# 20. Routing And Navigation

## Table of Content


1. Basics of React-Router-Dom
2. More about React-Router-Dom
3. Protected Routes


# 1. The react-router-dom library

Many modern websites are actually made up of a single page, they look like multiple pages because they contain components that render like separate pages. These are usually referred to as SPAs - single-page applications.

At its core, React Router conditionally renders certain components to display depending on the route being used in the URL (/ for the home page, /about for the about page, etc.).

To use React Router, you first have to install it using NPM:

```
npm install react-router-dom
```

▼ **BrowserRouter**

First, you'll need to set up your app to work with React Router. Everything that gets rendered will need to go inside the `<BrowserRouter>` element, so wrap your App in those first. It's the component that does all the logic of displaying various components that you provide it with.

```
// index.js
ReactDOM.render(
    <BrowserRouter>
        <App />
```

```
    </BrowserRouter>,
    document.getElementById('root')
)
```

### ▼ Route

Put simply, `Route` allows you to map your app's location to different React components. For example, say we wanted to render a `Dashboard` component whenever a user navigated to the `/dashboard` path. To do so, we'd render a `Route` that looked like this.

```
<Route
   path="/dashboard"
   element={<Dashboard />}
 />
```

The mental model I use for `Route` is that it always has to render something – either its `element` prop if the `path` matches the app's current location or `null`, if it doesn't.

You can render as many `Route`s as you'd like.

```
<Route path="/" element={<Home />} />
<Route path="/about" element={<About />} />
<Route path="/settings" element={<Settings />} />
```

### ▼ Routes

You can think of `Routes` as the metaphorical conductor of your routes. Whenever you have one or more `Route`s, you'll most likely want to wrap them in a `Routes`.

```
import { Routes, Route } from "react-router-dom";

function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/settings" element={<Settings />} />
      <Route path="*" element={<NotFound />} />
```

```
    </Routes>
  );
}
```

The reason for this is because it's `Routes` job is to understand all of its `children` `Route` elements, and intelligently choose which ones are the best to render.

### ▼ Link

Now that you know how to map the app's location to certain React components using `Routes` and `Route`, the next step is being able to navigate between them. This is the purpose of the `Link` component.

To tell `Link` what path to take the user to when clicked, you pass it a `to` prop.

```
<nav>
  <Link to="/">Home</Link>
  <Link to="/about">About</Link>
  <Link to="/settings">Settings</Link>
</nav>
```

#### Passing props via Links:

To pass data through a `Link` component to a new route, use **`Link`'s `state` prop**.

```
<Link to="/onboarding/profile" state={{ from: "occupation " }}>
  Next Step
</Link>
```

Anytime you pass data along via the `state` prop, that data will be available on the **`location`**'s **`state`** property, which you can get access to by using the custom **`useLocation`** Hook that comes with React Router.

```
import { useLocation } from 'react-router-dom'
function Profile ()
 {
  const location = useLocation()
  const { from } = location.state
```

```
  return (    ...  )
}
```

# 2. More about React-Router-Dom

### ▼ URL Parameters

Like function parameters allow you to declare placeholders when you define a function, URL Parameters allow you to declare placeholders for portions of a URL.

e.g.:

`<Route path="/wiki/:topicId" element={<Article />} />`

Now whenever anyone visits a URL that matches the `/wiki/:topicId` pattern (`/wiki/javascript`, `/wiki/Brendan_Eich`, `/wiki/anything`) , the `Article` component is rendered.

As of v5.1, React Router comes with a `useParams` Hook that returns an object with a mapping between the URL parameter(s) and its value.

```
import * as React from 'react'
import { useParams } from 'react-router-dom'
import { getArticle } from '../utils'

function Article () {
  const [article, setArticle] = React.useState(null)
  const { topicId } = useParams()

  React.useEffect(() => {
    getArticle(topicId)
      .then(setUser)
  }, [topicId])

  return (
    ...
  )
}
```

## ▼ Nested Routes

Nested Routes allow the parent Route to act as a wrapper and control the rendering of a child Route.

```
function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/messages" element={<Messages />}>
        <Route path=":id" element={<Chats />} />
      </Route>
      <Route path="/settings" element={<Settings />} />
    </Routes>
  );
}
```

Now, tell React Router **where** in the parent `Route` (`Messages`) should it render the child `Route` (`Chats`).

