**1 - Introduction**

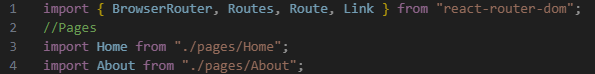
* <https://www.youtube.com/watch?v=OMQ2QARHPo0&list=PL4cUxeGkcC9iVKmtNuCeIswnQ97in2GGf>
* When you make app using react, you are making a single page Application (SPA) ; which means when you make a request for website on the browser a server just sends a single HTML page and react JavaScript code needed to render a content and react components into that page.
* Unlike traditional website which server responded with new html page for every request from client/browser, react handles routing in the browser in the frontend and do not send any additional request to server for new pages.
* React core library does NOT come with routing mechanism, so we need to use react router. So to use this feature we install additional package called “react-router-dom” which allows us to easily handle routing in the browser.
* This works, when you click a link to different page in the browser, react router intercept that request and stops it from reaching the server and instead the react router swaps the page content depending on whatever page we requested by injecting new content or component to the root. Hence Every page is essentially a react component, this process makes the website faster.
* Course files on github: <https://github.com/iamshaunjp/react-router-in-depth>
* Setup the project:
  + Create new react application (#npm create vite@latest -- --template react)
  + clean unnecessary contents( default CSS, logo and other)
  + Place your styles inside “inded.css”
  + **2 - React Router Basics**
* Installing react router
  + Run the following command in project directory to install react router

 // you can leave out the 6.4 version

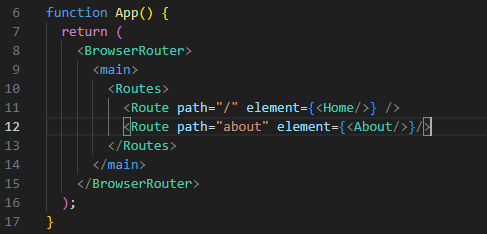
* + Create a pages/Components inside pages folder under “src” (Home.jsx, About.jsx …)
  + Import BrowserRouter, Routes, Route ,Link, NavLink from the “react-router-dom” inside root component (App.jsx) via de-structuring syntax



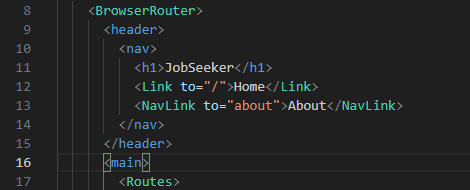
* + Import your pages to the “App.jsx”



* + Then Set up your routes, by wrapping everything in app.jsx with <BrowserRouter> and define each route by providing the path and the element they point to with <Route> component and Group all the routes with in the <Routes> component.

note, for root you can use index instead of path=”/”

* + This means when we go to just “/” which is the root path of the site, it is going to take the <Home/> component and render it inside <main> tag. Similar “/about” is going to swap the <Home/> with the <About/>component and the content of about page is shown in <main> tag instead.
  + Create <header> tag at the top(outside <main> tag which holds the Navigation menu links

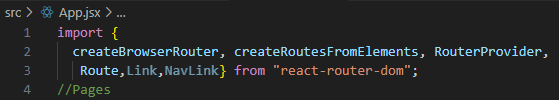


* + Note we used Link and NavLink in the <nav> tag to demonstrate the difference, NavLink will automatically adds ‘active’ class to the link tag, so it can allow as style any active links based on that class, however “Link” behaves like normal links with no special class added to them. So use NavLink if you want to style active links,

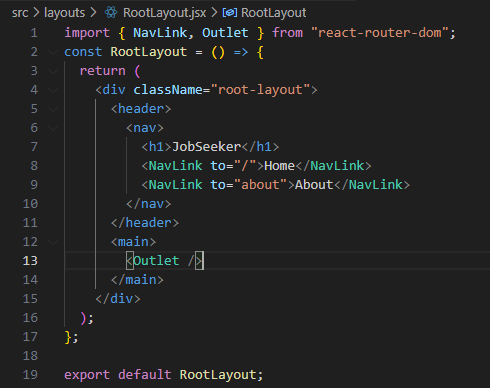
Note this way of using router is still valid, but after version 6.4 some features are added and how things implemented are a little bit different, so if you are using react router version 6.4 or latter start from the next lesson 3

