FREE eBook

LEARNING React Router

Free unaffiliated eBook created from **Stack Overflow contributors.**

#react-router

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: react-router

It is an unofficial and free React Router ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official React Router.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with React Router

Remarks

React Router is a popular and complete routing library for React.js that keeps UI in sync with the URL. It supports lazy code loading, dynamic route matching, and location transition handling, and was initially inspired by Ember's router.

TODO: this section should also mention any large subjects within react-router, and link out to the related topics. Since the Documentation for react-router is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
1.0.0	2015-11-09
1.0.1	2015-12-05
1.0.2	2015-12-08
1.0.3	2015-12-23
2.0.0	2016-02-10
2.0.1	2016-03-09
2.1.0	2016-04-11
2.1.1	2016-04-11
2.2.0	2016-04-13
2.2.1	2016-04-14
2.2.2	2016-04-14
2.2.3	2016-04-15
2.2.4	2016-04-15
2.3.0	2016-04-18
2.4.0	2016-04-28
2.4.1	2016-05-19

Version	Release Date
2.5.0	2016-06-22
2.5.1	2016-06-23
2.5.2	2016-07-01
2.6.0	2016-07-18
2.6.1	2016-07-29
2.7.0	2016-08-20
2.8.0	2016-09-09
2.8.1	2016-09-13
3.0.0	2016-10-24
3.0.1	2017-01-12
3.0.2	2017-01-18
3.0.3	2017-03-28
3.0.4	2017-04-09
3.0.5	2017-04-10
4.0.0	2017-04-12
4.1.0	2017-04-12
4.1.1	2017-04-12

Examples

Installation and Setup

To install React Router, just run the npm command

```
npm install --save react-router
```

And you're done. This is literally all you have to do to install react router.

Please Note : `react-router` is dependent on `react`, So make sure you install `react` as well.

To set up:

using an ES6 transpiler, like babel

```
import { Router, Route, Link } from 'react-router'
```

not using an ES6 transpiler

```
var Router = require('react-router').Router
var Route = require('react-router').Route
var Link = require('react-router').Link
```

Installation using UMD build

A build is also available on npmcdn. You can include the script like this:

```
<script src="https://npmcdn.com/react-router/umd/ReactRouter.min.js"></script>
```

The library will be available globally on `window.ReactRouter`.

Hello World with React and React Router

Once you've installed `react` and `react-router`, Its time to use both of them together.

The syntax is very simple, you specify the `url` and the `component` you want to render when that url is opened

```
<Route path="hello" component={ HelloComponent } />
```

This means when the url path is `hello`, Render the component `HelloComponent`

FILENAME: `app.js`

```
'use strict';

import React from 'react';
import { render } from 'react-dom';
import { Router, browserHistory, Link } from 'react-router';

// These are just demo components which render different text.

let DashboardPage = () => (
  <div>
    <h1>Welcome User</h1>
    <p>This is your dashboard and I am an example of a stateless functional component.</p>
    <Link to="/settings">Goto Settings Page</Link>
  </div>
)

let SettingsPage = () => (
  <div>
    <h1>Manage your settings</h1>
    <p>display the settings form fields here...or whatever you want</p>
    <Link to="/">Back to Dashboard Page</Link>
  </div>
)
```

```
let AuthLoginPage = () => (
  <div>
    <h1>Login Now</h1>
    <div>
      <form action="">
        <input type="text" name="email" placeholder="email address" />
        <input type="password" name="password" placeholder="password" />
        <button type="submit">Login</button>
      </form>
    </div>
  </div>
)

let AuthLogoutPage = () => (
  <div>
    <h1>You have been successfully logged out.</h1>
    <div style={{ marginTop: 30 }}>
      <Link to="/auth/login">Back to login page</Link>
    </div>
  </div>
)

let ArticlePage = ({ params }) => (
  <h3>Article {params.id}</h3>
)

let PageNotFound = () => (
  <div>
    <h1>The page you're looking for doesn't exist.</h1>
  </div>
)

// Here we pass Router to the render function.
render( (
    <Router history={ browserHistory }>

      <Route path="/" component={ DashboardPage } />
      <Route path="settings" component={ SettingsPage } />

      <Route path="auth">
        <IndexRoute component={ AuthLoginPage } />
        <Route path="login" component={ AuthLoginPage } />
        <Route path="logout" component={ AuthLogoutPage } />
      </Route>

      <Route path="articles/:id" component={ ArticlePage } />

      <Route path="*" component={ PageNotFound } />

    </Router>
  ), document.body );
```

