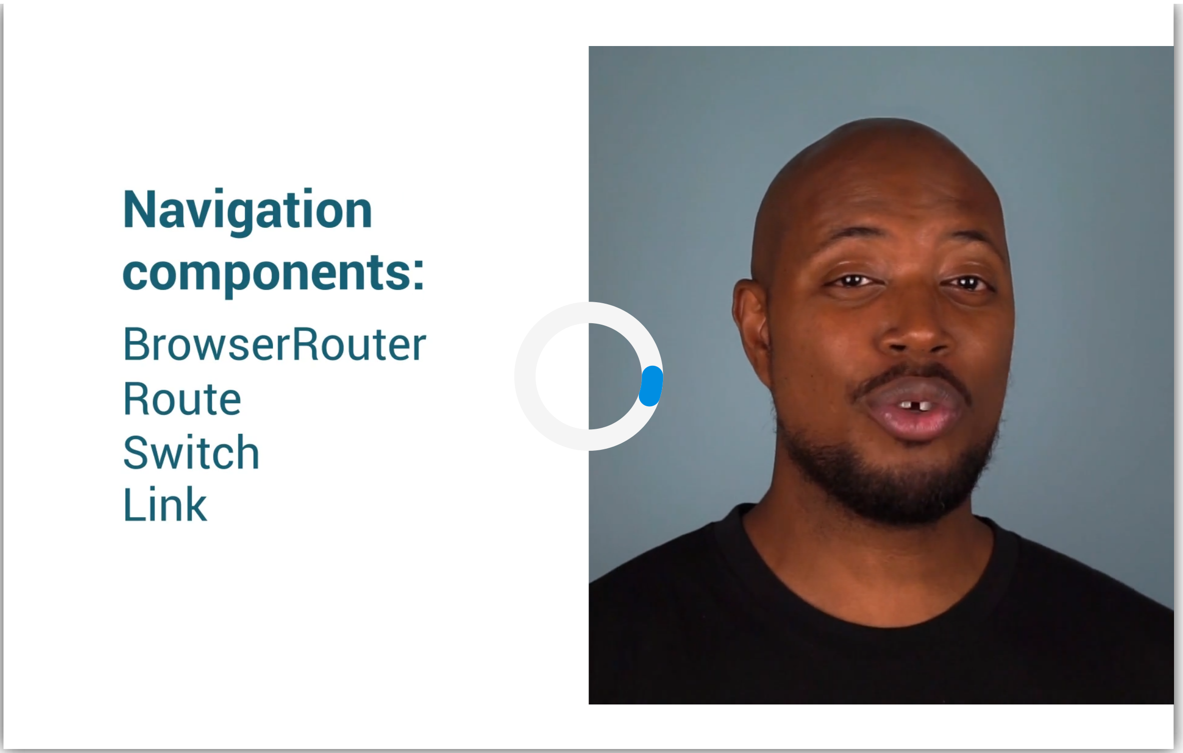
21.4.3 Set Up the Main URL Routes Using React Router

React is great for single-page applications. You can conditionally render components and pass in new props to make the page feel very dynamic and interactive. But what if you wanted your single-page app to feel like a multi-page app?

A multi-page app makes it easy for users to bookmark specific URLs, and use the forward and back buttons in their browser for quicker navigation. By default, you don't get that kind of functionality with React because you never leave the page.

Fortunately, there are libraries like React Router that can add client-side routing to your application while keeping the single-page responsiveness that React is known for. The idea behind React Router is that clicking a link like `/about` would actually stay on the same page but render a different component.

For an overview of how React Router works, watch the following video:



The best way to understand React Router, though, is to see it in action. Let's add it to the project.

First, create and checkout a new Git branch called `feature/routes`.

In the `client` folder, run the following command:

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

Note that there is a separate `react-router` package, which represents the core React Router library. For web-based projects, however, `react-router-dom` has everything you need in it.

IMPORTANT

> Sometimes when dealing with a lot of npm libraries that rely on related dependencies, you may receive some type of error on the command line or experience buggy behavior. This is usually resolved by removing

> both the `node_modules` directory and `package-lock.json` file from the directory and installing the dependencies again.

Open the `App.js` file and add the following `import` alongside the others:

```
import { BrowserRouter as Router, Route } from 'react-router-dom';
```

`BrowserRouter` and `Route` are components that the React Router library provides. We renamed `BrowserRouter` to `Router` to make it easier to work with.

While we're here, let's import the other page components with the following code:

```
import Login from './pages/Login';
import NoMatch from './pages/NoMatch';
import SingleThought from './pages/SingleThought';
import Profile from './pages/Profile';
import Signup from './pages/Signup';
```

Scroll down in the `App.js` file and update the `App` functional component to look like the following:

```
function App() {
  return (
    <ApolloProvider client={client}>
      <Router>
        <div className="flex-column justify-flex-start min-100-vh">
          <Header />
          <div className="container">
            <Route exact path="/" component={Home} />
            <Route exact path="/login" component={Login} />
            <Route exact path="/signup" component={Signup} />
            <Route exact path="/profile" component={Profile} />
            <Route exact path="/thought" component={SingleThought} />
          </div>
          <Footer />
        </div>
      </Router>
    </ApolloProvider>
  );
}
```

A few things have changed. We've wrapped the `<div className="flex-column">` element in a `Router` component, which makes all of the child components on the page aware of the client-side routing that can take place now.

