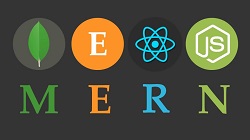
*MERN Full Stack Development*

**

**Class 9:**

**React Router**

**URL Parameters**

* We learned dynamic routes with react router, we created user’s listing page & nested user’s details page but we display details about user irrespective of who the user is, in real world application we would want to extract user id & do some with that id perhaps make an API call to fetch details about that user but we just display user id from URL in component.
* To extract route parameter we need to import a hook from react router package, in user details import {useParams} from react router Dom.
* Then call hook with component useparam() , the hook returns an object of key value pairs, assign it params is equal to useparam().
* This object contains key value pairs of dynamic params from current URL.
* The parameter we want to access is user ids; let’s store it in a userId.
* And its equal to params.userId(), user id on params object corresponds to dynamic segment we have specified in the route config in app.js
* <Route path=':userId' element={<UserDetails />} />
* See above code :UserId in app.js file, once we have user id we can render it as part of JSX, see text details about user & we render user id.

import { useParams } from 'react-router-dom'

const UserDetails = () => {

  const params = useParams()

  const userId = params.userId    //   <Route path=':userId'  app.js file

  return (

    <div>Details About User - {userId}</div>)

* See in browser & navigate to /users/1, we see text details about user 1.

C:\Users\CZ 3\Desktop\Untitled.png

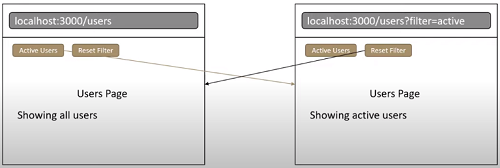
C:\Users\CZ 3\Desktop\Untitled.png

* Just change the id 1 to 2 or any other id in URL, we can see that id from URL, so this is how we get hold of URL params in component.
* Import useParams hook & invoke it & access property on returned object.
* We can destruct user Id in same line “const { userId} “ = useParam.
* It is called destruct & delete next line, see below, check in browser.

 const {userId} = useParams()

**Search Parameters**

* We have seen dynamic routes & URL params which represent part of URL that is dynamic, we also learned how to extract dynamic param within a component using use Param hook, URL params is not only way to add parameters to a route we can also add an optional query string.
* E.g. at end of current URL “localhost:3000/users/1?status=active, these parameters are called search params in react router.
* Add 2 buttons in users page 1 is actives & other is reset filter, by default page renders text showing all users, when click on active users we will add a search param called filter & set it to active.
* In the component we will display text showing active users if we click on reset we remove filter search param & revert the text.



* Step1: In users component add two buttons “active users” & “reset users”.
* Step2: On click of these buttons we need to add or remove search param, to deal with search params react router provides a hook called use search params, let import it {useSearchParams} this hook behaves similar to use State hook in react instead of storing state in memory though it is stored in URL, within component invoke hook “useSearchParams()”.
* This hook returns 2 values of which 1st is an object [mySearchPattern] which we are going to call mySearchPattern, 2nd value returned is a function to set search patterns name it “setSearchPattern”.

import { Outlet, useSearchParams } from 'react-router-dom'

const Users = () => {

  const [mySearchPattern, setSearchPattern] = useSearchParams()

  <Outlet />

           <div> <button>Active users</button>

                <button>Reset users</button>

* It is very similar to useState, Now using setSearchParams function we can add or remove parameter, 1st on click of active users button.
* In here we going to have an arrow function where we call “setSearch Params” & we pass in an object with 1 property called “status” & its value is “active”, next on click of reset users button call set search params again but pass an empty object see code below and test it in the browser.

<button onClick={() => setSearchPattern({ status: 'active'})}>Active users</button>

<button onClick={() => setSearchPattern({ })}> Reset users</button>

* In browser localhost:300/users, if we click on active users we can see in URL “?status=active” & if click on reset users search params is removed.

C:\Users\CZ 3\Desktop\Untitled.png C:\Users\CZ 3\Desktop\Untitled.png

* Now we need to do is check if “Status of search param” is set to active just display “Active users”, Step3, within component we make check on status parameter, create “const displayActiveUsers” & is = to Boolean value.
* To get hold of status parameter we use get function on search params.
* So call searchParams.get(), & what is pattern name it is “status”, we then compare the value with string active, so “displayActiveUsers” is true if we click on “active Users button” & false if we click on reset users button.
* Let’s use it to render JSX, in curly braces “displayActiveUsers” if it is true return “Displaying Active Users” & if its false return “Displaying All Users”.

  const [mySearchPattern, setSearchPattern] = useSearchParams()

  const displayActiveUsers = mySearchPattern.get('status') == 'active'

           {displayActiveUsers ? <h3> Displaying Active Users </h3>

                                : <h3> Displaying All Users </h3>}

* Save & see in browser, we can see “Displaying all users”, just click on active users button, we can see search param in URL & text is now change to “Displaying Active Users”, so our code search params is working fine.

