React Router - Parte 2

Tiempo estimado: 20min

En la lección anterior, introdujimos React Router, mostrando cómo el encaminador presenta unas vistas u otras, atendiendo a la interacción del usuario con la aplicación. Ahora, vamos a mostrar algunos aspectos que no presentamos o que requieren una atención más detenida.

Comenzamos la lección presentando cómo hay que crear los enlaces en las aplicaciones que usan React Router. A continuación, describimos la historia de URLs, muy importante en las aplicaciones SPA. Después, exponemos el contexto generado por el componente <Router> y que podemos utilizar en los componentes de la aplicación. Finalmente, mostramos cómo hay que cambiar de una vista a otra explícitamente y cómo actualizar algunas propiedades del documento como su título y su descripción.

Al finalizar la lección, el estudiante sabrá:

- Para qué se usa el componente <Link> de React Router.
- Cómo configurar el estilo de los enlaces cuando se consideran activos.
- Qué es la historia de URLs.
- Cómo configurar la historia de URLs.
- Qué contexto genera automáticamente React Router.
- Cómo movernos de una vista a otra explícitamente.
- Cómo actualizar los elementos <title> y <meta name="description"> a medida que el usuario se mueve por las distintas vistas de la aplicación.

Enlaces

Un enlace (*link*) es la representación de un elemento HTML <a> en la aplicación React. A través de ellos, el usuario navega por las distintas vistas de la aplicación. Cuando tengamos enlaces a vistas, no hay que usar directamente un componente HTML <a>, sino un componente Link, ubicado en el paquete react-router.

Este componente es compuesto y tiene la siguiente sintaxis:

```
<Link to=ruta
      activeClassName=nombre
      activeStyle=estilo
      onlyActiveOnIndex
      onClick=controlador>
  lo que presentar al usuario como enlace
</Link>
```

Propiedad to

La propiedad to se utiliza para indicar la ruta enlazada. Ejemplo:

```
<Link to={`/noticia/${news.slug}`}>{news.title}</Link>
```

Enlaces activos

Un enlace activo (*active link*) es aquel cuyo URL corresponde al que actualmente está presentando el encaminador. React Router permite indicar cómo estilizar un enlace si la ruta indicada en la propiedad to es la activa.

Se puede utilizar un nombre de clase CSS, en cuyo caso, hay que hacerlo mediante la propiedad activeClassName o bien un estilo en línea, mediante activeStyle. Veamos unos ejemplos ilustrativos:

```
<Link to={`/noticia/${news.slug}`} activeClassName="active">{news.title}</Link>
<Link to={`/noticia/${news.slug}`} activeStyle={{color: "red"}}>{news.title}</Link>
```

React Router tiene un comportamiento especial en las rutas compuestas. Se consideran activas cuando

nos encontramos en ellas mismas así como en cualquiera de sus subrutas. Si sólo deseamos que se considere activa cuando nos encontramos en ella misma, hay que indicar la propiedad booleana onlyActiveOnIndex.

Evento Click

Cada vez que el usuario hace clic en el enlace, se genera el evento Click. Si es necesario, podemos indicar un controlador para este evento. Este controlador tiene como parámetro el evento sintético generado, fn(evt).

> **Nota**. El sistema de eventos de React se expone detenidamente en una lección posterior.

He aquí un ejemplo ilustrativo:

```
<Link to={`/noticia/${news.slug}`} onClick={(evt) => this.handleNewsClick(evt)}>
  {news.title}
</Link>
```

Historia

La historia (*history*) representa el registro de acontecimientos pasados realizados por el usuario a lo largo de la sesión de la aplicación React. React Router se apoya en la historia de URLs (*URL history*), la historia que mantiene los URLs accedidos por el usuario durante la sesión en la aplicación. Para poder hacerlo, el encaminador necesita monitorizar los cambios de URL solicitados por el usuario, para lo que se apoya en algún tipo de biblioteca que le permita hacerlo fácilmente.

En React Router, podemos utilizar varios tipos de historias, siendo los siguientes los más utilizados:

- Historia del navegador (*browser history*). Usa la API estandarizada implementada por los propios navegadores.

