On this article, I will explain how to implement a React Router using React 16 + and react-router-dom 6 +. I mention these two versions because there are both significative in the sense of

* React 16 allows for functional components, which are used through the exercise
* Similar for Hooks which are also used extensively
* This major React upgrade triggered a bunch of other associated libraries to also evolve in such a manner as to also accept Functional Components and Hooks, which are a major functional upgrade of the library.

Before we upgraded to React 16 , the react-router-dom was the standard for routing within the Rect community and it worked very well. The arrival of Functional Components and Hooks created a bit of an upheaval on the React community and libraries were unsettled and unsteady. Version 6 + is the stable version of the react-router-dom that works in perfect synch with React 16, taking full advantage of what React 16 has to offer.

The use case for Routing is quite obvious from the time that your app has more than page. How do you navigate through the different pages of an app? Normally you do that by launching commands through events on components, like clicks of buttons, or menu selections or any given event. Web and Apps are displayed on either Web Browsers or Apps, but one characteristic is common: each page will have a unique associated url. So normally, a url = a page. We could think of a url as the unique identifier for the page. This is obvious when navigating on the internet.

What we’ll learn today is one method for letting your application do this magic of navigation. I will assume that you know nothing, but you are curious and have some computer literacy, that you have the basic tools , IDE, and above all that you are keen to learn.

We’ll start by creating a React app. From the command line , enter the following command:

npx create-react-app router-v6

This will create a React project and it will give the router-v6 name to both the project and the folder

Once it’s finished , do the following

cd router-v6

npm start

This will start the just created application. This application is a basic template for a React application. You will need to change some of the code created for you, I suggest to use Microsoft Visual Studio Code which you can download and install on your PC.

Under the src folder of the project, add two folders

pages

routers

On the pages folder you’ll create a few pages that we can use to navigate from and to. We’ll call them Home.js, About.js and NotFound.js.

The Home component will be like this:

const Home = () => {

    return (

        <>

        <h1>Home</h1>

        </>

    )

}

export default Home

The About component will be like this: (notice how similar they are!)

const About = () => {

    return (

        <>

            <h1>About</h1>

        </>

    )

}

export default About

Now we can change the App.js to remove the dummy code generated by the create-react-app command and we’ll start referring to OUR application. We should see something like this

import Home from './pages/Home';

function App() {

  return (

    <>

<div className='App-header'>

      <Home />

  </div>

    </>

  );

}

export default App;

Note that

* We have imported the Home component
* We’ll be displaying the <Home /> component
* Note that we have a className to the css already generated by the create-react-app. This will render the content with some basic style

If we run the application now (using npm start) we’ll see the Home page. So far so good.

Graphical user interface, chart

Description automatically generated with medium confidence

If we change App.js to display the About.js component, as shown below

Text

Description automatically generated

Here what we are doing is changing the code so that, instead of displaying the Home component, we are displaying the About component.

Then we would see the About page.

Graphical user interface, application

Description automatically generated

So we have a working React app with two pages and we have tested that if you recall them, they will display as expected. Of course, we can’t just change and recompile the code every time a page needs to go from one location to another.

This is where routing comes into play.

First we’ll declare the routes linking a given url to the page we want to display when the url is being used: In order to do that we need to do the following: Create a new file, under ‘routes’ folder called Routing.js

As before create the skeleton of the script

const Routings = () => {

    return(

        < >

  </>

    )

}

export default Routings

I’m assuming that you understand this code, but what we are doing with this code is to declare a function, which in this example does not get any parameters and it returns some html. The whole function can be exported to that it can be ‘imported’ by other functions.

Now we need to import the following two functions from the react-router-dom we installed earlier:

Import { Routes, Route } from ‘react-router-dom’

Then , inside the return statement we need to add the <Routes></Routes>

And, within the <Routes> component, add as many <Route> as needed. This is how we would declare for the two pages we created earlier, Home and About. Routings.js will look like this after the above changes:

import { Routes, Route } from 'react-router-dom'

import About from '../pages/About'

import Home from '../pages/Home'

const Routings = () => {

    return(

        <div>

        <Routes>

            <Route path='/' element= {<Home />}></Route>

            <Route path='/about' element={<About />}></Route>

        </Routes>

        </div>

    )

}

export default Routings

So to note the following :

* We imported Routes and Route from the router library.
* We return a <Routes></Routes>
* And Inside the <Routes> we add each <Route>
* Each <Route> has two parameters:
  + The path (or the url)
  + The element we want to display
  + Note that the Elements we want to display are also imported.