Route Parameters : Router path can be configured to take parameters so that we can read the parameter's value at the component. The path in `<Route path="articles/:id" component={ ArticlePage } />` have a `/:id`. This `id` variable serves the purpose of path parameter and it can be accessed at the component `ArticlePage` by using `{props.params.id}`.

If we visit `http://localhost:3000/#/articles/123` then `{props.params.id}` at component `ArticlePage` will be resolved to 123. But visiting url `http://localhost:3000/#/articles`, will not work because

there is no id parameter.

The route parameter can be made **optional** by writing it in between a pair of parenthesis:
`<Route path="articles(/:id)" component={ ArticlePage } />`

If you want to use **sub routes**, then you can do

```
<Route path="path" component={ PathComponent }>
  <Route path="subpath" component={ SubPathComponent } />
</Route>
```

- when `/path` is accessed, `PathComponent` will be rendered
- when `/path/subpath` is is accessed, `PathComponent` will be rendered and `SubPathComponent` will be passed to it as `props.children`

You can use `path="*"` to catch all the routes that doesn't exist and render `404 page not found` page.

Getting Started

This *getting started* assumes you are working with create-react-app, or something equivalent using Babel and all the goodies out there.

Also check out the great documentation right here.

First, install react-router-dom:

`npm install react-router-dom` or `yarn add react-router-dom`.

Then, create a component that exists of a basic Navbar with two items and basic pages:

```
import React from 'react'
import { BrowserRouter, Route, Link } from 'react-router-dom'

const Home = () => (
  <div>
    <p>We are now on the HOME page</p>
  </div>
)

const About = () => (
  <div>
    <p>We are now on the ABOUT page</p>
  </div>
)

const App = () => (
  <BrowserRouter>
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/about">About</Link></li>
      </ul>
      <hr/>
      <Route path="/" component={Home}/>
      <Route path="/about" component={About}/>
```

```
    </div>
  </BrowserRouter>
)
export default App
```

Let's go step by step through this code:

- `import React from 'react'`: Make sure you import `React`
- `import { BrowserRouter as Router, Route, Link } from 'react-router-dom'` split up:
- `BrowserRouter` is the actual router itself. Make sure to wrap your component within the `BrowserRouter` component.
- `Route` is one particular route that can be navigated to
- `Link` is a component that produces an `<a href="...">` tag, which you can use as a hyperlink.

- `const Home` is a function that returns the homepage.
- `const About` is a function that returns the About page.

- `const App` is the main component:
- `<BrowserRouter>` is the JSX component that wraps the components in which you want to use the `<Route>` component.
- '`is a single element to wrap all JSX inside the`BrowserRouter` in.
- `<ul>` is the Navbar. It contains a link to Home and a link to About.
- `<li><Link to="/">Home</Link></li>` links to the homepage. You can see that, since the link refers to "/", an empty relative path renders the homepage.
- `<li><Link to="/about">About</Link></li>` links to the About page.
- `<Route path="/" component={Home}/>` describes which component should be rendered if the relative path is `"/"`.
- `<Route path="/about" component={About}/>` describes which component should be rendered if the relative path is `"/about"`.

Lot to learn from here, but hopefully this explains the fundamentals, so from here you can continue your learnings.