**3 - Router Provider, createBrowserRouter & Outlet**

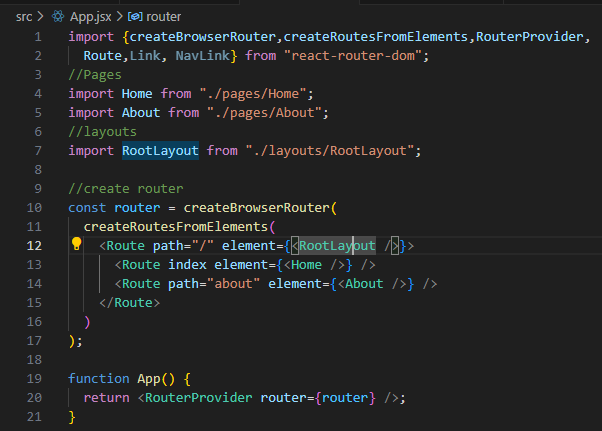
* Latest way of creating routes after the release of version 6.4
* Lets Convert the above route we created in lesson 2 to the latest version
  + Import the “CreateBrowserRouter” , “createRoutesFromElements” , ”RouterProvider” function at the top instead and remove the “BrowserRouter” and “Routes” from the de-structured list of imported functions



* + Use the “CreateBrowserRouter” function to create your router and store it on variable Globally outside the component definition. It takes the “createRoutesFromElementsFunction()” which intern takes tree of routes, in which the parent <Route> takes the path and element template(layout component for parent route) as a properties.
    - So to create the element template or layout component for the parent <Route> which wraps the other routes to pages/components inside it:
      * Create “layouts” folder inside “src” and inside it create “RootLayout.jsx” component
      * Import {NavLink} and {Outlet} component for layout, create the routes for each bage using “NaLink” and tell react where we want to output the content of this routes/pages when we click on each path using the “Outlet”. In this case to output the content in<main> tag. This means the routs in <header> tag always stay at the top of the page and when the links of child route clicked the corresponding content of the click component will be injected to the <main> tag.



* + Import it inside “App.jsx” and register(pass) it to the element property
  + Remove the <BrowserRouter> inside the app.jsx wrapping other routes inside it.
  + Take the router you created here and pass into route provider component



**4 - Nested Routes & Layouts**

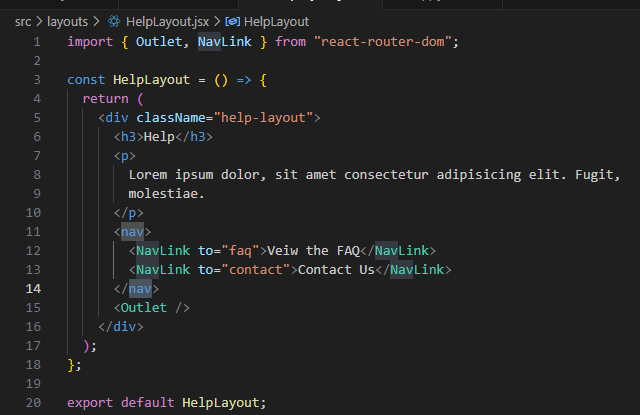
* Creating nested Routes: for example lets create help page having “faq” and “contact” as a children
  + Create layout component for help page named “HelpLayout.jsx” under layouts folder
  + Import NavLink, Outlet to it