C:\Users\CZ 3\Desktop\Untitled.png C:\Users\CZ 3\Desktop\Untitled.png C:\Users\CZ 3\Desktop\Untitled.png

* This is very common when we have to apply filters in a listing page.
* E.g. Website like “amazon” or any ecommerce website we have list of parameters on left side selecting a filter will update URL with a search param this lets us share link with others or even bookmark the link.
* Import useSeach params hook & invoke it, we get access to set search params function to set the value & search params object to get any value present in the URL.

**Relative Links**

* We are now learning another handy feature in react router which is relative, see the products page, if we click on “featured products”, the URL changes to “/products/featured & when we click “latest products” it changes to “/products/latest” & that is b/c of relative links have specified.

            <Link to='featured'>Featured Products</Link> |

            <Link to='latest'>Latest Products</Link>

Products.js

* A relative link is a link that does not start with a forward slash “/” & will inherit closest route in which they are rendered.

<Route path='order-summary' element={<OrderSummary />} />

App.js

* Since product page is rendered at “/products”, the “featured” link will append “/featured” to “/products” & “latest” link will append to “/latest /products” which is what is happening in the browser.’

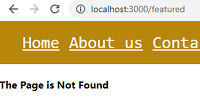
C:\Users\CZ 3\Desktop\Untitled.png

* Now if we wish to use absolute path, the path would be “/featured” or “/ latest”, this will construct path from the root of the app & not current URL.

            <Link to='/featured'>Featured Products</Link> |

            <Link to='/latest'>Latest Products</Link>

* Go to browser & click on “featured” or “latest” products link it navigates to /latest or /featured, links are breaking see below.



* To make this work with absolute links we need to add /products/featured & /products/latest, save see code & check in browser it should works now.

            <Link to='/products/featured'>Featured Products</Link> |

            <Link to='/products/latest'>Latest Products</Link>

* But let’s switch back to relative links, so this is relative links in react router.

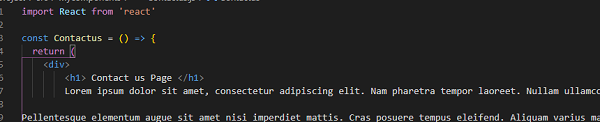
            <Link to='featured'>Featured Products</Link> |

            <Link to='latest'>Latest Products</Link>

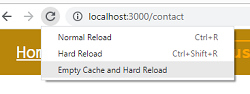
* Relative paths or relative links don’t start with a forward slash “/” & will inherit closest route in which they are rendered.
* They are very useful when we have to link deeply nested paths, absolute links are of course still valid but make more sense for components like the primary navigation bar see in the navbar.js file

**Lazy Loading**

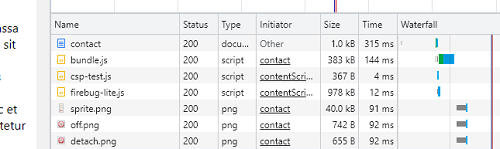
* Learn how to use lazy load routes with react router; lazy loading is a technique where components not required on the homepage.
* It can be split into separate code bundles & downloaded only when user navigates to that page it is as incrementally downloading application.
* It helps reduce initial load time thereby improving performance we are going to apply in contact us page, 1st add random text on contact us component go to lorem ipsum website & generate 30 paragraphs of text.
* Copy paste into contact us component & observe bundle size when entire app is loaded in initial load, open dev tools (inspect element).

****

* The select network tab & right click on “empty cache & hard reload” see.

****

**C:\Users\CZ 3\Desktop\Untitled.png**

****

* We can see size and time in mili seconds to load, now let’s lazy load contact us page & observe difference, to lazy load a route we need to use dynamic imports & react suspense, step 1 comment out import statement for contact us page in app.js instead we use dynamic import syntax.

import Aboutus from './mycomponents/Aboutus';

// import Contactus from './mycomponents/Contactus';

* For dynamic imports we need a default export of component, in contact.js remove “export” keyword & at bottom put “export default Contactus”.

const Contactus = () => {

export default Contactus

* In app.js use react lazy & dynamic imports so first import “React” from “react” & then contact us component “const lazyContact= lazy().
* Lazy is a function that takes another function as argument, this argument calls a dynamic import, so “import ‘./component/contactus” folders.

