

# React Router

## What is React Router?

React Router is a library that provides navigational components for React developers to create Single-Page Applications (SPAs) with dynamic, client-side routing.

Applications that use React-Router can benefit from the separation of content afforded to multi-page applications without the break in the user-experience caused by page reloads.

## Importing BrowserRouter

In order to use React Router, the `BrowserRouter` component (often alias as `Router` ) must be imported into the top-level component file.

Wrapping the top-level component with `BrowserRouter` gives your application's entire component tree access to React Router.

```
import React from "react";
import { BrowserRouter as Router
} from "react-router-dom";
```

```
export default function
TopLevelComponent () {
  return (
    <Router>
      application contents here
    </Router>
  )
}
```

## Route

React Router's `<Route>` component is designed to render its children when its `path` prop matches the current URL.

The `<Route>` component has a boolean prop `exact` that, when `true`, will cause the `<Route>` to render its children only when the current URL exactly matches the `<Route>` component's `path`. When `exact` is `false` (its default value), a `<Route>` will render if its path partially matches the current URL.

```
import React from "react";
import { BrowserRouter as Router,
Route } from "react-router-dom";
import Users from
"../features/users/Users"
import NewUser from
"../features/users/NewUser";

export default const App () {
  return (
    <Router>
      <Route path="/users" exact>
        <Users />
      </Route>
      <Route path="/users/new">
        <NewUser />
      </Route>
    </Router>
  )
}
```

## Link

React Router's `<Link>` component can be used to create links for navigation. The `to` prop specifies the location to which the user will be redirected after clicking on the `<Link>`.

Rendering a `<Link>` will insert an anchor tag ( `<a>` ) in your HTML document, but the anchor's default behavior (triggering a page reload) will be disabled. This allows the application's `<Router>` to respond to URL changes by rendering the appropriate content.

```
<Link to="/about">About</Link>
```

## NavLink

React Router's `<NavLink>` is a special type of `<Link>` that can be styled differently when the component's `to` prop matches the current location.

The `activeClassName` prop (whose default value is `'active'`) specifies the class that will be applied when the `to` prop on the `<NavLink>` matches the current location.

```
<NavLink
  to="/about"
  activeClassName="highlighted"
>
  About
</NavLink>
```

## URL Parameters

URL parameters are dynamic (ie. non-constant) segments of a `<Route>` component's `path` prop. They can be used to dynamically serve resources based on the current window location.

A URL parameter begins with a colon and is followed by the name of the parameter, like so: `:parameter`. To specify that a URL parameter is optional, append a question mark, like so: `:parameter?`.

```
import { BrowserRouter as Router,
Route } from "react-router-dom"
import Book from
"../features/books/Book"

function App () {
  return (
    <Router>
      { /* bookId is required to
render <Book /> */}
      { /* page is not required to
render <Book /> */}
      <Route
path="/books/:bookId/:page?">
        <Book />
      </Route>
    </Router>
  )
}
```

## useParams()

React Router's `useParams()` hook can be used by a component rendered by a `<Route>` with a dynamic path to get the names and values of the current URL's parameters.

This function returns an object containing a key/value pair for each URL parameter where the key is the URL parameter's name and the value is the parameter's current value.

```
import React from "react";
import { useParams } from "react-
router-dom";

// assume this component is rendered
by a <Route> with the path
"/users/:userName"
export default const UserProfile () {
  const { userName } = useParams()
  return (
    <h1> Welcome {userName}! </h1>
  )
  /*
    If the user visits /users/Codey,
    the following will be rendered:

    <h1> Welcome Codey!
    */
}
```

React Router's `<Switch>` renders the first of its child `<Route>` or `<Redirect>` components whose `path` prop matches the current URL.

When wrapping multiple `<Route>` components in a `<Switch>`, it is important to order the `<Route>` components from most specific to least specific.

