Programmatic Navigation Code-Along (CodeGrade)

 (https://github.com/learn-co-curriculum/react-hooks-react-router-programmatic-navigation-v6)  (https://github.com/learn-co-curriculum/react-hooks-react-router-programmatic-navigation-v6/issues/new)

Learning Goals

- Explain the use cases for programmatic navigation.
- Use the `useNavigate` hook to perform programmatic navigation.
- Use the `<Navigate>` component to perform programmatic navigation.

Introduction

So far, we've used a couple components from React Router to allow our users to navigate our React site: the `NavLink` and `Link` components. However, it would also be useful to direct our users to another page **without** them needing to click a link. For example:

- After logging in to the website, direct our user to the home page
- After logging out of the website, direct our user to the login page
- After creating a new item by filling out a form, direct our user to the detail page for that item

All of these actions require us to use **programmatic navigation** to change the browser URL, and show the user a new page in our application, **without** making the user click on a link.

We've included some files for you to code along in as we walk through the examples below. The files already have routing set up — we just need to update them to include programmatic navigation!

The `useNavigate` Hook

To enable programmatic navigation, we can use another custom hook from React Router: the `useNavigate` hook. Using it in our application is pretty straightforward:

- First, we import it into the component in which we want to use it; in this case, we'll be importing into our `App` component: `import { useNavigate } from "react-router-dom";` .
- Next, we need to invoke our `useNavigate` hook within our component and save the returned function in a variable. Let's call that variable `navigate` for simplicity: `const navigate = useNavigate()` .
- Then, whenever we want to use programmatic navigation, we'll simply pass the route we want to navigate our user to as an argument to the `navigate` function. `navigate("/")` .

Let's update our `App` component to include some programmatic navigation logic, as well as some state management logic that mocks user authentication:

```
// App.js
import { useState, useEffect } from "react";
// Add useNavigate to import
import { Outlet, useNavigate} from "react-router-dom";
import NavBar from "./components/NavBar";

function App() {
    // Add code to mock user authentication
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const navigate = useNavigate();

  const login = () =>{
    setIsLoggedIn(true);
  }

  const logout = () =>{
    setIsLoggedIn(false);
  };
```

```
    // Add programmatic navigation for login and logout
  useEffect(() =>{
    if (isLoggedIn) {
        // navigates to Home route if user is logged in
      navigate("/");
    } else {
        // navigates to Login route if user is logged out
      navigate("/login");
    };
  }, [isLoggedIn]);

  return (
    <div className="app">
      <NavBar logout={logout} />
        { /*Pass login function to Outlet as context */}
      <Outlet context={login}/>
    </div>
  );
};

export default App;
```

> **Note:** We placed our call to `navigate` within our `useEffect` because we want to navigate our user *after* they've successfully logged in or out. By placing the state that dictates whether or not a user is logged in or out within the dependency array of our `useEffect`, we can programmatically navigate our user whenever a change in state occurs. This approach means we only call `navigate` once our state has updated. You don't always have to put `navigate` inside of a `useEffect`, but it makes sense to do so in this case.

Now, we can update our `NavBar` component to handle user logout functionality.

```
// NavBar.js
import { NavLink} from "react-router-dom";
```

```
import "./NavBar.css"
function NavBar({ logout }) {

  return (
    <nav>
      <NavLink
        to="/"
        className="nav-link"
      >
        Home
      </NavLink>
      <NavLink
        to="/about"
        className="nav-link"
      >
        About
      </NavLink>
      {/* Add the logout function to handle the onClick event */}
      <button onClick={logout}>Logout</button>
    </nav>
  );
};

export default NavBar;
```

And we can update our `Login` component to handle user login.

```
// Login.js
import { useState } from "react";
import { useOutletContext } from "react-router-dom";

function Login() {
  // Access the login function passed as context
```

```
  const login = useOutletContext();
  const [formData, setFormData] = useState({
    username: "",
    password: "",
  });

  function handleChange(e) {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value,
    });
  };

  // Create a function that calls the login function when the form is submitted
  function handleLogin(e) {
    e.preventDefault();
    login();
  };

  return (
    <form onSubmit={handleLogin}>
      <label for="username">Username</label>
      <div>
        <input
          id="username"
          type="text"
          name="username"
          value={formData.username}
          onChange={handleChange}
        />
      </div>
      <label for="password">Password</label>
      <div>
```

```
          <input
            id="password"
            type="password"
            name="password"
            value={formData.password}
            onChange={handleChange}
          />
        </div>
        <button type="submit">Login</button>
    </form>
    );
};

export default Login;
```

The Navigate Component

In addition to the `useNavigate` hook, React Router also provides a special component for redirecting users to a new location: the `Navigate` component.

To use the `Navigate` component, we simply invoke it the same way we would invoke any other component, then pass it a `to` prop that points toward a route endpoint: `<Navigate to="/login" />` .

This component is particularly useful in cases where you need to handle some conditional rendering. For example, in the App component below, instead of rendering our `NavBar` component we can render a `Navigate` component that will navigate to the `/login` endpoint if the user is not logged in:

```
// App.js
import { useState, useEffect} from "react";
import { Outlet, Navigate, useNavigate} from "react-router-dom";
import NavBar from "./components/NavBar";

function App() {
```

```
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const navigate = useNavigate();


   const login = () =>{
    setIsLoggedIn(true);
  };

  const logout = () =>{
    setIsLoggedIn(false);
  };

  useEffect(() =>{
    if (isLoggedIn) {
      navigate("/");
    } else {
      navigate("/login");
    }
  }, [isLoggedIn]);

  return (
    <div className="app">
{/* Add conditional rendering so users have to be logged in to see pages on the site */}
      {isLoggedIn ? <NavBar logout={logout}  /> : <Navigate to="/login" />}
      <Outlet context={login}/>
    </div>
  );
};

export default App;
```

This means that any user who visits our app and is not logged in will only see the login page, and won't be able to use the `NavBar` to navigate to other parts of our website.

Conclusion

React Router gives us full control over how to navigate users around our website. In general, the preferred approach is to use the `<Link>` and `<NavLink>` components to let users perform navigation by clicking links.

However, there are certain scenarios when we want to navigate a user to a new page after they perform some other type of action, like submitting a form or logging out. React Router provides two tools to help us with these scenarios: the `useNavigate` hook and the `<Navigate>` component.

Resources

- **React Router useNavigate (https://reactrouter.com/en/main/hooks/use-navigate)**
- **React Router History (https://reactrouter.com/en/main/start/concepts#history-and-locations)**
- **Navigate (https://reactrouter.com/en/main/components/navigate)**

This tool needs to be loaded in a new browser window

Load Programmatic Navigation Code-Along (CodeGrade) in a new window