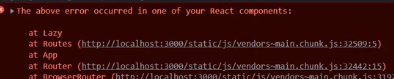
import React from 'react';

const LazyContactus = React.lazy(() => import('./mycomponents/Contactus'))

* Let move this to bottom a promise is returned by this dynamic import which is then converted into a module that contains a default exported react component which of course is our contact us component.
* So we done step1 now step 2 we include lazy component as part of route configuration pass in route see below “element = {LazyContact”}.

     <Route path='contact' element={<LazyContactus />} />

* Save & check in browser & right click & select “empty cache & hard reload.
* We can see in network tab size is reduced in size & time also has reduced in mili seconds we have improved our initial load time.
* But there is a problem if we trying navigate to contact us page we can see error in console ”a react component suspended while rendering but no fallback ui was specified to fix this error we have to use suspense component from react.



* In app.js wrapped lazy contactus component with react suspense.

<Route path='contact' element={

             <React.Suspense>

                 <LazyContactus />

             </React.Suspense> />

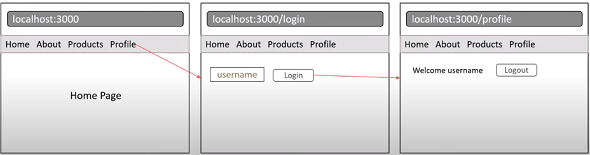
* And on react suspense we have to specify a fallback prop which can be any react element but now we put text “Loading page…”.

 <React.Suspense fallback='Loading page..'>

* Now go back to browser click network tab go to home page & right select “empty cache & hard reload and clear network tab & navigate to contacts page we can see the new chunk of code is downloaded.
* This chunk correspond to the contact us component in our application.
* We don’t see fallback loading page as size of this component is not large.
* This is how we can optimize initial load time with react router, react suspense & dynamic routes, when application grows in size & more 3rd party packages are installed the bundle bloats up causing initial load time to be very long.

**Authentication & Protected Routes**

* We going to see how to handle authentication with react router.
* When build web application we need to protect some routes in app from users who are not logged in, e.g. in an e-commerce site the product page might be publicly accessible whereas profile page or order history page requires that a user be logged in.
* The react router itself does not have a feature to protect routes but we can implement functionality without much difficulty.
* We going to learn how to protect routes using react router & context API from react, add new link in nav.js which should take us to profile page.
* But profile page is a protected route if we not logged in we will be redirected to login page, we can enter a username & click login button.
* We will then be able to view profile page which shows the logged in username “Welcome admin” & we also have logout button on click of which we will be navigated back to home page.



* Step 1 create a profile page include in the navbar & configure the corresponding route. So Profile is a simple functional component.
* In nav.js add new link “/profile” & in app.js configure new route “profile”.

App.js <Route path='profile' element={<Profile />} />

    Nav.js    <NavLink style={myNavLinkStyle}  to='/profile'>Profile</NavLink>

* Save file & test it out in browser just click on profile page to check.

C:\Users\CZ 3\Desktop\Untitled.png

* But we know this should be a protected route & only accessible if user is logged in, step 2: we need to implement functionality to figure out if user is logged in or not, we are going to maintain a user state variable & provide it to the entire component tree using react context.
* Create file Auth.js put in into utils folder but we will create folder later.
* Inside Auth.js create an auth context so at top import “createContext”.
* Then create AuthContext variable & call this createContext() & initial value is null, next define auth provider component.
* So “export const AuthProvider” it will be arrow function.
* And within this component we maintain user state & define function to login & logout, next import {useState} & create state variable.
* Make it [user,setUser] user variable & setUser function & initial value null.
* Now using setUser function to define log in & logout functions, so “make login” variable that receives user & we are going to call setUser passing in user similary “const logout” it does not receive any arguments we are going call setUser & passing in “null” see below code.

import { useState, createContext } from 'react'

const AuthContext = createContext(null)

export const AuthProvider = () => {

    const [user, setUser] = useState(null)

    const login = (user) => {

            setUser(user)

    }

    const logout = () => {

        setUser(null)