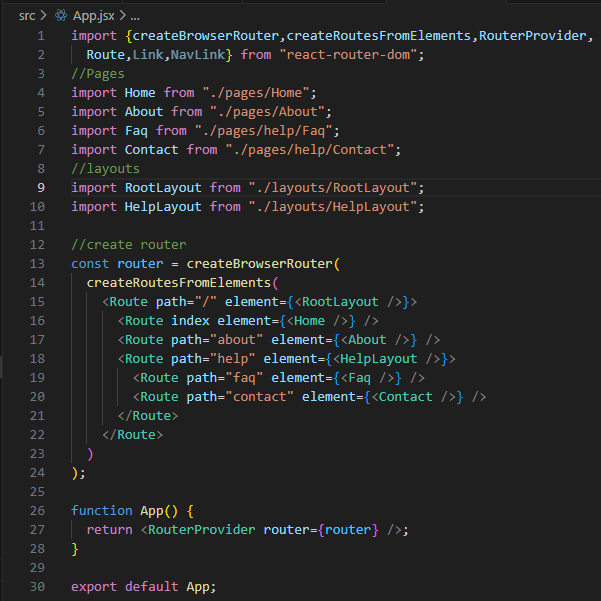
* + Create two components for “faq” and “Contact” inside” src/pages/help” folder, with their respective content and style them. Import them to the “app.jsx” register(pass) them to the routes on the element property:



* + Set the children routs and define where the content of the children rendered using <Outlet/>



* + Register this HelpLayout component in parent route on element property by importing into the root component “App.jsx” //line #24
  + Also Import and register the child components in the element property of child routs (line#25 and 26)

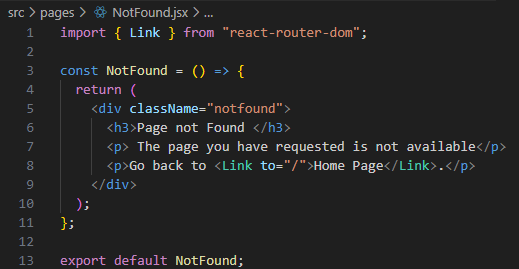


Note, When you nest a routes one inside the other like the above, hence the root path “/” is defined in the parent you do not need to add the forward-slash(/) before the child route, it will be automatically added for you. For example , Line #26 is equivalent to: /help/contact

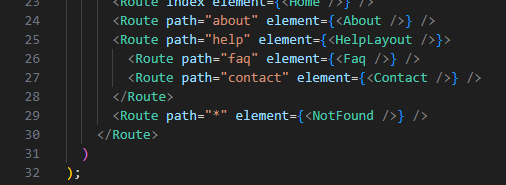
* Add “help” in on top menu by adding it in the RootLayout

**5 - Custom 404 Page**

* What if we go to the route/page that doesn’t exist. We can create our own error page(catch thrown error) and show a better error message to user in such cases.
* To handle such errors we can either use Error Elements or by creating catch all link inside root component(app.jsx) by registering all of our routes.
* Creating catch all rout to catch non existing page :
  + Create “NotFound.jsx” component inside pages folder that will show the 404 error

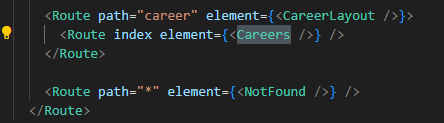


* + Below route tree in “app.jsx” create a route pointing to a Component that will display message if all the above routes in the parent <Route> doesn’t match the requested page. Then register the created page in the route tree. //Line #29



**6 - Loaders**

* A way we can load data into a component before it renders. For example, if we have products page which lists our product that come from API and loaders allow us to fetch those products from the API before the component renders in the browser. Inside that component we don’t have to worry about using useEffect hook to go and fetch a data when the component renders, because the loader function does all that for us ahead of time.
* Available on version 6.4 or later
* Using Loaders:
  + Usually when we fetch data for a Component from API we use “useEffect” hook and fetch data inside that hook. sometimes we probably need to update state to keep that data so that we can use it in our template
  + However, when we use Loader function, we don’t need to use the useEffect hook to fetch or store the data in local states.
  + To demo this Create nested routes Similar to the above(lesson 4) we have created inside “App.jsx” which routes to careers.