On previous versions of react-router-dom, the <Routes> component was a <Switch> component which is deprecated on this version, but it is useful for us to help understand the underlying process. <Routes> works as a swicth statement , it will only render the Element for which the url matches the best.

Now we’ll change the App.js application so that, whenever the url of the application matches the path on the Route, it displays the corresponding Element on the Route.

import './App.css';

import Routings from './router/routings';

function App() {

  return (

    <>    <div className='App-header'>

      <Routings/>

    </div>

    </>

  );

}

export default App;

So instead of saying display Home or About, we are saying display whichever element matches the current url. When we start the app, it will start in <http://localhost:3000/> so it already matching the Home page. So it will display the Home page.

if we test the application now we can see that on url http://localhost:3000 it will display the Home page and if we change the url to <http://localhost:3000/about> it will display the About page. These match with the routing. For the sake of completeness, we add a page to display in cases where there is no url match. This is common practice.

Please create a new page under the folder ‘pages’ and call it NotFound.js. It should be as follows:

const NotFound = () => {

    return (

        <h1>This page does not exist</h1>

    )

}

export default NotFound

Now we’ll add it to the Routings.js as follows:

import { Routes, Route } from 'react-router-dom'

import About from '../pages/About'cle'

import Home from '../pages/Home'

import NotFound from '../pages/NotFound'

const Routings = () => {

    return(

        <div>

        <Routes>

            <Route exact path='/' element= {<Home />}></Route>

            <Route exact path='/about' element= {<About />}></Route>

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

        </Routes>

        </div>

    )

}

export default Routings

Note that we have imported it , added the Route and that the path has ‘\*’. This is to indicate ‘none of the above’. This ‘carch-all’ is similar to the switch default in Java. It’s like ‘anyting else’. This should be the last <Route/>.

Refresh and test entering anything on the url (after the <http://localhost:3000/> ) that is neither the blank or ‘/about’ and you should see the new NotFound page, as follows:

Text

Description automatically generated

So far we’ve got the routes and the pages but we still have to do lots of manual work to move around. This is what we’ll learn next: how to make the app do it for us.

We have stablished that

* If url = <http://localhost:3000/> (or, what is the same , <http://localhost:3000>) we want to show the Home page
* If url = <http://localhost:3000/about>, then we want to display the About page.
* Finally for any other url that is not of the two above, we want to catch it up and display the NotFound page.

In order to implement it we’ll need some interface to allow the user to direct the pages: for example a button to direct the Home page to the About page, And also a button from the About page to the Home (or the previous pages) . Also from the NotFound page, we just want to direct the user to whichever was the previous page before landing in the NotFound page.

Let’s add some buttons and their onClick event handlers.

However, first let me also introduce you with the useNavigate function. This is a function from react-router-dom that lets you from within a page to direct the app to a different page. We’ll combine a button with the useNavigate function so that when a user clicks on a button, the page will direct the app towards a new page. In other words, it will change the app’s url. We have already seen that changing a url wil change the page.

Let’s start with the Home page. We need to first import the useNavigate and then use it. We also need to add a button and an handler so that when the user clicks on the button it will use the useNavigate to change the url.

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

const Home = () => {

    let navigate = useNavigate()

    const handleAbout = () => {

        navigate('/about')

    }

    return (

        <>

        <h1>Home</h1>

        <button onClick={handleAbout}>About </button>

        </>

    )

}

export default Home

Note the following :

* We have imported the useNavigate function from react-router-dom
* We have declared a variable that will inherit all its functionality (the navigate)
* We have added a button with the onClick event handler.
* The event handler (the click) uses the navigate function with a new path (url)

We have seen that the app does when we change the url, so we expect that when the user click the button the apps redirects to the About page.

You can try it.

It works, but now you see what happens, no? how do you get out of the About page, either back or to another page. We need to add a button the About page to go back to the Home Page.

Let’s do that:

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

    const About = () => {

let navigate = useNavigate()

        const handleBack = () => {

            navigate(-1)

        }

        return (

            <>

          <h1>About</h1>

            <button onClick={handleBack} >Back</button>

</>

        )

    }

    export default About

The logic with the button is exactly the same. What it’s different, however, is that the navigate now has a -1 instead of the url. Internally react-router-dom keeps a counter of the visited pages. So indicating -1 indicates that you want to go back to the previous one.