    }

* Now we are going to provide these values using AuthContext provider.
* So return “AuthContext.Provider” & this is going to wrap children props.
* Provider component needs a value prop & this is equal to an object with same key value pairs value={{user, login, logout}}.
* Make sure to destructor the children props, finally define a function that returns value of “auth context” so export const useAuth = arrow function where we return useContext() & passing in “AuthContext”.
* Make sure that to import useContext hook at top see code below.

import { createContext, useState, useContext } from 'react'

export const AuthProvider = ({children}) => { (new code)

 setUser(null)

    }

(new code)

    return (

        <AuthContext.Provider value={{ user, login, logout}}>

            {children}

        </AuthContext.Provider>

    )

}

export const useAuth = () =>{

      return useContext(AuthContext)

}

* Now that we have auth context provider, go to app.js.
* And we wrap entire component tree within AuthProvider & make sure import {AuthProvider} at top & save and see the code below.

App.js

import { AuthProvider } from './mycomponents/Auth';

<div className='App'>

       <AuthProvider />

         <Nav />

         <Routes>

(end page)

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

         </Routes>

        <AuthProvider/>

* Now have access to user so login & logout functions are available throughout our application so we proceed with step3.
* In Step 3: let’s implement a login route, create login.js a functional component & this needs to accept an email so import “useState”.
* Create a new state variable “user” & “setUser” & it takes empty string as initial value this state variable should be tied to an input element.
* Use html input with place holder takes email address from user & onchange we get hold of event & called setUser passing in event.target. value, next add login button & onclick of this button we assign a function called “userLogin()”, define a function “const userLogin()” assign to “=>” function & inside this we need to call login function from auth context.
* So call “useAuth” which is to be imported from auth file & assign return value to a const called “auth” but not put this into userLogin function.
* Then within userLogin call “auth.login” passing in “user”, once we set a username which is actually an email address we navigate the user to home page for that use “useNavigate” hook, so import useNavigate from react router dom & inside component assign “const navigate = useNavigate().
* And then with “userLogin()” just navigate to the “./” to the root.

import React from 'react'

import {useNavigate} from 'react-router-dom'

import { useAuth } from './Auth'

const Login = () => {

    const [user, setUser] = useState('')

    const auth = useAuth()

    const navigate = useNavigate()

    const userLogin = () => {

        auth.login(user)

        navigate('./')

    }

    return (

        <div>

            <label>

                Email: {''}

                <input type='text' placeholder='Enter Email...' onChange={(e) => setUser(e.target.value)} />

                <button onClick={userLogin}>Login </button>

            </label>

        </div>

    )

}

export default Login

* Finally add a login button in nav.js, the button should only be shown if user is not logged in, so once again get hold of “auth context”.
* “Const auth = useAuth() & import {auth} from “./auth.js”, below in JSX after profile navLink use curly braces “if not auth.user” means user is not logged in means render a link login link “<Login />” this should be to the login route, once we have the new link lets configure it app.js routes.

import { useAuth } from './Auth'

  const auth = useAuth()

  return (

  <NavLink style={myNavLinkStyle}  to='/profile'>Profile</NavLink>

        {

          !auth.user && (

            <NavLink style={myNavLinkStyle}  to='/login'>Login</NavLink>

        )}

    </nav>

* So new route in app.js path is “login” & element is “<Login>” component.

            <Route path='login' element={<Login />} />

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

* Now Step3 is completed for step 4, let’s display logged in email address in profile component & also add a logout button & for display of email we again rely on “authContext”. Import auth from auth file, See below code

import {useAuth} from './Auth'

const Profile = () => {

  const auth = useAuth()

  return (

    <div>

        <h2>Welcome {auth.user}</h2>

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

    </div>

* And in JSX call {auth.user}, and add a button for logout call a function on click named it userLogout & define this function assign to arrow function & inside call “auth.logout()”, after logging out we will redirect to homepage.
* So add {useNavigate} hook from react router dom then call this hook assign to “homeNavigate” & call it after logout & pass “/” root path.

import { useNavigate } from 'react-router-dom'

const Profile = () => {

  const auth = useAuth()

  const homeNavigate = useNavigate()

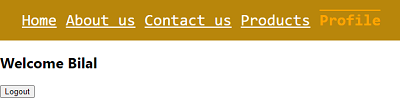
  const userLogout = () => {

      auth.logout()

      homeNavigate("./")

  }

* So that we have completed login & logout feature for our application so save & check in browser, just click on “login” button.



