React Routes

Routes and Links

- In vanilla HTML, various HTML pages could be linked together using an <a href= '...' > anchor tag.

- In the same way, we can have components render in response to different URL paths entered in the browser address bar.

- So far the applications we've seen are Single Page Applications (SPA's) as there is really just one single component with sub-subcomponents that are selectively re-rendered

- Providing routes also gives the end user the ability to bookmark important paths.

Routing Dependencies

- Routing is considered an “add on” to the basic React project install

- The appropriate dependency must be listed in package.json:

```
"dependencies": {
  "@fortawesome/fontawesome-svg-core": "^6.4.0",
  "@fortawesome/free-regular-svg-icons": "^6.4.0",
  "@fortawesome/free-solid-svg-icons": "^6.4.0",
  "@fortawesome/react-fontawesome": "^0.2.2",
  "axios": "^1.7",
  "react": "^18.3.1",
  "react-dom": "^18.3.1",
  "react-router-dom": "^6.26.2"
 },
```

BrowserRouter

- The BrowserRouter tag will wrap around the sections of the application for which we want to establish routes.
    - In practice, this means that it will probably be in App.jsx

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

export default function App() {
  return (
    <div id="store-app">
        <BrowserRouter>
      …
        </BrowserRouter>
    </div>
    );
}
```

Routes and Route

- The next step is to define a Routes tag with one or more Route tags.
- Each Route tag maps a component to a URL path
- On this specific example, there are two routes:
  - /about will render the AboutUsView component
  - /products will render the ProductList View component.

```
export default function App() {
  return (
    <div id="book-app">
          <BrowserRouter>
                <Routes>
                    <Route
                    path="/about"
                    element={
                        <AboutUsView />
                    }
                  />
                  <Route
                    path="/products"
                    element={
                        <ProductListView />
                    }
                  />
                </Routes>
          </BrowserRouter>
    </div>
    );
}
```

Dynamic Routes

- Routes can contain dynamic pieces of information
  - Think about the path variables you worked on while setting up the server’s controllers

```
<Route
        path="/products/:id"
        element={
                <ProductDetailView />
        }
/>
```

On this example, the component <UserProfileView> will be rendered in response to any of the following paths:

/products/**1**
/products/**2**
/products/**3**
…etc

Dynamic Routes

- If a component is mapped to a route with a dynamic parameter, we can bring the value down from the URL to use in the component’s logic.
- On the mapped component, we will utilize useParams()

```
…/products/3
```

```
<Route
        path="/products/:id"
        element={
                <UserProfileView />
        }
/>
```

```
import { useParams } from ‘react-router-dom’;

export default function UserProfileView() {
  const { urlParams } = useParams();
  // urlParams is set to 3
```

Protected Routes

We will need to protect some routes so that only logged in users can view them. First we define this “wrapper” component:

```
import { Navigate } from 'react-router-dom';
import { useAuth } from './AuthProvider';

export default function ProtectedRoute ( { children } ) {

  // Get the user from the auth context
  const { user } = useAuth();

  // If there's an authenticated user, continue to child route
  if (user) {
    return children;
  }

  // Otherwise, send to login page
  return <Navigate to="/login" />;
};
```

- The prop children, will be referencing the component that’s protected.
- If the user object is populated, then we will display the children component
- Otherwise, we redirect them to the login page.

Linking

- There are two primary tags used for linking, these take on the same role as the anchor tag in regular HTML: <NavLink> and <Link>
- We will stick with <NavLink> as it has the advantage of applying a different styling format to the current component we're on.
- Within our JSX, we can say:

```
<NavLink to="/product">
    Profile
</NavLink>
```

On this example, we assume that **/product** is a valid <Route> tag.

Let’s implement a protected route

Let’s setup the Routes

Let’s setup a Dynamic Route