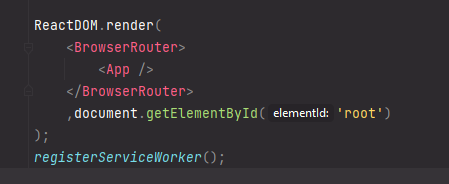
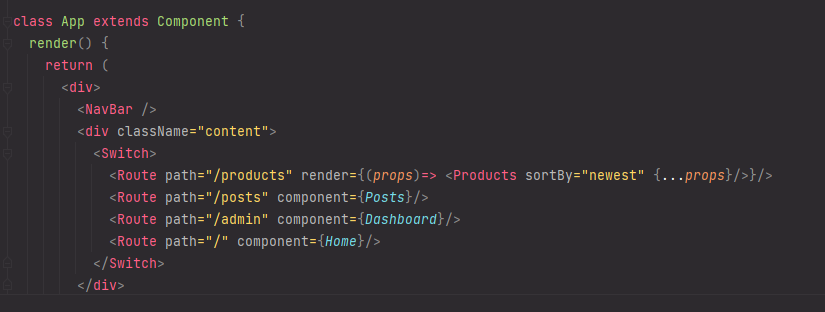
React by itself don’t have routing. We have to install a package called **react-router-dom.**

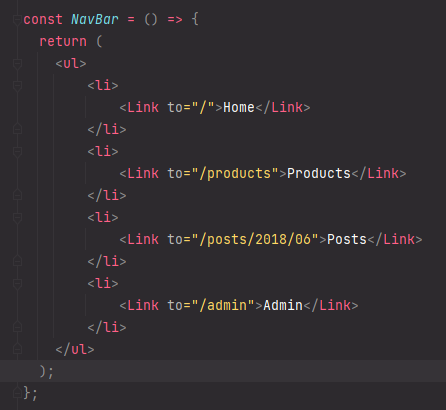
* We have to wrap the app component in the index.js with **BrowserRouter** component. This component wraps the history object in browsers and passes it downs to component tree.



* Next, we need to register routes (means telling react what component to mount based on a given URL). We add the **Route** component in the appropriate div and configure routes there. This component takes two props, path (the URL) and component (component to mount).
* Here We have to keep in mind about the matching logic, to fix the exact routing path that is written, we can write **exact** in the props, or wrap the components in **Switch** component. They both will fix the issue. The more specific route will be first.



* Another issue is, once we send request, the browser redownload the entire bundle.js file and http file every time. To prevent this, instead of using <a> tag, we can use **Link** component. This will make the application a single page app. Note – here instead of href attribute, we have to write to attribute.

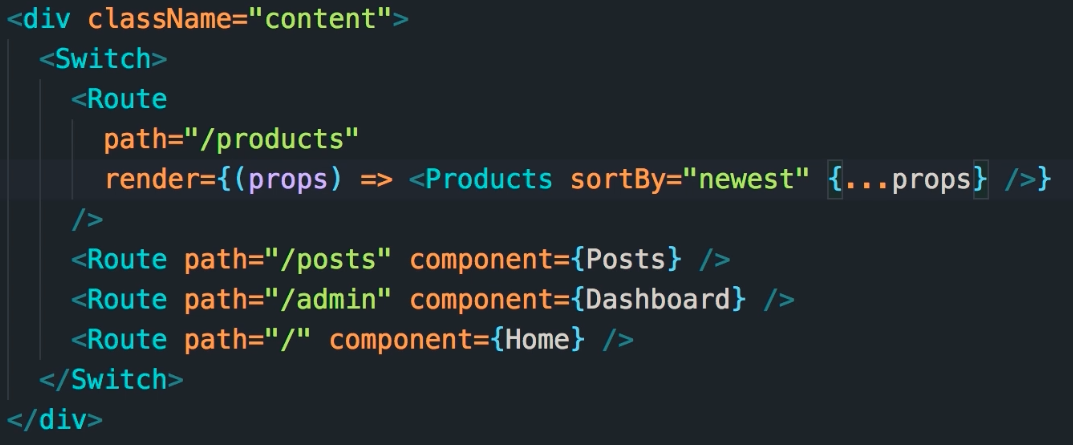


**Route Props:** By default, each route has three props – **location (**represents where the app is now**), history (**history object from browser**)** and **match (**contains information about how the URL was matched**).**

**Passing props to Route component:** In this case, instead of passing **component** as a prop, we can pass **render**. This takes an arrow function, and returns the component with additional props.



