Linking Back End and Front End

- Link Redux to components
- Create protected routes (pages only certain users can access based on their authentication)
- Keep a user logged in when they refresh or leave the page (in other words, until they either logout or the jwt expires)

Linking Redux to `Register` component

From the Register.js component, dispatch the `registerUser` action.

```
const onSubmit = e => {
    e.preventDefault();

    const newUser = {
        name: name,
        email: email,
        password: password,
        password2: password2
    };

    dispatch(registerUser(newUser, props.history));
};
```

Link Redux to `Login` component

From the Login.js component, dispatch the `loginUser` action.

```
const onSubmit = e => {
    e.preventDefault();

    const userData = {
        email: email,
        password: password
    };
    dispatch(loginUser(userData));
};
```

Create Dashboard component for logged-in users

In the `component` directory, create a `dashboard` directory and within it, a `Dashboard.js` file.

Place the following in `Dashboard.js`:

```
import React from "react";
import { useDispatch, useSelector } from "react-redux";
import { logoutUser } from "../../actions/authActions";

const Dashboard = () => {

  const user = useSelector(state => state.auth.user);

  const dispatch = useDispatch();

  const onLogoutClick = e => {
    e.preventDefault();
    dispatch(logoutUser());
  };

  return (
    <div style={{ height: "75vh" }} className="container valign-wrapper">
      <div className="row">
        <div className="landing-copy col s12 center-align">
          <h4>
            <b>Hey there,</b> {user.name.split(" ")[0]}
            <p className="flow-text grey-text text-darken-1">
              You are logged into a full-stack{" "}
              <span style={{ fontFamily: "monospace" }}>MERN</span> app 👏
              </p>
          </h4>
          <button
            style={{
              width: "150px",
              borderRadius: "3px",
              letterSpacing: "1.5px",
              marginTop: "1rem"
            }}
            onClick={onLogoutClick}
            className="btn btn-large waves-effect waves-light hoverable blue
accent-3"
          >
            Logout
            </button>
        </div>
      </div>
    </div>
  );
}

export default Dashboard;
```

Creating protected routes

In `components` directory, create a `private-route` directory and within it, a `PrivateRoute.js` file.

Place the following in `PrivateRoute.js`:

```
import React from "react";
import { Redirect } from "react-router-dom";
import { useSelector } from "react-redux";

const PrivateRoute = ({ component: Component }) => {

  const auth = useSelector(state => state.auth);

  return (
    auth.isAuthenticated === true ? (
      <Component />
    ) : (
        <Redirect to="/login" />
      )
  )
}

export default PrivateRoute;
```

Tying it all together in App.js

In `App.js`, we will

- Check `localStorage` for a token to keep the user logged in even if they close or refresh the app (e.g. until they log out or the token expires)
- Pull in the `Dashboard` component and define it as a `PrivateRoute`

```
import React from "react";
import { BrowserRouter, Route, Switch } from "react-router-dom";
import jwt_decode from "jwt-decode";
import setAuthToken from "./utils/setAuthToken";

import { setCurrentUser, logoutUser } from "./actions/authActions";
import { Provider } from "react-redux";
import store from "./store";

import Navbar from "./components/layout/Navbar";
import Landing from "./components/layout/Landing";
import Register from "./components/auth/Register";
import Login from "./components/auth/Login";
import PrivateRoute from "./components/private-route/PrivateRoute";
import Dashboard from "./components/dashboard/Dashboard";

import "./App.css";

// Check for token to keep user logged in
if (localStorage.jwtToken) {
  // Set auth token header auth
  const token = localStorage.jwtToken;
  setAuthToken(token);
```

```
  // Decode token and get user info and exp
  const decoded = jwt_decode(token);
  // Set user and isAuthenticated
  store.dispatch(setCurrentUser(decoded));
  // Check for expired token
  const currentTime = Date.now() / 1000; // to get in milliseconds
  if (decoded.exp < currentTime) {
    // Logout user
    store.dispatch(logoutUser());

    // Redirect to login
    window.location.href = "./login";
  }
}
const App = () => {

  return (
    <Provider store={store}>
      <BrowserRouter>
        <div className="App">
          <Navbar />
          <Switch>
            <Route exact path="/" component={Landing} />
            <Route path="/register" component={Register} />
            <Route path="/login" component={Login} />
            <PrivateRoute path="/dashboard" component={Dashboard} />
          </Switch>
        </div>
      </BrowserRouter>
    </Provider>
  );
}

export default App;
```

One last step

It wouldn't make sense for logged in users to be able to access the `/login` and `/register` pages. If a logged in user navigates to either of these, we should immediately redirect them to the dashboard.

To achieve this, add the following to `Register.js` and `Login.js`:

```
  if (isAuthenticated) {
    props.history.push("/dashboard");
  }
```