Title: Introduction React Router

Answer:

REACT ROUTER 4 IS THE PERFECT TOOL TO LINK TOGETHER THE URL AND YOUR REACT APP. REACT ROUTER IS THE DE-FACTO REACT ROUTING LIBRARY, AND IT'S ONE OF THE MOST POPULAR PROJECTS BUILT ON TOP OF REACT.

React Router is the de-facto React routing library, and it’s one of the most popular projects built on top of React.

React at its core is a very simple library, and it does not dictate anything about routing.

Routing in a Single Page Application is the way to introduce some features to navigating the app through links, which are **expected** in normal web applications:

1. The browser should **change the URL** when you navigate to a different screen
2. **Deep linking** should work: if you point the browser to a URL, the application should reconstruct the same view that was presented when the URL was generated.
3. The **browser back (and forward) button** should work like expected.

Routing links together your application navigation with the navigation features offered by the browser: the **address bar** and the **navigation buttons**.

React Router offers a way to write your code so that **it will show certain components of your app only if the route matches what you define**.

INSTALLATION

With npm:

```
npm i --save react-router-dom
```

With Yarn:

```
yarn add react-router-dom
```

TYPES OF ROUTES

React Router provides two different kind of routes:

- `BrowserRouter`
- `HashRouter`

One builds classic URLs, the other builds URLs with the hash:

```
https://application.com/dashboard   /* BrowserRouter */
https://application.com/#/dashboard /* HashRouter    */
```

Which one to use is mainly dictated by the browsers you need to support. `BrowserRouter` uses the History API, which is relatively recent, and not supported in IE9 and below. If you don’t have to worry about older browsers, it’s the recommended choice.

COMPONENTS

The 3 components you will interact the most when working with React Router are:

- `BrowserRouter` , usually aliased as `Router`
- `Link`
- `Route`

`BrowserRouter` wraps all your Route components.

`Link` components are - as you can imagine - used to generate links to your routes

`Route` components are responsible for showing - or hiding - the components they contain.

BROWSERROUTER

Here's a simple example of the BrowserRouter component. You import it from react-router-dom, and you use it to wrap all your app:

```
import React from 'react'
import ReactDOM from 'react-dom'
import { BrowserRouter as Router } from 'react-router-dom'

ReactDOM.render(
  <Router>
      <div>
        <!-- -->
      </div>
  </Router>,
  document.getElementById('app')
)
```

A BrowserRouter component can only have one child element, so we wrap all we're going to add in a `div` element.

LINK

The Link component is used to trigger new routes. You import it from `react-router-dom` , and you can add the Link components to point at different routes, with the `to` attribute:

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

ReactDOM.render(
  <Router>
      <div>
        <aside>
          <Link to={`/dashboard`}>Dashboard</Link>
          <Link to={`/about`}>About</Link>
        </aside>
        <!-- -->
      </div>
  </Router>,
  document.getElementById('app')
)
```

ROUTE

Now let’s add the Route component in the above snippet to make things actually work as we want:

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

const Dashboard = () => (
  <div>
    <h2>Dashboard</h2>
    ...
  </div>
)

const About = () => (
  <div>
    <h2>About</h2>
    ...
  </div>
)

ReactDOM.render(
  <Router>
    <div>
      <aside>
        <Link to={`/`}>Dashboard</Link>
        <Link to={`/about`}>About</Link>
      </aside>

      <main>
        <Route exact path="/" component={Dashboard} />
        <Route path="/about" component={About} />
      </main>
    </div>
  </Router>,
  document.getElementById('app')
)
```

When the route matches `/`, the application shows the Dashboard component.

When the route is changed by clicking the “About” link to `/about`, the Dashboard component is removed and the About component is inserted in the DOM.

Notice the `exact` attribute. Without this, `path="/"` would also match `/about`, since `/` is contained in the route.

MATCH MULTIPLE PATHS

You can have a route respond to multiple paths simply using a regex, because `path` can be a regular expressions string:

```
<Route path="/(about|who)/" component={Dashboard} />
```

INLINE RENDERING

Instead of specifying a `component` property on `Route`, you can set a `render` prop:

```
<Route
  path="/(about|who)/"
  render={() => (
    <div>
      <h2>About</h2>
      ...
    </div>
  )}
/>
```

MATCH DYNAMIC ROUTE PARAMETER

You already saw how to use static routes like

```
const Posts = () => (
  <div>
    <h2>Posts</h2>
    ...
  </div>
)

//...

<Route exact path="/posts" component={Posts} />
```

Here's how to handle dynamic routes:

```
const Post = ({match}) => (
  <div>
    <h2>Post #{match.params.id}</h2>
    ...
  </div>
)

//...

<Route exact path="/post/:id" component={Post} />
```

In your Route component you can lookup the dynamic parameters in `match.params`.

`match` is also available in inline rendered routes, and this is especially useful in this case, because we can use the `id` parameter to lookup the post data in our data source before rendering Post:

```
const posts = [
  { id: 1, title: 'First', content: 'Hello world!' },
  { id: 2, title: 'Second', content: 'Hello again!' }
]

const Post = ({post}) => (
  <div>
    <h2>{post.title}</h2>
    {post.content}
  </div>
)

//...

<Route exact path="/post/:id" render={({match}) => (
  <Post post={posts.find(p => p.id === match.params.id)} />
)} />
```

Tags: react