DigitalHouse> Coding School

Fullstack Web Development

React Router

React no es un Framework, es una Librería para trabajar las vistas en el front-end. Por esta razón es que no tenemos por defecto un sistema de ruteo como sí lo tiene Node.js. React Router viene a solucionar justamente esto, permitiéndonos construir lo que normalmente se conoce como una SPA (*Single Page Application*).

Instalación e implementación

React Router es un paquete como cualquier otro. Para instalarlo seguimos el procedimiento habitual. Claramente, para utilizarlo deberíamos previamente tener una aplicación de React creada y funcionando.

```
npm install react-router-dom
```

Una vez instalado el paquete, en nuestro **index.js** deberemos importar el módulo y englobar nuestro componente principal `<App />` dentro del componente que nos provee React Router: `<BrowserRouter>`.

```
import { BrowserRouter } from 'react-router-dom'

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

Los componentes de React Router

Además del `<BrowserRouter>`, React Router nos brinda un par de componentes adicionales para trabajar los enlaces y las decisiones de ruteo.

Link

Link nos permite generar un enlace a una ruta de React. Su estructura es similar a la de un anchor o etiqueta `<a href="una-direccion"></a>`, con la diferencia que en lugar de usar “a” usaremos “Link” y en lugar de “href” usamos “to”.

```
<Link to="/">Home</Link>
```

Veamos el código completo, importando el componente e implementando:

```
import React from 'react';
import { Link } from 'react-router-dom';

const Navbar = () =>
   return (
       <nav>
           <ul>
               <li><Link to="/">Home</Link></li>
               <li><Link to="/about">About</Link></li>
               <li><Link to="/contact">Contact</Link></li>
           </ul>
       </nav>
   )
}

export default Navbar;
```

NavLink

El NavLink es casi idéntico al Link que ya vimos, la diferencia es que tendrá en cuenta la dirección donde se encuentra el usuario y, en caso de coincidir con la propiedad `to`, agrega la clase de CSS `active` al componente.

```
import React from 'react';
import { NavLink } from 'react-router-dom';

const Navbar = () =>
   return (
       <nav>
           <ul>
               <li><NavLink to="/">Home</NavLink></li>
               <li><NavLink to="/about">About</NavLink></li>
               <li><NavLink to="/contact">Contact</Link></li>
           </ul>
       </nav>
   )
}

export default Navbar;
```

Navlink va a buscar coincidencias de manera parcial. Esto quiere decir que si estamos en la página de about (`/about`) el NavLink de la home (`/`) también se va a marcar como activo. Para evitar esto podemos usar la propiedad `exact` para decirle que el NavLink solo se debe marcar como activo si la coincidencia es exacta.

```
<li><NavLink exact to="/">Home</NavLink></li>
```

Route

Route será el componente que nos permitirá hacer la lógica de **ruteo**.

React Router se encargará de hacer una comparación parcial entre el path donde se encuentra el usuario contra las posibles rutas y dibujarán todos los componentes que coincidan.

Recibirá dos propiedades:

- El `path` al que deberá responder. Recibirá el mismo valor que tiene la propiedad `to` del Link correspondiente.
- El `component` que deberá renderizar.

Por ejemplo, si la ruta actual fuera `/users` pasaría lo siguiente con estos componentes:

```
<Route path="/" component={ Home } />
// No hay coincidencia

<Route path="/users" component={ UserList } />
// Coincidencia exacta, se renderiza

<Route path="/users/login" component={ UserLogin } />
// Coincidencia parcial, se renderiza

<Route path="/users/register" component={ UserRegister } />
// Coincidencia parcial, se renderiza
```

Renderizando con props

En caso de que necesitemos pasarle propiedades al componente que vamos a renderizar para la ruta, podemos usar los children. Para eso, abrimos y cerramos la etiqueta del componente y, dentro de él, ponemos el o los componentes que queramos renderizar para esa ruta.

```
<Route path="/">
   <Home title="Página principal">
</Route >
```

Rutas dinámicas

Route nos permite también atajar rutas dinámicas (como cuando nos llega un ID por ejemplo), utilizando los mismos comodines que vimos en Node.js: los dos puntos seguidos del nombre que queremos darle al parámetro de la ruta.

```
<Route path="/product/:id" component={ ProductDetail } />
```

Estos parámetros de la ruta, estarán luego disponibles como `props` dentro de cualquiera de los componentes que se hayan renderizado dentro de esa ruta. Para acceder a ellos lo podremos hacer a través de: `props.match.params.nombre-del-parametro`.

```
const Product = (props) =>
   return (
       <div>
           <h3>ID del producto: { props.match.params.id }</h3>
       </div>
   )
}
```

Switch

Hace un rato hablamos de que `<Route />` por defecto va a renderizar todos los componentes que coincidan con la ruta actual. En cambio, `<Switch />` solo renderiza los componentes de la primera coincidencia que ocurra con la ruta actual.

```
import { Route, Switch } from 'react-router-dom';

const App = () =>

   return (

         <Switch>

           <Route exact path="/">

               <Home />

           </Route>

           <Route>

               <NoMatch />

           </Route>

       </Switch>

   )}
```

Switch y las rutas anidadas

Las reglas de ruteo pueden anidarse, esto quiere decir que si dentro de nuestra página de about tenemos subpáginas, las podemos trabajar con un nuevo `<Switch />`.

Por lo general, cuando implementemos rutas anidadas, tendremos que utilizar el `exact` en las rutas de más arriba.

Redirect

Por último, puede llegar el caso de que queramos hacer una redirección a otra ruta de nuestra aplicación. Para esos casos podemos utilizar el componente `<Redirect>`. Al igual que los enlaces, recibirá la propiedad `to` para indicar a dónde debemos dirigir al usuario.

```
<Redirect to="/"/>
```