* + To simulate data fetching from data source( API endpoint ) we create JSON file named “db.json” in “data” folder that contains array of objects and we fetch it with the help of a package called “Json-Server” which allows us to wrap Json files with API endpoints so that we can interact with it like REST API.
    - So Install Json-Server package globally on your computer

// the -g flag is for global

* + - Then serve the Json using the Json-Server on different port(unoccupied, for e.g. 4000) (open new terminal and run the following command in your project directory to serve and watch the Json file

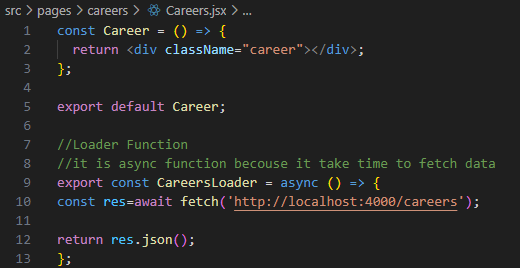


//the -p flag is for port number and the -w flag is watch the specified file

* + - When the above code runs and the Json-servers starts to serve our file, it will list for us the API endpoint at which we can access our resource



* + Now we Load our data inside the Careers Component from data source with loaders. This is done by creating function outside the component in the same component file(career.jsx).

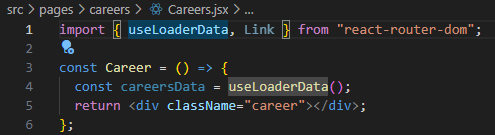


* + Now associate this function with index route inside the careers route in route tree of by importing into “App.jsx” file and passing to the “loader” property.

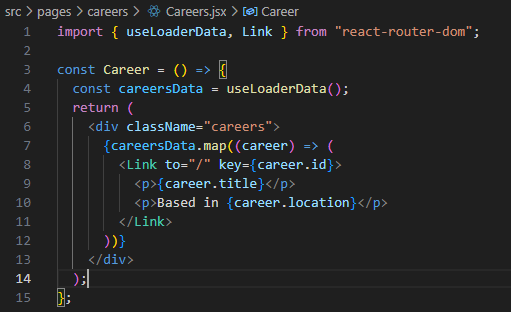




i.e. whenever we visite this route(line#35) it is going to run the “CareersLoader” function ahead of time which fetches data and returns Json as a promise, then that promise is resolved and we access it inside “Careers” component by using special hook react router give us known as “useLoaderData” as follows

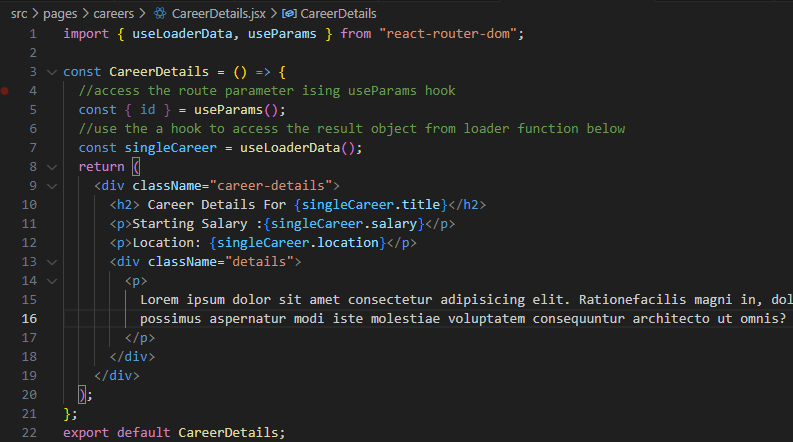


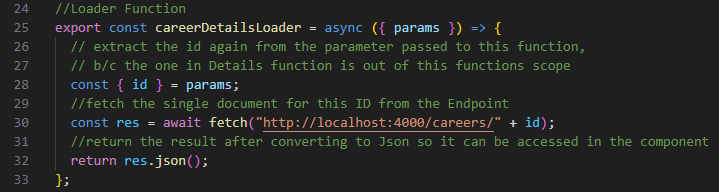
* + Output the data you received data:



