Client-Side Routing with React Router

Springboard

Download Demo Code

Goals

- Describe what client-side routing is and why it's useful
- Compare client-side routing to server-side routing
- Implement basic client-side routing with React Router

Server-Side Routing

- "Server-side routing" is the traditional pattern
  - Clicking a **<a>** link causes browser to request page & replace entire DOM
- Server decides what HTML to return based on URL requested

Client-Side Routing

Faking Client Side Routing

demo/nonrouted/src/App.js

```
function App() {
  const [page, setPage] = useState("home");

  function goToPage(newPage) {
    setPage(newPage);
  }

  function showPage() {
    if (page === "home") return <Home />;
    if (page === "eat") return <Eat />;
    if (page === "drink") return <Drink />;
  }

  return (
    <main>
      <nav>
        <a onClick={() => goToPage("drink")}>Drink</a>
        <a onClick={() => goToPage("eat")}>Eat</a>
        <a onClick={() => goToPage("home")}>Home</a>
      </nav>
      {showPage()}
    </main>
  );
}
```

That's okay

- It does let us show different "pages"
  - All in the front-end, without loading new pages from server
- But we don't get
  - A different URL as we move around "pages"
  - The ability to use the back/forward browser buttons ➡️ ⬅️ 😔
  - Any way to bookmark a "page" on the site 🤦 📄 😱

Real Client-Side Routing

React can give us real Client-Side Routing

Client-Side Routing: What?

- Client-side routing handles mapping between URL bar and page user sees via *browser* rather than via *server*.
- Sites that exclusively use client-side routing are **single-page applications**.
- We use JavaScript to manipulate the URL bar with a Web API called History

React Router

Installation

To get started with React Router, install ***react-router-dom***.

```
$ npx create-react-app routed
$ cd routed
$ npm install react-router-dom@5.2.1
```

Including the Router

demo/routed/src/App.js

```
import { BrowserRouter, Route } from "react-router-dom";

function App() {
  return (
    <div className="App">
      <BrowserRouter>
      </BrowserRouter>
    </div>
  );
}
```

Wrap the things that need routing with ***<BrowserRouter>***

There are other routers besides ***BrowserRouter*** — don't worry about them.

Note: Other types of routers

If you read through the React Router docs, you'll see examples of other types of routers. Here's a brief description of them:

- ***HashRouter***: this router is designed for support with older browsers that may not have access to the full history API. In such cases, you can still get single-page type functionality by inserting an anchor (#) into the URL. However, this does not provide full backwards-compatibility: for this reason, the React Router documentation recommends ***BrowserRouter*** over ***HashRouter*** if possible.
- ***MemoryRouter*** This router mocks the history API by keeping a log of the browser history in memory. This can be helpful when writing tests, since tests are typically run outside of a browser environment.
- ***NativeRouter*** This router is designed for React Native applications.
- ***StaticRouter*** This is a router that never changes location. When would you ever use this? According to the docs, "This can be useful in server-side rendering scenarios when the user isn't actually clicking around, so the location never actually changes. Hence, the name: static. It's also useful in simple tests when you just need to plug in a location and make assertions on the render output."

Routes and Links

A Sample Application

App.js

```
import React from "react";

import Home from "./Home";
import Eat from "./Eat";
import Drink from "./Drink";
import NavBar from "./NavBar";

import { BrowserRouter, Route } from "react-router-dom";

function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <NavBar />
        <Route exact path="/drink">
          <Drink />
        </Route>
        <Route exact path="/eat">
          <Eat />
        </Route>
        <Route exact path="/">
          <Home />
        </Route>
      </BrowserRouter>
    </div>
  );
}

export default App;
```

Route Component

```
<Route exact path="/eat"><Eat /></Route>
```

- ***Route*** component acts as translation service between routes & components.
  - Tell it a path to look for in URL, and what to render when it finds match.
- Props you can set on a ***Route***:
  - ***exact*** *(optional bool)*, does path need to match *exactly? /foo/bar* in URL bar will match `path="/foo"` — but match won't be *exact*.
  - ***path***: path that must match

That example: "when path is exactly /eat, render ***<Eat />*** component"

Note: Stick with child components for rendering

If you look in the React Router docs, you'll see that there are actually four different ways to pass a component into ***Route***: you can use the ***render*** prop, the ***component*** prop, the ***children*** prop, or just make the component that you want to render a child of the ***Route*** component.

You can learn about the differences between them later; we'll be using child components most of the time, as this is flexible enough to handle our use cases.

Navigation Links & Nav Bars

Link Component

- The ***<Link>*** component acts as a replacement for **<a>** tags.
- Instead of an ***href*** attribute, ***<Link>*** uses a ***to*** prop.
- Clicking on ***<Link>*** does *not* issue a GET request.
  - JS intercepts click and does client-side routing

```
<p>Go to <Link to="/drink">drinks</Link> page</p>
```

NavLink Component

- ***<NavLink>*** is just like link, with one additional feature
  - If at page that link would go to, the **<a>** gets a CSS class of *active*
    - This lets you have CSS like this:

```
.MyNavBarClass a {
  color: white;
}

.MyNavBarClass a.active {
  color: black;
}
```

  - You should include an ***exact*** prop here as well
- Very helpful for navigation menus

Note: Other ways to stylize current navigation link

By itself, ***NavLink*** puts the *active* class on the **<a>** element for ***NavLink*** components that match the path. This lets you handle the appearance of active navigation links in your CSS, as you normally would.

There are two options you may find useful:

- ***activeClassName***: if you'd prefer a different class name to be put onto the active links, you can specify it here.
- ***activeStyle***: if you'd prefer to specify the appearance of active links directly in your JavaScript, you can pass a object to this of CSS properties to put on active links, like:

```
const ACTIVE_STYLES = {
  fontWeight: "bold",
  color: "black",
};

function MyNavBar() {
  return (
    <nav className="MyNavBarClass">
      <NavLink exact to="/" activeStyle={ ACTIVE_STYLES }>Home</NavLink>
      <NavLink exact to="/eat" activeStyle={ ACTIVE_STYLES }>Eat</NavLink>
      <NavLink exact to="/drink" activeStyle={ ACTIVE_STYLES }>Drink</NavLink>
    </nav>
  );
}
```

A Sample Navigation Bar

Nav.js

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

function NavBar() {
  return (
    <nav className="NavBar">
      <NavLink exact to="/">
        Home
      </NavLink>
      <NavLink exact to="/eat">
        Eat
      </NavLink>
      <NavLink exact to="/drink">
        Drink
      </NavLink>
    </nav>
  );
}

export default NavBar;
```

Wrap-Up

- With React-Router, you can get "client-side routing"
  - "Moving around site" doesn't require server load
  - URL bar, bookmarks, and back/forward button still work
- You need to
  - Wrap parts that use routing with a ***<BrowserRouter>***
  - Use a ***<Route>*** component for each different route
  - For navigation links to those routes, use a ***<Link>***

Client-side vs. Server-side

Client-side Routing

- Potentially improved UI/UX
- More modern architecture
- Potentially worse SEO

Server-side Routing

- Page reload with every URL change
- More traditional architecture
- Potentially better SEO

Which is better? **It depends.**

Looking Ahead

Coming Up

- More on route props
- Redirecting with React Router
- How to organize your routes