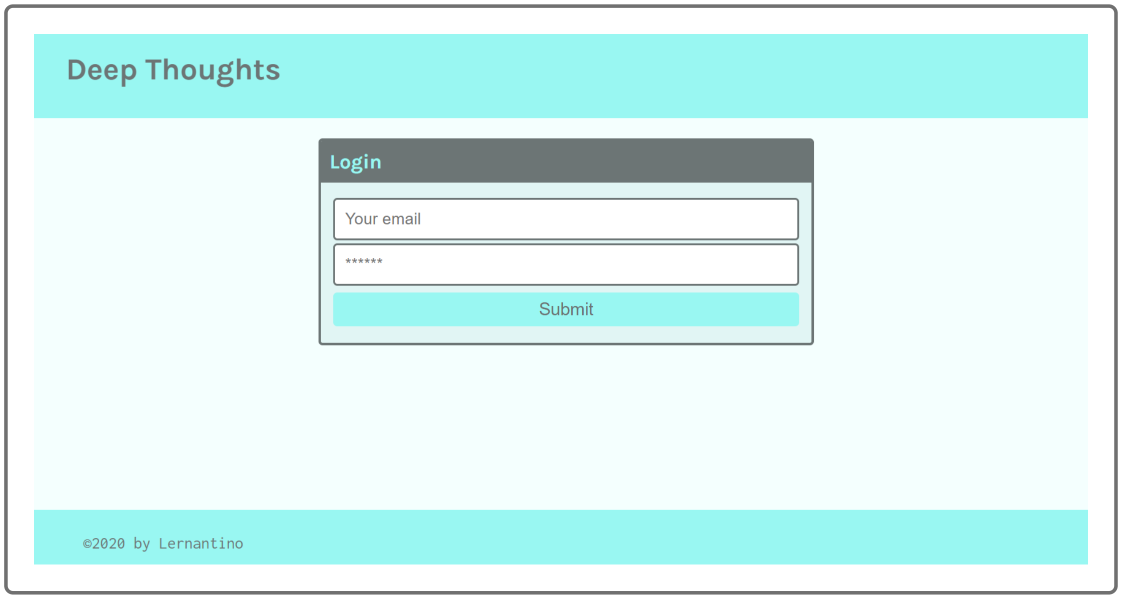
In the `<div className="container">` element, we've set up several `Route` components that signify this part of the app as the place where content will change according to the URL route. When the route is `/`, the `Home` component will render here. When the route is `/login`, the `Login` component will render.

Save your changes. If the app isn't running yet, be sure to run `npm start` from the root directory to start the back-end server and Create React App server at the same time.

In the browser, visit the following URLs and notice how the inner component changes while the surrounding `Header` and `Footer` remain the same:

- http://localhost:3000/ (http://localhost:3000/)
- http://localhost:3000/login (http://localhost:3000/login)
- http://localhost:3000/signup (http://localhost:3000/signup)
- http://localhost:3000/thought (http://localhost:3000/thought)

For example, the `/login` route should look like the following:



What happens if you visit a route that hasn't been defined, though, like `/about` or `/contact`? The page will still load, but nothing will display in between the `Header` and `Footer` components. Users wouldn't normally land on these empty pages, anyway, but if they do, we should at least display a "404 Not Found" message to avoid any confusion.

REWIND

With Express on the back end, you can define a catch-all route after your other routes with the following code:

```
app.get('*', (req, res) => {
  res.status(404).sendFile(path.join(__dirname, './public/404.htm
});
```

This would mean any route that hasn't been defined would be treated as a 404 and respond with your custom `404.html` file.

With React Router, you can use another component from the library called `Switch` that will allow you to set a catch-all route. This is very similar to a JavaScript `switch` statement that has a `default` case.

In `App.js`, update the `react-router-dom` import statement to look like the following:

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
```

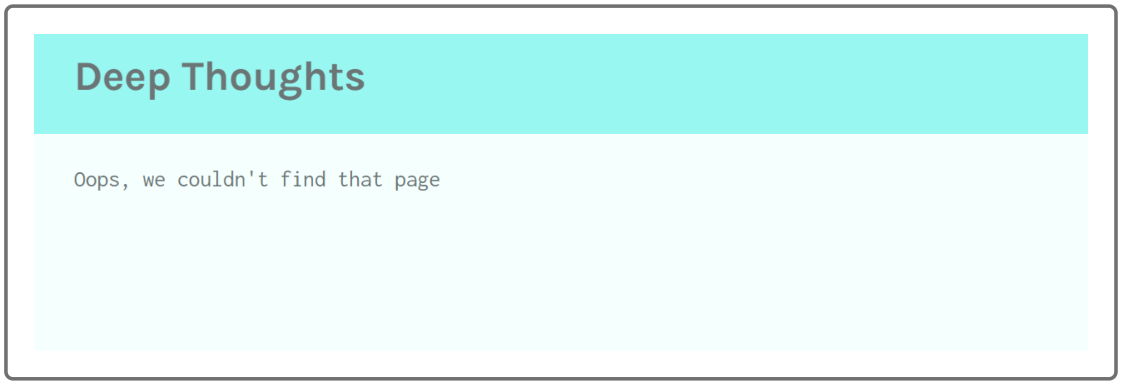
In the `App` component's JSX, update the `<div className="container">` element as follows:

```
<div className="container">
  <Switch>
    <Route exact path="/" component={Home} />
    <Route exact path="/login" component={Login} />
    <Route exact path="/signup" component={Signup} />
    <Route exact path="/profile" component={Profile} />
    <Route exact path="/thought" component={SingleThought} />

    <Route component={NoMatch} />
  </Switch>
</div>
```

We've wrapped all of the `Route` components in a `Switch` component and included one more `Route` at the end to render the `NoMatch` component. If the route doesn't match any of the preceding paths (e.g., `/about`), then users will see the 404 message.

The following image demonstrates this 404 message in action:



```
Oops, we couldn't find that page
```

Awesome, we have client-side routing set up, but the only way to visit these other pages is to manually type in the URLs in the address bar. In the next section, we'll expand on React Router's features to implement in-app navigation.