Client-Side Routing with React Router

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

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

Client-Side Routing

Faking Client Side Routing

demo/nonrouted/src/App.js

```
class App extends Component {
  state = {page: "home"};

  goToPage(page) {
    this.setState({page: page});
  }

  showRightPage() {
    if (this.state.page === "home") return <Home />;
    else if (this.state.page === "eat") return <Eat />;
    else if (this.state.page === "drink") return <Drink />;
  }

  render() {
    return (
      <main>
        <nav>
          <a onClick={() => this.goToPage('home')}>Home</a>
          <a onClick={() => this.goToPage('eat')}>Eat</a>
          <a onClick={() => this.goToPage('drink')}>Drink</a>
        </nav>
        { this.showRightPage() }
      </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 📖 📄 😱
    - More complex route/pattern matching

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 the content a 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***.

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

Including the Router

demo/routed/src/index.js

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

ReactDOM.render(
    <BrowserRouter>
      <App />
    </BrowserRouter>,
    document.getElementById("root")
);
```

Wrap your ***<App />*** renders with a ***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, Switch, and Links

A Sample Application

App.js

```
import React, { Component } from "react";
import Home from "./Home";
import Eat from "./Eat";
import Drink from "./Drink";
import NavBar from "./NavBar";
import {Route, Switch} from "react-router-dom";

class App extends Component {
  render() {
    return (
      <div className="App">
        <NavBar />
        <Switch>
          <Route
            exact path="/"
            render={() => <Home />} />
          <Route
            exact path="/eat"
            render={() => <Eat />} />
          <Route
            exact path="/drink"
            render={() => <Drink />} />
        </Switch>
      </div>
    );
  }
}

export default App;
```

Route Component

```
<Route exact path="/eat" render={() => <Eat />} />
```

- ***Route*** component acts as translation service between routes & components.
  - Tell it 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
  - ***render*** what should be rendered (expects function that returns JSX)

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

> **Note: Stick with render**
>
> If you look in the React Router docs, you'll see that there are actually three different ways to pass a component into ***Route***: you can use either the ***render*** prop, the ***component*** prop, or the ***children*** prop. Unfortunately, this is one of the most confusing parts of the library, as these all do similar but slightly different things.
>
> We'll use ***render*** exclusively, and this should be fine for all of your needs.

Switch Component

App.js

```
          <Switch>
            <Route
              exact path="/"
              render={() => <Home />} />
            <Route
              exact path="/eat"
              render={() => <Eat />} />
            <Route
              exact path="/drink"
              render={() => <Drink />} />
          </Switch>
```

- Since we only expect one of these to match, wrap in ***<Switch>***
- This stops searching once it finds a match
- This is *almost* always what you want

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 stylize links to “page you are already at” using the `activeStyle` (in-line) or `activeClassName` props
    - You should include an ***exact*** prop here as well
- Very helpful for navigation menus

A Sample Navigation Bar

Nav.js

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

class NavBar extends Component {
  render() {
    const activeStyle = {
      fontWeight: "bold",
      color: "mediumorchid"
    };
    return (
        <nav>
          <NavLink exact to="/"
            activeStyle={activeStyle}>Home</NavLink>
          <NavLink exact to="/eat"
            activeStyle={activeStyle}>Eat</NavLink>
          <NavLink exact to="/drink"
            activeStyle={activeStyle}>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 contents of your ***<App>*** 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