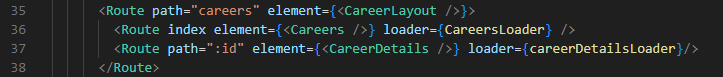
**7 - Route Parameters**

* Route parameters are changeable parts of a route when the same page component is shown for each variation. For example, we might have products detail page that has route path of ‘/products/id’ where the id part is changeable(could be 1,2,3 or abc567). This ‘id’ is known as a route parameter, because it is the part of the path that changes. And for each of those different changing ID values, we still want to show the same product details page components and same template, we also want to fill that component template with the product associated with whatever the id is in the URL.
* Using react router we can access that route parameter both in components using special react router hook and also in loader functions in parameter react router automatically gives to us.
* Usage Demo:
  + Create “CareerDetails.jsx” component in “src/pages/careers”, inside this component access a rout parameter “ID” using special hook known as “*useParams*” by de-structure syntax. This id is the Id received from the route. In addition, create a loader function “*careerDetailsLoader*” inside this component which also access this “id” and fetches the details of single career. Then access that data and store it on variable in the component (CareerDetails) via special hook called “useLoaderData”. Then output this data to the DOM via template.

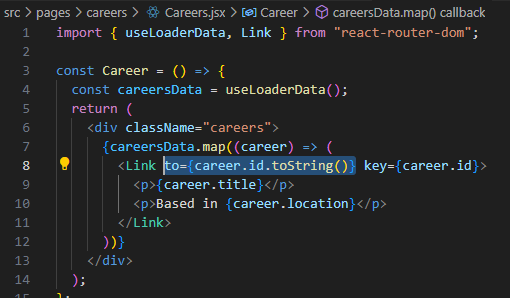


* +   
    Inside root component(App.jsx”) create route for details page as a child to careers route, with path referencing the id(“/:id” ). The colon signifies the changeable part of route which can be captured when this route is requested and returns a component associated with it.
  + import both the component and loader function to the “App.jsx” and register the component to element and loader function to ‘loader’ property of the route.



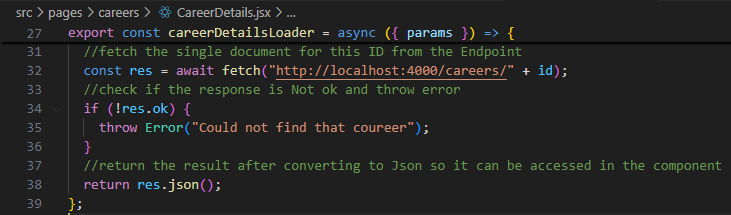
This means the “careerDetailsLoader” function runs to obtain data before the “careerDetails” component is rendered.

* + Now inside careers page modify the Link to index “/” to the route parameter

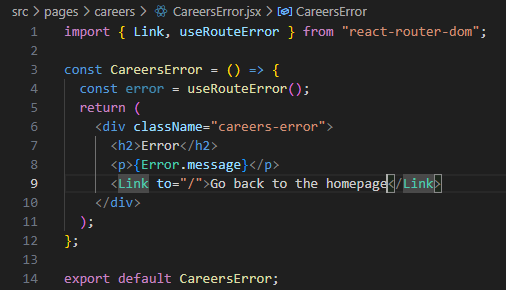


**8 - Error Elements**

* Recall when we created catch all route when the page is not found to catch the error, here is more better way to show error message when the resource/route is unavailable
* Steps:
  + Do error handling in the loader function when we try to fetch a career, if the career doesn’t exist we through custom error inside this function.
    - Go to “CareerDetails” component and check if the response is “OK”. The response object in the loader function has an “ok” property (it is true if the response have some data and false if it don’t have any data)