// Right: navigating to `"/songs/123"` will cause the first route to render, whereas navigating to `"/songs"` will cause the second to render

```
<Switch>
  <Route path="/songs/:songId">
    <Song />
  </Route>
  <Route path="/songs">
    <AllSongs />
  </Route>
</Switch>
```

// Wrong: navigating to `"/songs/123"` OR `"/songs"` will cause the first route to render. The second route will never render.

```
<Switch>
  <Route path="/songs">
    <AllSongs />
  </Route>
  <Route path="/songs/:songId">
    <Song />
  </Route>
</Switch>
```

## useRouteMatch()

`<Routes>` may be rendered in any component that descends from your `Router`. So, even components rendered by a `<Route>` can themselves render other `<Route>` components.

React Router's `useRouteMatch()` hook helps construct relative path and to props for `<Route>` and `<Link>` components by returning a `match` object with `url` and `path` properties:

The `path` property is used to build a nested `<Route>` component's `paths` prop relative to the parent `<Route>`.

The `url` property is used to build a nested `<Link>` component's `to` prop relative to the parent `<Route>`.

```
// App.js
import React from "react";
import { BrowserRouter as Router,
Route } from "react-router-dom";
import UserProfile from
"../features/users/UserProfile";

export default function App () {
  return (
    <Router>
      <Route path="/users/:userId">
        <UserProfile />
      </Route>
    </Router>
  )
}
```

```
// UserProfile.js
import React from "react";
import { Route, Link, useRouteMatch
} from "react-router-dom";
import FriendList from
"../FriendList";

export default function UserProfile
() {
  const { path, url } =
useRouteMatch();

  return (
    <div>
      <SomeUserProfileInformation/>

      { /* Redirects to
'/users/123/friends' */ }
      <Link to=
`${url}/friends`>Friends</Link>

      { /* Renders <FriendList/> for
the path '/users/:userId/friends' */ }
      <Route path=
`${path}/friends`>
        <FriendList/>
      </Route>
    </div>
  )
}
```

## Redirect

When rendered, React Router's `<Redirect>` component will change the current URL's path to the value of its `to` prop.

```
const Profile = ({isLoggedIn}) => {
  if (!isLoggedIn) {
    return <Redirect to="/sign-up"
  />
  } else {
    return <ProfileInfo />
  }
}
```

## useHistory()

React Router's `useHistory()` hook returns an instance of the `history` object, which has a mutable stack-like structure that keeps track of the user's session history and contains the following useful methods:

- `history.push(location)` - imperatively redirects the user to the specified location
- `go(n)` - Moves the pointer in the history stack by `n` entries
- `goBack()` - Equivalent to `go(-1)`
- `goForward()` - Equivalent to `go(1)`

```
import React from "react";
import { useHistory } from "react-router-dom";

export default function Footer () {
  const history = useHistory();

  return (
    <footer>
      <button onClick={() =>
        history.goBack()}>
        Back
      </button>
      <button onClick={() =>
        history.goForward()}>
        Forward
      </button>
      <button onClick={() =>
        history.push('/about')}>
        About
      </button>
    </footer>
  )
}
```

## Query Parameters

Query parameters appear in URLs beginning with a question mark ( `?` ) and are followed by a parameter name assigned to a value. They are optional and are most often used to search, sort and/or filter resources. For example, if you were to visit the provided URL you would be taken to Google's `/search` page displaying results for the search term `'codecademy'`. In this example, the name of the query parameter is `q`.

```
https://www.google.com/search?
q=codecademy
```

## useLocation()

React Router's `useLocation()` hook returns an object whose `search` property corresponds to the current URL's query string.

Passing the search string to the `URLSearchParams` constructor yields an object whose `.get(paramName)` method returns the value of `paramName`.

```
// If the user visits /search/?  
term=codecademy...  
const { search } = useLocation();  
// The value of search would be '?  
term=codecademy'  
const queryParams = new  
URLSearchParams(search);  
// queryParams is an object with  
a .get() method...  
const termValue  
= queryParams.get('term');  
// ... and termValue would be  
'codecademy'
```