* And we are at login route put name value “Bilal” & click login button & we are navigated to homepage then click on profile page.
* We can see “welcome Bilal”, now click on logout we are back again at home page, but profile is still not protected we access directly.
* Step 5 protect profile route, now protecting a route should be a reusable block of code, for now we just need to protect profile page but we might have more protected pages in the future.
* So we are going to create a reusable wrapper component that decides if the component can be rendered or if the user has to login first.
* Create new component named it “IncludeAuth” inside define functional component & get hold of auth context so call “const auth = useAuth()”.
* Here we check that if the user is not logged in, so code is if not auth.user & if this is true case we redirect the user to the login route.
* And we do that using the navigate component from react router dom.
* And then within the “if block code” we are going to return navigate component with a “to” prop is equal to “/login”.
* So back to login route, if the user is logged in however we are going to return the children prop, DE structured it return children.

import {useAuth} from './Auth'

import { Navigate } from 'react-router-dom'

function IncludeAuth( {children} ) {

  const auth = useAuth()

  const loginNavigate = useNavigate()

  if (!auth.user){

      return <Navigate to='/login' />

  }

  Return children

}

* Now we just need to do in app.js, so wrapped the profile component with “IncludeAuth” component, so in the element prop “<IncludeAuth>” & is going to wrap the profile component see below code.

<Route path='profile' element=

                                        {<IncludeAuth>

                                          <Profile />

                                        </IncludeAuth>}

                                        />

* Make sure that we import “IncludeAuth” at the top, now check in browser.
* We are currently not logged in “see home page”
* C:\Users\CZ 3\Desktop\Untitled.pngClick on profile we can see we are redirected to the login route, because at the moment are not login it not giving us access to the profile page so put name and click login button & we are in the home page click on profile.
* We can now see profile page “Welcome Bilal” & logout button, just logging out and we are directed to the home page.
* Our protected is working well but we need to improve the application.
* If we login & click the back button are at the login page again to fix this in login file on the navigation function add a 2nd argument (‘/’, { replace: true}) if we not refresh the page & again login by put username.
* And click the back button we don’t see the login route anymore.
* Another thing we can do hat redirecting the user back to the protected route after the login, right now if we click profile & put username for login.
* We are in the home page; it would be nice if we were back at the profile page. For that we are going to work with state when navigating.
* In IncludeAuth component at top import use Location from react router.
* And within component call the hook “const Location = useLocation”.
* Next on the navigate component pass in the state prop “state = {{ path: location.pathname}} “ & pass in an object, we can make use of this state in the login component, see code below.

import { Navigate, useLocation } from 'react-router-dom'

  const location = useLocation()

  if (!auth.user){

      return <Navigate to='/login' state={ { path: location.pathname } } />

* In login component at top import “{ useLocation } again & invoke the hook “ location = useLocation “, next check if the path is set on the state object & use the same as redirect path, so “const redirectPath = location.state ? .path || ‘/’ of not navigate to the root, pass in redirect path to navigate.

Login.js

import {useNavigate, useLocation} from 'react-router-dom'

const location = useLocation()

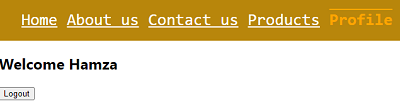
const redirectPath = location.state?.path || '/'

const userLogin = () => {

auth.login(user)

navigate(redirectPath, { replace: true})

* See code above & test it out in the browser again, refresh we have profile go to login enter “username” hit login button, we are back in profile, now click logout, we are directed to the home page.
* Authentication & protected routes with react router has been successfully implemented, but see the code again & let summarize the steps.



* So user already logged in it redirect us to the profile page and hide the login link in navigation & click logout then login link appear again.
* First we create profile route which should be protected, next we implemented the functionality to log in & log out a user, for that we relied on react context & provided the context value to the entire component tree, after that we implemented the login page.
* We use the auth context to sign in & the navigate function to redirect, in profile page we used the same auth context to display the logged in user & handle the logout button, & finally we created the include auth component that checks if the user is logged in or not.
* If the user is not logged in it redirects to the login route, if the user is logged in it renders the children prop, we wrap any component that needs to be protected with the “IncludeAuth” component.
* This is done in app.js when we configure the particular route & last in login component we used replace set to “true” to prevent the user from viewing the login page after logging in & bit of useLocation technique to keep track of redirection, so this is how we handle authentication with react router.