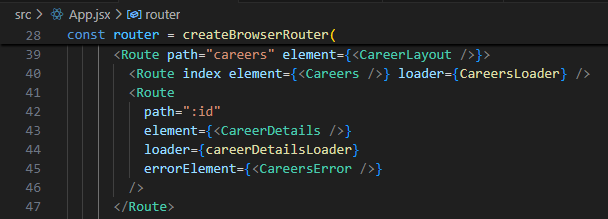


* + Make an error component that we can show when we have the above error.
    - Create “CareerError.jsx” component inside the career folder, inside it access the error thrown from loader function via a react router hook called “useRouteError”

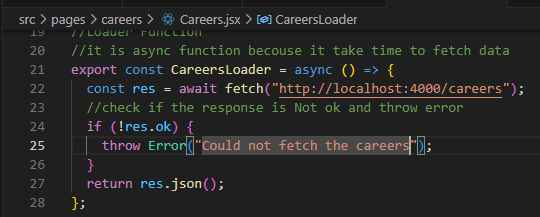


* + Import and Register/Associate the error component to career details route via “errorElement” property, so when we throw that error in route component in loader function the react router will show the error component associated with that route. And that error component will have access to the error we through.

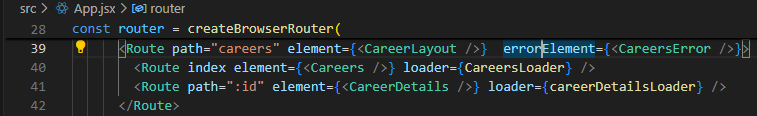




* + You might also check if the response is “ok” inside the careers component too when fetching the entire data, in case if the endpoint is wrong or API is down or something else you need to handle that error.

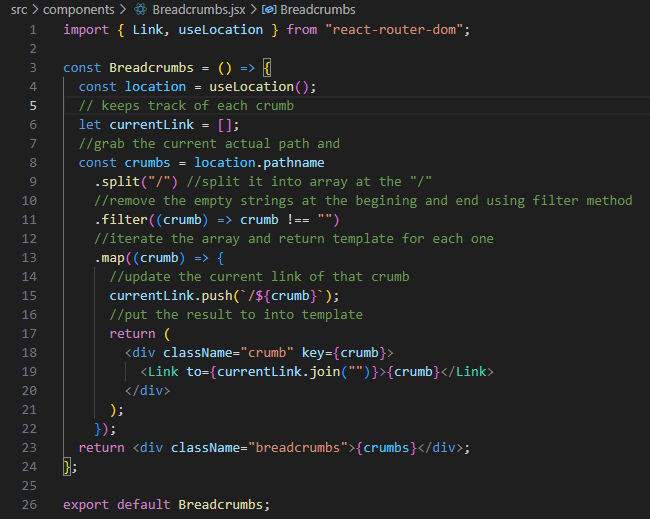


* + Create component to display this error and pass it to errorElement or reuse the above error component to the parent route so that it can serve both of the children.



