11.4 - Protecting Routes

**Instructions**

In this exercise, we’ll protect a route from unauthenticated access. We’re continuing our work with the same exercise app as we saw time, with some changes.

When the /secret route is accessed, we want to send the user to the /login route which features a rudimentary authentication mechanism. On that page, the user will see an input field to key in a password (which happens to be ‘pizza’). After the user logs in, the /secret route should be accessible.

1. Open App.js. You’ll notice the presence of a state property named ‘isLoggedIn’ and a Context provider component that offers access to this property as well as a function named doLogin which allows setting ‘isLoggedIn’ to true or false based on a passcode. This passcode has been hard-coded in as ‘pizza’. We also have a component named Login which displays a form element that allows users to key in the above passcode and hit enter. Let’s first include a Route component to ensure we can render this component on the /login route path. Add the following statement as a child to the Switch component in App.js

|  |
| --- |
| <Route path="/login">  <Login />  </Route> |

1. Next, let’s go into src/components/Login.js. We’re bringing in the isLoggedIn value and the doLogin function, thanks to the UserCtx context. We need to wire our input element to invoke the doLogin function, passing in the user’s keyed in passcode. If the user types in ‘pizza’, the isLoggedIn property will be set to true. If that happens, we want to redirect the user to the /secret route which we can do using the history object from the useHistory hook that React Router provides. We’ll use useEffect to action the redirection whenever the isLoggedIn is changed and is set to true. Modify the component as highlighted below.

|  |
| --- |
| **import** React, { useContext, useEffect } from "react";  **import** { useHistory } from "react-router-dom";  **import** UserCtx from "./UserCtx";  **const** Login = () => {  **const** { isLoggedIn, doLogin } = useContext(UserCtx);  **const** history = useHistory();  useEffect(() => {  isLoggedIn && history.replace({ pathname: "/secret" });  }, [isLoggedIn, history]);  **return** (  <div className="login">  <input  type="password"  id="login-passcode"  placeholder="Enter password & press enter..."  onKeyUp={e => {  **if** (e.keyCode === 13) {  *// Do Login*  doLogin(e.target.value);  }  }}  />  </div>  );  };  **export** **default** Login; |

1. But how do we reach the /login route. Ideally, accessing the /secret route should automatically redirect the user to /login if the user isn’t logged in. To protect a route, we’ll build a special version of the Route component that simply checks the authentication status and effects a redirection if needed. Create a new file in src/components named Protected.js and add the following code.

|  |
| --- |
| **import** React from "react";  **import** { Route, Redirect } from "react-router-dom";  **const** Protected = ({ isLoggedIn, children }) => (  <Route  render={() =>  isLoggedIn ? (  children  ) : (  <Redirect  to={{  pathname: "/login"  }}  />  )  }  />  );  **export** **default** Protected; |

1. The purpose of the Protected component that we built above is to redirect the user to the /login route if the isLoggedIn prop value is false. If however the value is ‘true’, the component will render its child elements. So, back in the App.js file, import the Protected.js in first.

|  |
| --- |
| **import** Protected from "./components/Protected"; |

1. We’ll now replace the Route component instance that renders our Secret component, with the Protected component as shown below.

|  |
| --- |
| <Protected isLoggedIn={isLoggedIn} path="/secret">  <Secret />  </Protected> |

1. That’s it. Try accessing /secret now and you’ll be redirected to the /login route. Type in ‘pizza’ and press enter, and you should now be able to see the Secret component on the /secret route. Subsequent access to the /secret route will continue to work until you refresh the page because we’re not persisting the user session anywhere, purely for simplicity.