Read Getting started with React Router online: https://riptutorial.com/react-router/topic/5546/getting-started-with-react-router

Chapter 2: React Router 4 with TypeScript

Introduction

Some samples of integrating TypeScript with `react-router` 4.x.

The goal is to preserve as much type safety as possible.

How to accomplish this with TypeScript is not obvious when following the projects documentation.

Examples

Basic routing

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

class Home extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>HOME</div>
        <div><Link to='/one'>Goto Page One</Link></div>
        <div><Link to='/two'>Goto Page Two</Link></div>
      </div>);
  }
}

class One extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>ONE</div>
        <Link to='/'>Goto Home</Link>
      </div>
    );
  }
}

class Two extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>TWO</div>
        <Link to='/'>Goto Home</Link>
      </div>
    );
  }
}

ReactDOM.render(
  <Router>
    <div>
```

```
      <Route exact path="/" component={Home} />
      <Route exact path="/one" component={One} />
      <Route exact path="/two" component={Two} />
    </div>
  </Router>

  , document.getElementById('root')
);
```

This represents very basic `react-router` routing with TypeScript `React.Component` classes.

Routing with typed parameters

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

// define React components for multiple pages
class Home extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>HOME</div>
        <div><Link to='/details/id123'>Goto Details</Link></div>
      </div>);
  }
}

interface DetailParams {
  id: string;
}

interface DetailsProps {
  required: string;
  match?: match<DetailParams>;
}

class Details extends React.Component<DetailsProps, any> {
  render() {
    const match = this.props.match;
    if (match) {
      return (
        <div>
          <div>Details for {match.params.id}</div>
          <Link to='/'>Goto Home</Link>
        </div>
      );
    } else {
      return (
        <div>
          <div>Error Will Robinson</div>
          <Link to='/'>Goto Home</Link>
        </div>
      )
    }
  }
}

ReactDOM.render(
```

```
  <Router>
    <div>
      <Route exact path="/" component={Home} />
      <Route exact path="/details/:id" component={(props) => <Details required="some string"
{...props} />} />
    </div>
  </Router>

  , document.getElementById('root')
);
```

In order to preserve type safety for the `Details` component which has a required property named `required`, the `<Route>` definition defines an anonymous function-based component which composes a new component of type `<Details>` and specifying the `required` property.

The spread operator is utilized to re-apply the `props` passed to the anonymous function-based component onto the composed `<Details>` component.

The `match` property is defined as optional, since it's filled in dynamically by `react-router`, we, unfortunately, cannot define it as required property. This means a type guard is required when accessing the values later.

Routing with typed parameters and injected properties

This solution is more involved, leveraging custom TypeScript decorators which inject `match`, `history` and/or `location` data into your `React.Component` class, which gets you full type safety without needing any type guards, as the previous example required.

