, increment } from './action';

function App() {

  const dispatch = useDispatch();
  const counter = useSelector(state => state.count)
  return (
    <div className="App">
      <button onClick={() => { dispatch(increment()) }}>+</button>
      <h1>{counter}</h1>
      <button onClick={() => { dispatch(decrement()) }}>-</button>
      <button onClick={() => {
        dispatch(() => { return { type: "inc" } })
      }}>+</button>
      <h1>{counter}</h1>
      <button onClick={() => { dispatch({ type: "inc" }) }}>-</button>

    </div>
  );
}

export default App;
```

action.js

```
export const increment = () => {
    return { type: "inc" }
}

export const decrement = () => {
    return { type: "dec" }
```

```
}
```

Again go to reducer and calculate the increment and decrement the state value and store previous state value.

Using useSelector hook receive the state value.

```
  const counter = useSelector(state => state.count)
```

❖ Redux Project

Index.js in src folder

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import { RouterProvider } from 'react-router-dom';
import 'mdb-react-ui-kit/dist/css/mdb.min.css';
import "@fortawesome/fontawesome-free/css/all.min.css";
import { Provider } from 'react-redux';
import CustomStore from '../src/Store/CustomStore.jsx';
import MainRouter from './Mainroutes.jsx';
// import App from './App';
// import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={CustomStore}>
    <RouterProvider router={MainRouter} />
  </Provider>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
// reportWebVitals();
```

Create CustomStore file in store folder

```
import rootReducer from '../Reducer/index';
import { legacy_createStore as createStore, applyMiddleware } from 'redux'
import { thunk } from 'redux-thunk';

const store = createStore(rootReducer, applyMiddleware(thunk));
export default store;
```

thunk is basically use to handle the async await. It is used for handle the http request.

rootReducer is the main reducer file here we define all reducer that is created.

Create index.jsx inside the reducer folder

```
import { combineReducers } from "redux";
import productReducer from "./Product";

const rootReducer = combineReducers({
    products: productReducer,
});
export default rootReducer;
```

Product.jsx inside the reducer folder

```
import {
    FETCH_PRODUCTS
} from '../Action/types';

const initialState = {
    items: [], // Initial empty array for products
    cart: [], // Initial empty array for cart
    productDetail: null,
};

const productReducer = (state = initialState, action) => {
    switch (action.type) {
        case FETCH_PRODUCTS:
            return {
                ...state,
                items: action.payload,
            };
        default:
            return state;
    }
};
```

```
export default productReducer;
```

Create Action folder create types.jsx file

```
export const FETCH_PRODUCTS = 'FETCH_PRODUCTS';
```

Then come with the UI Part create a HeaderCompo, Home and accessstor component and create MainRoutes for router.