**9 - Making Breadcrumbs (useLocation hook)**

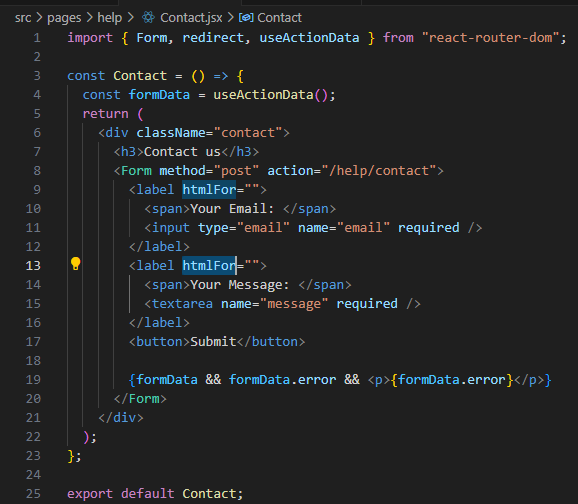
* “useLocation” Is react route hook which gives us a lot of data about current location of user on the site and we can use that hook to make breadcrumbs component which sits in the root component just underneath the navbar for every page.
* Breadcrumbs in sites are just a list of locations users visited to reach that current destination. i.e. They are like little trails of breadcrumbs to the current page. So, we are going to look at URL and break it down into individual pages to find current location of the user.
* Implementation:
  + Make “Breadcrumbs.jsx” component inside components folder (a folder for non-page specific reusable components that can be used in different places in the application).
  + Use “useLocation” hook inside the component, extract the “pathname” property out of it and split each part of the route then output them in the template as a links to those pages.



* + Go to the “RootLayout” import and place it under the <nav> tag in the <header> tag

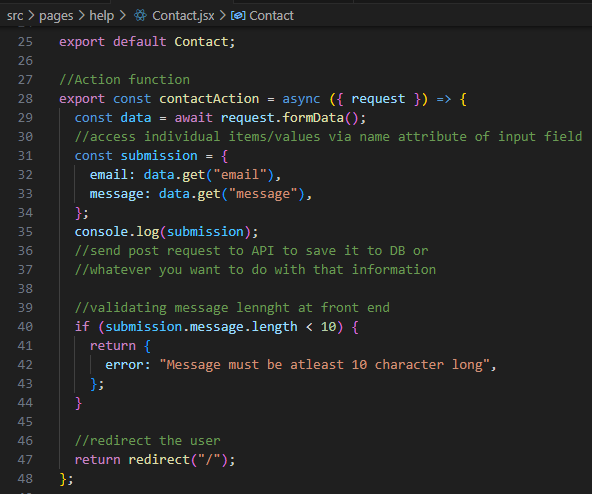
**10 - Forms & Actions**

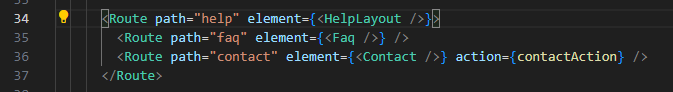
* Works on react router version 6.4 or later like loaders
* Reduce the need to track individual form fields using local state and event handler functions.
* Previously you need to attach each form field to an event handler function to update local state for that field and then bundle all those field values together, when the form is submitted inside some other submit event handler. After version 6.4 you don’t have to do that. All you have to do is use the react router form component which will automatically track all those values for us and when we submit the form they will bundle together all of those values into a request object and pass it into an action function. The action function is something we make ourselves a bit like loader function. And you get access to form data so that you can do something with it.
* Implementation steps.
  + Create a <Form> component using react router Dom: pages/help/Contact.jsx



//we use the name property to access the values of the fields

* + The action property is a function that fires when the form is submitted, and we don’t specify it like function name, we just specify the route path instead.
  + Then we create function which takes request Object as argument (contains all form data) and attach that function as an action property to a specific route in App.jsx. i.e. find the action that is associated with the route “/help/Contact”



* + We can access the returned values from action function inside the component using “useActionData” hook
  + Import and Register this action function to the route in App.jsx
  + 

**11 - Navigate Component**

* Using navigate component to redirect user based on certain conditions. For example, if you want to redirect someone who is NOT logged in, you could do quick authentication check using some kind of global state or other means , and if that is null(user not logged in) then you could return navigate component to redirect user away from any given page.
* This is useful for protecting frontend pages.
* Note, if you want to protect any data from unauthenticated users, that needs to be primarily locked out in the server side.
* Implementation:
  + Let’s use the about page for this demo, since we don’t have an authentication system in place lets just simulate that by creating local state inside our component
  + 