// `Routed.ts` - defines decorators

```
import { RouteComponentProps, match } from 'react-router';
import { History, Location } from 'history';

// re-export for convenience, uppercase match to be in line with everything else
export { History, Location, match as Match };

// names for the three types we support injecting
type InjectionPropType = 'location' | 'history' | 'match';

// holder for a given property to be injected as a specific type
class InjectionProp {
  prop: string;
  type: InjectionPropType;
}

// a store, key = class name (constructor.name) and array of InjectionProp's for that class
// this will be filled in by the three property decorators @RoutedMatch, @RoutedLocation and
@RoutedHistory
class InjectionStore {
  [key: string]: InjectionProp[];
}

// instance of the store
const store: InjectionStore = {};
```

```
// type guard for RouteComponentProps
function instanceOfRouteProps<P>(object: any): object is RouteComponentProps<P> {
  return 'match' in object && 'location' in object && 'history' in object;
}

// class level decorator, wraps the constructor with custom one which injects
// values into instances based on the InjectionStore instance
export function Routed<T extends { new (...args: any[]): {} }>(constructor: T) {

  // get the class name from the constructor
  const className = (constructor as any).name;

  // return a new class with a new constructor which calls super(..)
  return class extends constructor {

    constructor(...args: any[]) {
      super(args);

      // if there is a React props passed as arg[0]
      if (args.length >= 1) {

        const routeProps = args[0];

        // check type guard to see if the React props is enriched with RouteComponentProps by
react-router
        if (instanceOfRouteProps(routeProps)) {
          // check if the current class has any registered properties to be injected
          if (store[className]) {
            const injectionProps = store[className];
            // iterate over properties to inject
            for (let i = 0; i < injectionProps.length; i++) {
              const injectionProp = injectionProps[i];
              // inject the specified property with the appropriate type
              switch (injectionProp.type) {
                case 'match':
                  (this as any)[injectionProp.prop] = routeProps.match;
                  break;
                case 'history':
                  (this as any)[injectionProp.prop] = routeProps.history;
                  break;
                case 'location':
                  (this as any)[injectionProp.prop] = routeProps.location;
                  break;
              }
            }
          }
        }
      }
    }
  }
}

// generic property decorator, registers a classes property for inject in the store above
function RoutedInjector(proto: any, prop: string, type: InjectionPropType): any {
  const className = proto.constructor.name;
  if (!store.hasOwnProperty(className)) {
    store[className] = [];
  }
  store[className].push({
    prop: prop,
    type: type
```

```
  });
}

// property decorator for Match instances
export function RoutedMatch(proto: any, prop: string): any {
  RoutedInjector(proto, prop, 'match');
}

// property decorator for Location instances
export function RoutedLocation(proto: any, prop: string): any {
  RoutedInjector(proto, prop, 'location');
}

// property decorator for History instances
export function RoutedHistory(proto: any, prop: string): any {
  RoutedInjector(proto, prop, 'history');
}
```

// index.ts

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

import { History, Location, Match, Routed, RoutedHistory, RoutedLocation, RoutedMatch } from
'./Routed';

// define React components for multiple pages
class Home extends React.Component<any, any> {
  render() {
    return (
      <div>
        <div>HOME</div>
        <div><Link to='/details/id123'>Goto Details</Link></div>
      </div>);
  }
}

interface DetailParams {
  id: string;
}

interface DetailsProps {
  required: string;
}

@Routed
class Details extends React.Component<DetailsProps, any> {

  @RoutedMatch
  match: Match<DetailParams>;

  @RoutedLocation
  location: Location;

  @RoutedHistory
  history: History;

  render() {
    return (
```

```
      <div>
        <div>Details for {this.match.params.id} on location {this.location.pathname}</div>
        <span
          onClick={(e) => this.history.push('/')}
          style={{ textDecoration: 'underline', cursor: 'pointer' }}
        >Goto Home</span>
      </div>
    );

  }
}

ReactDOM.render(
  <Router>
    <div>
      <Route exact path="/" component={Home} />
      <Route exact path="/details/:id" component={(props) => <Details required="some string"
{...props} />} />
    </div>
  </Router>

  , document.getElementById('root')
);
```

This example uses custom decorators to inject some `react-router` specific data instances using property decorators `@RoutedMatched`, `@RoutedLocation` and `@RoutedHistory` on a `React.Component` decorated with `@Routed`.

The end result here is that the `react-router` specific data types are now decoupled entirely from the custom `React.Component` properties and state. An added benefit is that they are no longer optional properties, which means you don't need type guards to safely access their values.

The `history` parameter shows getting access to the `History` object from the `history` package (a dependency of `react-router`), which can be used, as shown, to programmatically manipulate the browser history.

The `location` parameter carries information about the current location

Read React Router 4 with TypeScript online: https://riptutorial.com/react-router/topic/9815/react-router-4-with-typescript

Credits

S. No	Chapters	Contributors
1	Getting started with React Router	Alex, Community, Diego V, imdzeeshan, Ming Soon, Random User, Sventies, Vishnu Y S, YasserKaddour
2	React Router 4 with TypeScript	Alex