

Will React Hooks Replace React Router?



Piumi Liyana Gunawardhana



9 min read



Jun 14, 2024

Updated



TL;DR: *Level up your React app's navigation with either the powerful React Router or the user-friendly Hookrouter! This guide helps you choose the perfect fit for your project.*

Routing is a necessity when it comes to single-page applications. Developers use various routing libraries to implement routing for their apps. For React, [React Router](#) is the go-to library to handle routing, and there is a massive community around it.

However, with the emergence of [React Hooks](#) as an alternative solution, the [Hookrouter](#) was introduced as a flexible, fast router based on Hooks.

In this article, let's discuss whether there is a possibility of replacing React Router while leveraging the power of hooks with Hookrouter.

Defining Routes

Assume that you are developing a React app with three pages. The conventional way of implementing routing with React Router is as follows.



```
import Nav from './components/Nav';
import Home from './components/Home';
import Users from './components/Users';
import Profile from './components/Profile';
import {BrowserRouter as Router,Route} from'react-router-dom';

function App() {
  return (
    <Router>
      <div className="App"><Nav/>
        <Route path = '/' component = {Home}/>
        <Route path = '/users' component = {Users}/>
        <Route path = '/profile' component = {Profile}/></div>
    </Router>
  );
}

export default App;
```

Defining routes with Hookrouter is pretty straightforward. You just have to define routes as a plain JavaScript object. The **useRoutes()** Hook in the Hookrouter module analyzes and returns a result for a predefined routes object. We have to define routes as keys in the routes object, and their values as functions that are triggered when the routes match.



```
import {useRoutes} from 'hookrouter';
import Nav from './components/Nav';
import Home from './components/Home';
import Users from './components/Users';
import Profile from './components/Profile';

function App() {
  const routes = {
    '/' : ()=><Home/>,
    '/users' : ()=> <Users/>,
    '/profile' : ()=> <Profile/>,
  };
  const routeResults = useRoutes(routes);
  return (
    <div className="App">
      <Nav/>
      {routeResults}
    </div>
  );
}

export default App;
```

Routing with **useRoutes()** seems appealing since there isn't much to do for us. For each route in the app, we have to render the **<Route/>** component when using React Router. But with Hookrouter, we can simply use the defined routes in our app by sending them to the **useRoutes()** Hook.

Hookrouter produces the same outcome as the React Router but in a more streamlined and lightweight manner.



Syncfusion React UI components are the developers' choice to build user-friendly web applications. You deserve them too.

Explore Now

Navigation

React Router offers the `<Link/>` component to configure route navigations and handle interactive routing.

```
import React from 'react';
import {Link} from 'react-router-dom';

function Nav() {
  return (
    <div>
      <nav>
        <ul className='nav-links'>
          <Link className='Link' to='/'>
            <li>HOME</li> </Link>

```

Copy

```

        <Link className='Link' to='/users'>
          <li>USERS</li> </Link>
        <Link className='Link' to='/profile'>
          <li>PROFILE</li> </Link>
      </ul>
    </nav>
  </div>
);
}

export default Nav;

```

To facilitate navigation, the Hookrouter module wraps the HTML anchor tag `<a/>` as `<A/>`.

```

import React from 'react';
import {A} from 'hookrouter';

function Nav() {
  return (
    <div>
      <nav>
        <ul className='nav-links'>
          <A className='Link' href='/'>
            <li>HOME</li>
          </A>
          <A className='Link' href='/users'>
            <li>USERS</li>
          </A>

```

 Copy

```
        <A className='Link' href='/profile'>
          <li>PROFILE</li>
        </A>
      </ul>
    </nav>
  </div>
) ;
}

export default Nav;
```

Hookrouter is available as a React component and is completely feature-equivalent to the inherent `<a/>` tag. The only difference is that instead of loading a new page, Hookrouter moves navigations to the history stack.

In React apps, Hookrouter allows us to configure route navigations and handle interactive routing by rendering the routes on-screen and navigating to them when clicked.

Hookrouter provides programmatic navigation functionality with the **navigate()** Hook. We can apply the **navigate()** Hook to the direct users to a particular page defined by a given absolute or relative URL. Use the following example to navigate to a home page.

```
navigate('/home');
```



As every call to the **navigate()** function is forward navigation, users can use the back button on their browser to return to the previous URL. The **navigation()** Hook accepts three parameters: **navigate(url, [replace], [queryParams])**. We can use the second of these to replace the back button behavior. It deletes the existing history entry and creates a new one in its stead. To do this, simply set its argument to true.

```
navigate('/home', true);
```



Switch Functionality

When the predefined navigation routes are not matched, React Router renders a default page using the **<Switch/>** component.