But this has a small issue, the default props (location, history and match) are lost if we do this. To fix that, we can pass props as a parameter of the arrow function, and de-structure the props in the component props.

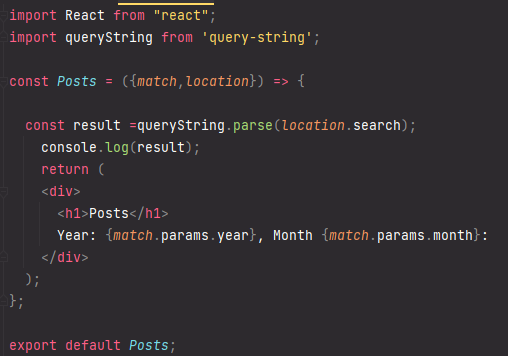


**Route Parameter:** To define a parameter in the route, we have to add a colon before it. We can define multiple parameters in same way.We can read that parameter using the match object (match.params.id for example). To make a parameter optional, we can append “?” with the parameter.



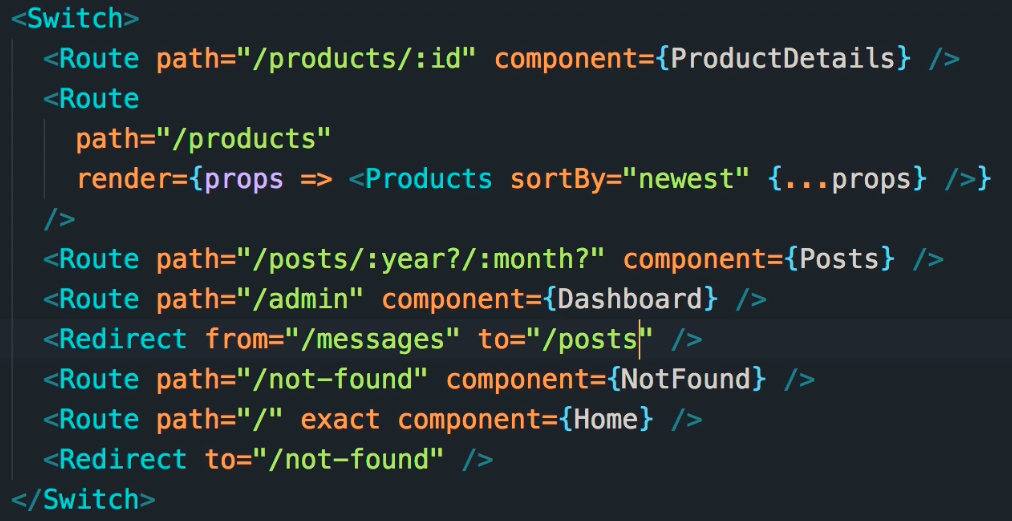
**Query String Parameter:** We should avoid using optional parameters in the route. Instead, we should use them using query string. In the **location** prop, the **search** property has the entire query string.

We don’t want to parse them ourselves, for that we will use library called **query-string**. This package has a method called parse which we can use to get the query string object.



**Redirection:** To redirect the user to a different URL, we need to import the **Redirect** from the **react-router-dom** package. After this, it’s best to specify the homepage to be exact in the props. Otherwise, the redirection will take it to homepage.

Next, we need to specify the redirect component path at the very bottom (since only time it will go there is when it don’t find any matching route). We can specify where we want to redirect the users to there. In this case we have to specify which component to mount on that URL using a Route for not-found.



In case we want to redirect users to another URL, in this case in the redirect we can specify both **from** and **to.**

**Programmatic Navigation:** For example, after submitting a form we want to redirect the user to product page. To achieve this, we need to use the 3rd prop in the Route, **history.** It has some useful methods like **push()** and **replace().** Difference between them is, push will add the new address in the browser history, and replace will replace the current address. So, in case of replace we can’t go back to previous address. This is useful in cases like removing login page or form submission from the history.

**Nested Routing:** in case we want routing in a separate area, for example admin page, we can use nested routing.

