React Router

Objectives

- Learn about React Router

There are a number of ways for us to introduce routes into our React project. Unlike some other frameworks, React doesn't provide a solution to use for us, we will need to install our own. The most popular library for this is React Router. React Router will be easy for us to install and we will need to add this as a dependency for each project that needs routing. From inside the project folder, we can install this using npm.

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

Next we will need to add some code into our project, for a quick demonstration let's start with our `App.js`.

Inside of App.js

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

function App() {
  return (
    <BrowserRouter>
      <h1>Routing Example</h1>
    </BrowserRouter>
  );
}

export default App;
```

This is a really basic app so far which will show up as a simple header with just an h1 element. We've added in a new component that we have imported from `"react-router-dom"` so let's talk about what it does.

Browser Router

The `BrowserRouter` exists to wrap around all of the components we want to be aware of our current route. It will enable us to work with the history of which routes our users have visited on our site. This component is usually at the root component of our application, so we will put it in app.js. We can even wrap it around the <App/> component in index.js!

For our next trick, we'll need two components: `Home` and `About`. For this example, we write them all inside of our App.js, but you can do the same thing by making them separate files, exporting and then importing them into the App.js!

What we want to do is have the `Home` component display when the user is at `http://localhost:3000/` and the `About` component display when the user is at `http://localhost:3000/about`. To accomplish this we're going to import two more components from `"react-router-dom"`: `Routes` and `Route`.

The `Routes` component will act something like the stage of a play and when we go from one scene (route) to another it will change out what component(s) we see. It will wrap around all of our individual routes.

The `Route` components will define the URL paths for a particular component using its `path` prop. When the URL's path matches the path prop as it was set in the Route component, then it will display only information passed into its element prop. Here, and in the majority of this course, we will be passing in a functional component like Home or About into the element prop.

Still inside of App.js

```
import React from "react";
import {
  BrowserRouter,
  Routes,
  Route,
  Link
} from "react-router-dom";
//Remember, we can write these functional components in their own files
    //to be exported and imported to the App component.
    //We've combined them here to simplify our example.
const Home = (props) => {
  return (
    <h1 style={{color: "red"}}>Home Component</h1>
  );
}

const About = (props) => {
  return (
    <h1 style={{color: "blue"}}>About Component</h1>
  );
}

function App() {
  return (
    <BrowserRouter>
      <h1>Routing Example</h1>
      <Routes>
        <Route path="/about" element={<About />} />
        <Route exact path="/" element={<Home />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App
```

Moving Around Our SPA

With our About and Home components' paths set within the Route component, we will need a way to navigate around our SPA (Single Page Application). Above, you may have noticed we imported another component from react-router-dom. The Link Component is going to work pretty similarly to the a tag you are all used to seeing in your html!

In fact, go ahead and write an a tag in your Home component with an href property of "/about" like we have here:

```
//The rest is removed for brevity
const Home = (props) => {
  return (
    <div>
        <h1 style={{color: "red"}}>Home Component</h1>
        <a href="/about">Go to About </a>
    </div>
  );
}
//The rest is removed for brevity
```

What did you notice?

Well, for starters it worked, but something else happened as well. Our DOM window in the browser was completely refreshed! When building applications using HTML, CSS and JavaScript outside of React, this is normal and expected behavior. The problem lies in our SPA and React. That refresh would've completely wiped out any state we may have written!

Let's try that again, this time using the Link component from react-router:

Note: If you inspect the rendered HTML in your browser, your "Link" looks like an "a" tag in the DOM for styling purposes! Keep that in mind when trying to apply default styles at the element level in your app.css.

App.js

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

const Home = (props) => {
  return (
    <div>
        <h1 style={{color: "red"}}>Home Component</h1>
        <Link to={"/about"}>Go to About </Link>
    </div>
  );
}

const About = (props) => {
  return (
    <div>
        <h1 style={{color: "blue"}}>About Component</h1>
        <Link to={"/"}>Go Home</Link>
    </div>
  );
}

function App() {
  return (
    <BrowserRouter>
      <h1>Routing Example</h1>
      <Routes>
        <Route path="/about" element={<About />} />
        <Route exact path="/" element={<Home />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App
```

What "Link" does is enables us to change the URL's path to what we specify in Link's "to" property without causing the page to refresh as traditional anchor tags would do. We can try this out with our new code above and see that when we click on the links they will change the

URL from http://localhost:3000/ to http://localhost:3000/about and back without triggering a page reload.

As you can see, this transition was much more seamless than before. In addition to not completely refreshing the DOM in our browser, which would have wiped state, this is much quicker and part of the performance boost that comes from working inside of a SPA like React!

Previous

Next