It helps us to render a 404 page to inform the user that the requested route isn't available in the app. To achieve this, we encapsulate all rendered routes within the **<Switch/>** component and render the 404 page without specifying a path prop.



```
import Nav from './components/Nav';
import Home from './components/Home';
import Users from './components/Users';
import Profile from './components/Profile';
import Error from './components/Error';
import {BrowserRouter as Router, Switch,Route} from'react-router-dom';

function App() {
return (
  <Router>
    <div className="App">
      <Nav/>
      <Switch>
        <Route path = '/' exact component = {Home}/>
        <Route path = '/users' component = {Users}/>
        <Route path = '/profile'exact component = {Profile}/>
        <Route><Error/></Route>
      </Switch>
    </div>
  </Router>
);
}

export default App;
```



All Syncfusion's 70+ React UI components are well-documented. Refer to them to get started quickly.

Read Now

It's effortless to conditionally render routes with Hookrouter since we create a routes object that comprises all of our route paths. You have to pass that object into the **useRoutes()** Hook. When a defined route is not matched, just pass the 404 error file for rendering, along with the result function.



```
import {useRoutes} from 'hookrouter';
import Home from './components/Home';
import Users from './components/Users';
import Profile from './components/Profile';
import Error from './components/Error';

function App() {
  const routes = {
    '/' : ()=><Home/>,
    '/users' : ()=> <Users/>,
    '/profile' : ()=> <Profile/>,
  };

  const routeResults = useRoutes(routes);
  return (
```

```
<div className="App">
  <Nav/>
  {routeResults || <Error/>}
</div>
);
}

export default App;
```

One thing I've discovered with the React Router **<Switch>** is that if we didn't declare the exact path, it will result in inaccurate routings in some instances.

As an example, if we didn't mention the path to home as exact, the app will not route to any other path that begins with '/'. As a consequence, the app will never navigate to the Users or Profile pages, and instead keep falling back to the home page.

Hookrouter, on the other hand, does not require explicit definitions of exact paths since the routes are declared as an object.

Redirects

When users wish to dynamically redirect from one route to another, we need redirection. We can perform this with React Router using a few methods, such as

using the history object or the `<Redirect/>` component.

For example, if we have a login form, we can use the browser history object to redirect logged-in users to the `'/home'` route.

```
import React from 'react';

class Login extends React.Component {
  loginUser = () => {
    // if (user is logged in successfully)
    this.props.history.push('/home')
  }
  render() {
    return (
      <form>
        <input type="name" />
        <input type="email" />
        <button onClick={this.loginUser}>Login</button>
      </form>
    );
  }
}

export default Login;
```



Hookrouter handles redirects with the simple `useRedirect()` Hook, which takes the source route and target route as parameters.



```
import {useRoutes, useRedirect} from 'hookrouter';
import Home from "../components/Home";

const routes = {
  '/login': () => <Login />,
  '/home': () => <Home />
};

const Users = () => {
  useRedirect('/login', '/home');
  const routeResult = useRoutes(routes);
  return routeResult;
}
```

Since a replacement navigation intent is triggered by this Hook, the navigation history will only have one entry. As a result, if the `'/login'` route is redirected to `'/home,'` as indicated in the last code snippet, the `'/login'` route will be removed from the browser history.



Be amazed exploring what kind of application you can develop using Syncfusion React components.

Try Now

Final Thoughts

React Router is undoubtedly a fantastic tool. But, for some projects, React Router may be excessive if you only need simple navigation and routing functionality.

On the other hand, React Router has a lot of navigational components that you can use to structure your app declaratively, which may be quite beneficial for broader and more sophisticated navigational requirements in React apps.

However, with the launch of Hooks, a lot has happened in React. So, when handling routes in smaller apps, the Hooks-based approach provides a more flexible and clean solution.

Currently, I don't see a possibility that Hooks will replace React Router. But, with time, if it continues to evolve with rich functionality, it might have the potential to take over.

I hope you found this useful. Thank you for reading.

The [Synconfusion React](#) suite offers high-performance, lightweight, modular, and responsive UI components in a single package. It's the only suite you'll ever need to

construct a complete application.

If you have questions, you can contact us through our [support forum](#), [support portal](#), or [feedback portal](#). We are always happy to assist you!

Related blogs

- [Improve React Custom Hook Debugging with useDebugValue](#)
- [Understanding React's useEffect and useState Hooks](#)
- [Perform CRUD Operations in a React Data Grid Using Redux](#)
- [Top 5 React Lazy-Loading Libraries for 2022](#)



MEET THE AUTHOR

Piumi Liyana Gunawardhana

Software Engineer | Technical Writer since 2020