React Router

Previously, we used React to build a single-page app. This was just to demo the core concepts such as components, props and states and where to find the basic React code.

For a more complex site, you may find that you need to handle multiple pages. For better accessibility, you should have different routes to deal with different page content (this is typical website behaviour). To do this, we can use React Router—more specifically, we need the package for web apps: React Router DOM.

To install: `npm i react-router-dom`

Some resources

Some good reads about routing with React are:

- https://www.techomoro.com/how-to-create-a-multi-page-website-with-react-in-5-minutes/
- https://www.digitalocean.com/community/tutorials/how-to-handle-routing-in-react-apps-with-react-router
- https://reactrouter.com/en/main/router-components/browser-router

Note that these resources were used as a base for some of the instructions in this document.

.js vs .jsx files

In the above resources, you may find the use of .js as the file format for components (what we did before) and .jsx as the file format for components.

In practice, it doesn't matter which one you use because the code transpiler will interpret it correctly. If you want to differentiate between what would be regular JS and what would contain/use components, then use .jsx for component files (Airbnb uses .jsx as part of their coding standard). The main thing is to stay consistent so your code is predictable.

Basic steps for routing

1. Install react-router-dom.
2. Create components for each page's content. We'll be using React Router to render the appropriate component for a specific route.
3. Set up the routes. We'll do this in ***App.jsx*** (or ***App.js***) as our central code file. (I've used ***App.jsx*** in keeping with the boilerplate installed using Vite.)

Create components for your page content

This can be as simple as just creating a simple component with the return() containing the page contents (main content). We'll also use components for common page parts such as the header, footer and nav.

1. Just as we did before, create a ***/src/components*** folder where we can add our component files.
2. Create a ***Header.jsx***, ***Nav.jsx***, and ***Footer.jsx*** files within ***/src/components***. We'll add some basic code first and add more to it later as we go along.
3. Add the following code to ***Header.jsx*** and save. We'll modify this later once we add ***Nav.jsx*** code.

```
export default function Header() {
  return(
    <header id="header">
      <h2 id="site-name"><a href="/">My React site</a></h2>
    </header>
  );
}
```

4. Add the following code to ***Footer.jsx*** and save.

```
export default function Footer() {
  return(
    <footer id="footer">
      <div>&copy; HTTP5222, 2024.</div>
    </footer>
  );
}
```

5. Before we worry about the nav stuff, let's first create code for the contents of Home and About. Create a ***/src/routes*** folder and place ***Home.jsx*** and ***About.jsx*** within it.

> *Note about page content components*
>
> You can put page content components into the ***components*** folder if you want. I have separated them a little to differentiate between the page content and the smaller reusable components.

6. Now let's modify ***App.jsx*** to include the header and footer.

```
import Header from "./components/Header";
import Footer from "./components/Footer";

function App() {
  return (
    <div className="page">
      <Header />
```

```
      <Footer />
    </div>
  )
}

export default App
```

7. Add the following to ***Home.jsx*** and save.

```
export default function Home() {
  return(
    <main id="main">
      <h1>Welcome to my React app</h1>
      <p>This is some random content.</p>
    </main>
  );
}
```

8. Now add the following to ***About.jsx*** and save. This should give us enough to test out routes.

```
export default function About() {
  return(
    <main id="main">
      <h1>About</h1>
      <p>Random stuff about this company.</p>
    </main>
  );
}
```

Add route handling

React Router requires you to import `BrowserRouter` (in our case, for web apps), `Routes`, and `Route` from `react-router-dom`.

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

Once you've done that you can define routes like so:

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    {/* ...more Route paths... */}
  </Routes>
</BrowserRouter>
```

<BrowserRouter>

In the first resource listed on the first page of this document, the tutorial uses the Router component instead of BrowserRouter. BrowserRouter is used for regular URL paths and uses the History API (i.e. handles the history stuff for you), which makes it more suitable for web.

Resources

- https://medium.com/front-end-weekly/choosing-a-router-in-react-apps-85ae72fe9d78
- https://stackoverflow.com/questions/56707885/browserrouter-vs-router-with-history-push

<Routes>

Contains all routes in the app.

<Route path="<*relative_path_to_page*>***" element={***<*component_to_render*>***} />***

Specify the `path` to match (i.e. the relative URL path for the page in question) and the `element` (component) to render for that path.

Add routing code to app

In our case, we have some simple page paths to add.

1. Open ***App.js***. We'll put our routes here since we're putting our main logic here.
2. Add the import statement to ***App.js*** and include the <BrowserRouter> component as shown below. Note that if you don't need the for styling purposes, you can get rid of it because <BrowserRouter> can serve as the root element.

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Header from './components/Header';
import Footer from './components/Footer';

function App() {
  return (
    <div className="page">
      <BrowserRouter>
        <Header />
        <Routes>

        </Routes>
        <Footer />
      </BrowserRouter>
    </div>
  );
}
```

```
export default App;
```

- Notice the <Header /> and <Footer /> are within the <BrowserRouter> but not within <Routes>.

3. Include imports for the Home and About content.

```
//import page content components
import Home from './routes/Home';
import About from './routes/About';
```

4. Now add the <Route> elements within <Routes>.

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

5. Try testing the / and /about paths in the browser. You should see the content change.

<Link to="<path>" /> and <NavLink to="<path>" />

Within React Router DOM there are special components we can use: <Link> and <NavLink>.

Why not use <a> instead? <Link> and <NavLink> behave a little differently. Where <a> would reload the page, <Link> and <NavLink> only changes the page content **without reloading the entire page**. This results in faster performance.

<NavLink> has been created especially for menu links as it automatically adds an "active" class for active links. You can also specify a custom active class name using the `activeClassName` property.

Resources

- https://reactrouter.com/en/main/components/link
- https://reactrouter.com/en/main/components/nav-link

1. Let's now start to fill out our ***Nav.jsx*** file. Open ***Nav.jsx*** in your code editor.
2. Add the following code:

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

export default function Nav() {
  return(
    <nav id="main-navigation" aria-label="Main navigation">
      <ul className="menu">
        <li>
```

```
            <NavLink to="/">Home</NavLink>
          </li>
          <li>
            <NavLink to="/about">About</NavLink>
          </li>
        </ul>
      </nav>
    );
}
```

- React Router DOM stuff has been marked in yellow so you can see why you need the import.
- If you want to set a custom activeClassName for the nav links, add an `activeClassName="<your_class_name>"` in the NavLink opening tag.

3. Save and import in the ***Header.jsx*** file.

```
import Nav from "./Nav";
```

4. Add the <Nav /> in the ***Header.jsx*** return statement.

```
<header id="header">
  <h2 id="site-name"><a href="/">My React site</a></h2>
  <Nav />
</header>
```

5. Save everything and test it out in the browser. Notice that clicking on the menu links don't completely reload the page; only the main content is changing.
6. Inspect the link in the browser. By default, a class named "active" is added to the menu link