HeaderCompo.jsx

```
// import { useDispatch, useSelector } from 'react-redux';
import React, { useEffect, useState } from 'react';
import {
    MDBContainer,
    MDBNavbar,
    MDBNavbarBrand,
    MDBNavbarToggler,
    MDBIcon,
    MDBNavbarNav,
    MDBNavbarItem,
    MDBDropdown,
    MDBDropdownToggle,
    MDBDropdownMenu,
    MDBDropdownItem,
    MDBCollapse,
} from 'mdb-react-ui-kit';
import { Link } from 'react-router-dom';


const HeaderCompo = () => {
    const [showBasic, setShowBasic] = useState(false);
    // const cart = useSelector((state) => state.products.cart); // Get the cart items from
Redux store

    const menuItems = [
        {
            title: 'Home',
            url: `/`,
        },
        {
            title: 'Shop',
            url: `/shop`,
        }, {
            title: 'about',
```

```
            url: `/about`,
        }, {
            title: 'Access Store',
            url: `/accessstore`,
        },
        {
            title: 'Services',
            url: '/services',
            submenu: [
                {
                    title: 'web design',
                    url: 'web-design',
                },
                {
                    title: 'web development',
                    url: 'web-dev',
                },
                {
                    title: 'SEO',
                    url: 'seo',
                },
            ],
        }
    ];

    const MenuData = menuItems.map((data, index) => {
        if (data.submenu != undefined) {
            var submenudata = data.submenu.map((submenu, index) => {
                return <MDBDropdownItem key={index} link>{submenu.title}</MDBDropdownItem>;
            });
        }
        if (data.submenu == null) {
            return (
                <MDBNavbarItem key={index}>
                    <Link className='nav-link' to={data.url}>
                        {data.title}
                    </Link>
                </MDBNavbarItem>
            );
        } else {
            return (
                <MDBNavbarItem key={index}>
                    <MDBDropdown>
                        <MDBDropdownToggle tag='a' className='nav-link' role='button'>
                            {data.title}
                        </MDBDropdownToggle>
                        <MDBDropdownMenu>{submenudata}</MDBDropdownMenu>
                    </MDBDropdown>
                </MDBNavbarItem>
            );
        }
    });

    return (
        <>
            <MDBNavbar expand='lg' light bgColor='light' className='sticky-top' >
                <MDBContainer fluid>
                    <MDBNavbarBrand href='#'>Brand</MDBNavbarBrand>

                    <MDBNavbarToggler
                        aria-controls='navbarSupportedContent'
```

```
                        aria-expanded='false'
                        aria-label='Toggle navigation'
                        onClick={() => setShowBasic(!showBasic)}
                    >
                        <MDBIcon icon='bars' fas />
                    </MDBNavbarToggler>

                    <MDBCollapse navbar show={showBasic}>
                        <MDBNavbarNav className='ms-auto w-auto mb-2 mb-lg-0 '>
                            {MenuData}

                            <MDBNavbarItem>
                                <Link className='nav-link btn btn-primary text-light'
to='/login'>
                                    Login
                                </Link>
                            </MDBNavbarItem>

                        </MDBNavbarNav>
                    </MDBCollapse>
                </MDBContainer>
            </MDBNavbar>
        </>
    );
};

export default HeaderCompo;
```

MainRoutes.jsx

```
import { createBrowserRouter } from "react-router-dom";
import HeaderCompo from "./CommonCompo/HeaderCompo";
import Home from "./Pages/Home";
import Accessstore from "./Pages/Accessstor";

const MainRouter = createBrowserRouter([
    {
        path: "/",
        element: <>
            <HeaderCompo />
            <Home/>
        </>
    },
    {
        path:"/accessstore",
        element:<>
            <HeaderCompo/>
            <Accessstore/>
        </>
    }
])
export default MainRouter;
```

Home.jsx inside the pages folder

```jsx
import { useEffect } from "react";
import { useDispatch } from "react-redux";
import { fetchProducts } from "../Action";

const Home = () => {
    const dispatch = useDispatch();
    useEffect(() => {
        dispatch(fetchProducts());
    }, [dispatch]);
    return (<>

    </>);
}

export default Home;
```

fetchProducts is define in the index.jsx file inside the Action folder.

```jsx
import API from '../http-common';
import { FETCH_PRODUCTS } from './types';



export const fetchProducts = () => {
    return async (dispatch) => {
        try {
            const response = await API.get('/products');
            // const response = await API.get('/products');
            // fetch
            // axios
            // console.log(response.data); // Check the data returned by the API
            dispatch({
                type: FETCH_PRODUCTS,
                payload: response.data,
            });
        } catch (error) {
            console.log('Error:', error);
        }
    };
};
```

The common API is define in the http-common.jsx that is in the src folder.

http-common.jsx

```jsx
import axios from 'axios';

export default axios.create({
    baseURL: 'https://dummyjson.com/',
    headers: {
        'Content-type': 'application/json',
```

```
    },
});
```

Then again it goes to the reducer(Product.jsx) inside the reducer folder and store the previous fetch data.

Accessstore.jsx

```
import { useSelector } from "react-redux";

const Accessstore = () => {
    const products = useSelector((state) => state);
    return (<>
        <div className="container">
            <div className="row">
                <div className="col">
                    {JSON.stringify(products)}
                </div>
            </div>
        </div>
    </>);
}

export default Accessstore;
```

Receive all the data in using the useSelector.