React (Day-7)

useEffect

1.// in no dependency array => useEffect is called on every render

```
useEffect(()=>{
    console.log("useEffect called");
});
```

2nd arg in useEffect is optional btw (dependency array)
↳ not mandatory

2.// if dependency array is empty = []
useEffect is called on initial render (just once)

```
useEffect(()=>{
    console.log("useEffect called");
}, []);
```

3.// if dependency array is [loginState]
useEffect is called everytime loginState is updated.

```
useEffect(()=>{
    console.log("useEffect called");
}, [loginState]);
```

useState

→ Never call useState hook outside your component. Infact, All hooks should be called inside body of a function component.

→ It has a purpose, to maintain local state of your functional component, so always call your hooks inside functional component and on top level of body.
↳ not nested inside some blocks/conditionals/loops

→ Always call your useState() on the start of function component (on the initial lines of code).

<u>Routing</u> (react-router-dom)

We'll be using createBrowserRouter and RouterProvider for define our routes.

Read reactrouter.com about different routers and their use cases.

'createBrowserRouter': is a new router that supports the new data APIs.

<BrowserRouter> doesn't support data APIs.

<u>Usage</u>:

```
const router = createBrowserRouter ([      ↳[route config]
   {
      path: '/',
      element: <App />
   },
   { path: '/about',
      element: <About />

   },
   {
      path: '/contact',
      element: <Contact />
   }
 ]);                                  [Provide the root config
                                       to our routerprovider]
root.render (<RouterProvider router = {router} />);
```

If there is a bad request / error in url entered, define it :-

```
const router = create...([
  {
    path : '/'
    element : <App />,
    ~~error : <Error />~~ errorElement : </Error>,
  }                 ↳ our custom error component
  :  :  :
  :  :  :
]);
```

Note: In the "custom <Error /> component that you have designed, make use of the useRouteError hook which will give more info about the error.

Ex:
```
import {useRouteError} from 'react-router-dom';
const Error = () => {
    const err = useRouterError();
        ↳ error object with all error info
          which we can use to display
          on our error component.
    return (
        <div> Error - {err.status}: {err.statusText}</div>
    );
}
```

The above component will now display the error status code and status text.

Ex: Error - 404: Not Found

→ How to keep a root component intact and render other components below it according to the route.

For ex: Navbar always stays intact, but body, about us & contact us changes below navbar according to the route.

→ So, put the header (Navbar) as the root route and put all the dependent routes as the children routes in the root route.

→ And render the children routes according to the specific child route using the <Outlet/> inside the root component.

Ex:
```
const App = () => {
    return (
        <div>
            <Header /> ↳ contains navbar
            <Outlet />
        </div>
    );
};

const router = createBrowserRouter([
    {
        path: '/',
        element: <App /> ↳ root route
        children: [
            {
                path: '/',
                element: <Body />
            },
            {
                path: '/about',
                element: <About />
            },
            {
                path: '/contact',
                element: <Contact />
            }
        ]
    }
]);
```

→ These are children routes of App component. Any of children can be replaced with the <Outlet /> according to the route of each child.

Navigating between routes in react application

Using anchor tag? NO!

using <a href='/'> ~~Home~~ </a>

↳ this will cause a whole refresh to the app and loads the whole app. Not at all recommended in React Apps.

Instead, use <Link to="/"> Home < Link>

↳ this will only refresh the component that needs to be changed according to the route.

{ can be used in navbar for each list item of navbar }

It changes the route & content without refreshing the page or loading the page.

This is where the concept of SPA (Single page Application) comes. We never reload our react page, we just shift between routes without reloading, which gives us the feeling of multi-page but ours will be SPA.

Client side Routing vs Server Side Routing

↓

But here we render everything from client side. Everything (content) is already present on client and we render the components for different routes without reloading the page.

Whole point of SPA is possible cuz of client side routing

(Modern)

↳ Navigating through different pages depends on server.

When we go to an endpoint (route) we would request data from server and then populate it on the page corresponding to that route.

(Traditional)