Now if we test it, we will see that it does work.

We will repeat the same for the NotFound page, so that we can either go back to the previous or go back to the Home. Let’s do it:

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

const NotFound = () => {

    let navigate = useNavigate()

    const handlePrevious = () => {

        navigate(-1)

    }

    const handleHome = () => {

        navigate('/')

    }

    return (

        <>

            <h1>This page does not exist</h1>

            <button onClick={handlePrevious}>Previous</button>

            <button onClick={handleHome}>Home</button>

        </>

    )

}

export default NotFound

I guess you are starting to get the feel. Now we have a fully routed application.

Needless to say the exercise button, could be anything, like selecting a menu, a link, any event. We just decided to implement it with buttons because of the simplicity.

I suggest that you add a button on the About page to do a navigate to a url that does not exist and then you could validate that the NotFound Previous button take you to the About page, while the Home button, takes you to the Home page.

Its on Git, but I thought it would be a little exercise for you to get practice.

I would like now to introduce a new use case to demonstrate the use of passing state from one page to another. On most real world applications this will be required.

I’ll show you how. Let’s start with modifying the notice ‘This page does not exist’ to Page <whichever page> does not exist. This message is more user-friendly but above all, it’s dynamic!

This will help us introduce another beast from react-router-dom, the Location. Location can be used to get any details of the current location, including the pathname. And other things too, as we will see.

I have now changed the NotFound element

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

const NotFound = () => {

    let navigate = useNavigate()

    let location = useLocation()

    const handlePrevious = () => {

        navigate(-1)

    }

    const handleHome = () => {

        navigate('/')

    }

    return (

        <>

            <h1>Page {location.pathname} does not exist</h1>

            <button onClick={handlePrevious}>Previous</button>

            <button onClick={handleHome}>Home</button>

        </>

    )

}

export default NotFound

I have added the following :

* Import useLocation
* Declared a variable (location) which inherits the useLocation functionality
* On the <H1> element, I have accessed the pathname from the location. This will always contain the actual location (the url) that the browser is accessing on the current page.

This very basic use of location has helped us to introduce it, but it can also be used for something far more useful: the state.

It may help to understand that the *useNavigate* and *useLocation* work in tandem. useNavigate can be used to change the App’s url and also to pass state (data) into the element that will be displayed for the new url. And *useLocation*, apart from the pathname (as shown above) can also recover the data passed by useNavigate. To demonstrate it we will pass some data across from the Home to the About page and we’ll see how the *useLocation* can be used to recover the data.

We’ll create an object in the Home page and we’ll add it to the navigate.

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

const Home = () => {

    let navigate = useNavigate()

    const someState = {

            one: 1,

            two: 2,

            twenty: 20,

            today: new Date().toLocaleString()

    }

    const handleAbout = () => {

        navigate('/about' , {state: {someState}} )

    }

    return (

        <>

        <h1>Home</h1>

        <button onClick={handleAbout}>About </button>

        </>

    )

}

export default Home

Note the following

* An Object (someState) has been defined with a few properties as key value. For this example it does not matter the contents. I have created an object but any state type would work.
* On the navigate call, note that I have added the state as parameter. And I’m passing to it the contents of the someState Object.

Note that the application will run OK, but unless the About element uses the state passed to it, it will ignore it and do as if it didn’t exist. For that we need the location.

On the About Element I have now added a couple of elements to both get the state and display it.

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

    const About = () => {

        let location = useLocation()

        let navigate = useNavigate()

        const {pathname , state } = location

        const handleBack = () => {

            navigate(-1)

        }

        const handleArticle =() => {

            navigate('/article', {state: 'From the About'})

        }

        return (

            <>

                <h1>About</h1>

                <code>{JSON.stringify(state)}</code>

                <br/>

                <code>{pathname}</code>

                <br/>

                <button onClick={handleBack} >Back</button>

                <button onClick={handleArticle} >Article </button>

            </>

        )

    }

    export default About

Note that

* I have imported the useLocation from ‘react-router-dom’
* Created a variable ‘location’ that uses the useLocation.
* I’m deconstructing the pathname and the state from the location.
* I’m displaying the state and the pathname on separate lines

This is how you pass state from one page to another.

I hope you found it useful. As usual, the code can be found in git

* I’m displaying them on the page

This will give us this