To do this, you use React Router's `Outlet` component.

```
import { Outlet } from "react-router-dom";

function Messages() {
  return (
    <Container>
      <Conversations />

      <Outlet />
    </Container>
  );
}
I
```

If the app's location matches the nested `Route`'s `path`, this `Outlet` component will render the `Route`'s `element`. So based on our `Routes` above, if we were at `/messages`, the `Outlet` component would render `null`, but if we were at `/messages/1`, it would render the `<Chats />` component.

## ▼ Programmatically Navigate

React Router offers two different ways to programmatically navigate, depending on your preference. First is the **imperative** `navigate` method and second is the declarative `Navigate` component.

To get access to the imperative `navigate` method, you'll need to use React Router's `useNavigate` Hook. From there, you can pass `navigate` the new **path** you'd like the user to be taken to when `navigate` is invoked.

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

function Register () {
  const navigate = useNavigate()

  return (
    <div>
      <h1>Register</h1>
      <Form afterSubmit={() => navigate('/dashboard')} />
    </div>
  )
}
```

## 3. Protected Routes

Private Routes in React Router (also called Protected Routes) require a user being authorized to visit a route (read: page). So if a user is not authorized for a specific page, they cannot access it.

```
function ProtectedRoutes() {
    const isLoggedIn = false;
    return {isLoggedIn ? <Outlet/> : <Navigate to="/login" replace />};
}

export default ProtectedRoute;
```

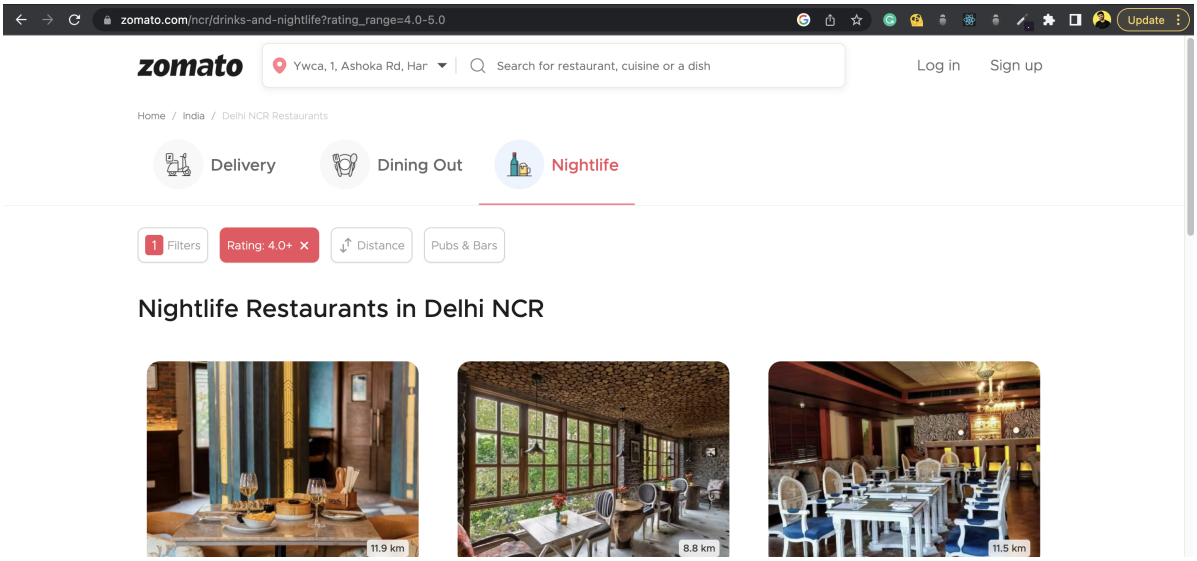
Now, you can surround any Routes inside the `ProtectedRoutes` Component.

## Further readings:

https://www.robinwieruch.de/react-router-private-routes/

## Assignments

1. Build a Platform with Login and Logout Functionalities. (Use State variables to store the authentication data)
2. Create a website with navbar to toggle in between multiple pages. Also add a TabLayout in one of the pages to toggle between multiple layouts. like this:




3. List all the recipes using this API and open each one separately on click in a separate component.: https://spoonacular.com/food-api/docs