React Router

Agenda

- Getting started
- Basic Components
- Basic routing
- URL Parameters
- Redirects (Auth)
- Recursive Paths
- Navigation Prompt
- Exercise

Why

React Router keeps your UI in sync with the URL. It has a simple API with powerful features like lazy code loading, dynamic route matching, and location transition handling built right in. Make the URL your first thought, not an after-thought.

Getting started

npm install -g create-react-app

create-react-app AppName

cd AppName

npm install react-router-dom

Required Imports

```
import React from 'react';

import {
  BrowserRouter as Router,
  Route,
  Link
} from 'react-router-dom';
```

Basic Routing

```
const App = () => (
  <Router>
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/about">About</Link></li>
        <li><Link to="/topics">Topics</Link></li>
      </ul>
/* The reason we are using Link here instead of anchor(a) tag is because Link component exposed by the react-router-dom behaves same as anchor tag but stops the page from refreshing. Instead is just replaces the current Component with the one specified for the current path. */
```

Basic Routing

```
      <Route exact path="/" component={Home}/>
      <Route path="/about" component={About}/>
      <Route path="/topics" component={Topics}/>
    </div>
  </Router>
);
```

/* The Route component mainly takes two props to work. First is “path”, .which is the path that is to be matched and second is the “component” that is to be rendered when that specific path has been matched. Now here in this example we can also see a third prop “exact”. If this prop is not passed the Home component will always get rendered because “/” will always be there is the path. So we provide the prop exact which tells the router to render the the Home component if the path matches exactly. */

Url Parameters

```
/* Lets assume our links look like this */
 <ul>
        <li><Link to="/randomId1">David</Link></li>
        <li><Link to="/randomId2">Michelle</Link></li>
        <li><Link to="/randomId3">Someone</Link></li>
 </ul>
/* and on click we want to show the data of the person whose name was clicked. Instead of
making three different components for this we can just make a single component for
showing data based on id. */
```

Url Parameters

```
/* Our route will look like this */
<Route path="/:id" component={UserDetails}/>
/* UserDetails Component */
const UserDetails = ({ match }) => (
  <div>
    <h3>ID: {match.params.id}</h3>
  </div>
)
/* The “match” prop is passed by the react-router to the component */
```

Redirection (Authentication)

```
const PrivateRoute = ({ component, ...rest }) => (
  <Route {...rest} render={props => (
    Auth.isAuthenticated ? (
      <Component {...props} />
    ) : (
      <Redirect to={{
        pathname: '/login',
        state: { from: props.location }
      }}/>
    )
  )}/>
)
```

Redirection (Authentication)

```
/* Using PrivateRoute Component */

<PrivateRoute path="/some-protected-route" component={ProtectedComponent}/>

/*
The state that we passes in the redirect component will be available in ProtectedComponent
in this.props.location.state.
*/
```

Recursive Path

/* Because React Router is just components, you can do crazy things like having recursive routes */

```
function UserDetails({ match }) {
    const id = getReletedUserDetails();
    return (
        <Route path={`${match.url}/:id`} component={UserDetails} />
        )
    }
```

/* Recursive routes aren't the most pragmatic thing in the world, but they really show off the benefits of React Router's component based approach to routing.
The main idea here is that since React Router is just components, theoretically, you can create recursive and therefor infinite routes */

Navigation Prompt

```
/* Required Imports */

import { Prompt } from "react-router-dom";

/* There is a feature of this function method on Prompt that is very useful. When receiving the next location.

 imagine if you had a wizard form and just the section of the form was changing but the component where you held your form state would still be mounted. */

 <Prompt
    when={!!this.state.name}
    message={location => `Are you sure you want to go to ${location.pathname}`}
 />
 <div>Nice looking profile! What's your name?</div>
 <input value={this.state.name} onChange={e => this.setState({ name: e.target.value
})} />
```

Navigation Prompt

/*

- This component doesn't take any children, you render it and control it with a few props.
- The first prop is the when component. This is what enables the prompt. When it's true it'll show the prompt message or if it's false will let any navigation happen.
- Then we have our message, this message can either be a string or a function. The function method will receive the next location to navigate to.

*/

CODE DEMO

Exercise

- Create Application with pages as mention:
  - Home Page
  - About us Page
  - Login Page
  - User Info (Private Page)

Your Queries Please!!!