  Las rutas de la aplicación serán similares a mi.punto.com/ruta/a/vista.
- Historia de hash (*hash history*). Usa la almohadilla (#) en las rutas.

  Las rutas de la aplicación serán similares a mi.punto.com/#/ruta/a/vista.

Atendiendo a la historia que utilicemos, veremos los URLs escritos de una manera u otra. Es muy importante saber cómo serán, sobre todo si la aplicación se debe indexar en los buscadores de Internet como, por ejemplo, Google o Bing.

Historia del navegador

Se recomienda la utilización de la historia del navegador (*browser history*), aquella que se apoya en la API estandarizada que implementan y proporcionan los navegadores webs.

Sus principales características son las siguientes:

- Las rutas serán del estilo /la/vista/a/presentar.

  Son simples y sencillas de leer e interpretar.
- La aplicación React debe ser servida por un servidor web como, por ejemplo, una aplicación Node bajo Express o un servidor web puro y duro como Apache o Nginx.

  Si la aplicación se ejecutará a partir de un archivo solicitado por el navegador al sistema de ficheros de nuestra máquina, no se puede usar.
- El encaminador utilizará la API History proporcionada por los navegadores.

Es muy importante tener claro que la aplicación debe servirla un servidor web. Y es muy importante comprender el funcionamiento de una aplicación SPA, para poder configurar correctamente el servidor web. Recordemos que es el encaminador de la aplicación el que se encarga de monitorizar los cambios de URL solicitados por el usuario cuando navega por la aplicación. Para ello, detecta el cambio, consulta su mapa de rutas y, entonces, carga la vista correspondiente. No hay comunicación con el servidor web, a menos que la aplicación necesite algo de él y lo acceda mediante Ajax, por ejemplo, mediante la API Fetch.

¿Qué ocurre si el usuario va al navegador y accede directamente a un URL de una vista de la

aplicación? Básicamente, tendremos un problema. Para el servidor, el único URL que conoce de la aplicación, salvando claro está los de las imágenes, los archivos CSS, etc., es el URL de la raíz de la aplicación. Cuando recibe una petición HTTP asociada a la raíz de la aplicación, es cuando envía el archivo index.html y, a partir de él, se carga la aplicación React, debido al elemento <script>.

Ahora bien, si tenemos una ruta en la aplicación, digamos, /noticia/quien-usa-react, la cual tiene asociada una determinada vista, debe quedar muy, pero que muy claro que el URL lo gestiona la aplicación React, más concretamente el encaminador. No lo hace el servidor. Si el usuario accede directamente a ese URL en su navegador, el servidor web enviará muy probablemente un error 404 Not Found.

Para resolver el problema, lo que hay que hacer es que cuando el servidor web reciba un URL de un recurso que no tiene en su *hosting*, remita siempre el archivo index.html de la aplicación. Es así como el usuario recibirá la aplicación. Entonces, el encaminador detectará que el usuario está accediendo a /noticia/quien-usa-react y cargará la vista correspondiente.

Así pues, cuando usemos la historia del navegador, deberemos configurar el servidor para que envíe automáticamente index.html cuando tenga la necesidad de enviar una repuesta HTTP 404 Not Found.

Cómo configurar la historia del navegador

Para usar esta historia, hay que configurar el encaminador. Para ello, hay que importar el objeto browserHistory del paquete react-router y asignárselo a la propiedad history del componente <Router>, el cual se define, por convenio y buenas prácticas, en el archivo app/index.jsx.

Ejemplo:

```
//imports
import {Router, browserHistory} from "react-router";
…

//componente raíz
ReactDOM.render(
  <Router history={browserHistory}>…</Router>,
  document.getElementById("react-app")
);
```

Historia de hash

La historia de hash (*hash history*) añade una almohadilla (#) a los URLs. Éstas aparecerán en el navegador como /#/la/vista/a/presentar?_k=ckuvup. Observe la diferencia con respecto a los URLs que se usan con la historia del navegador. Son menos elegantes y muy poco recomendadas si la aplicación debe seguir pautas SEO para su posicionamiento en Internet.

Es la historia que hay que utilizar si la aplicación se va a acceder directamente desde nuestro sistema de archivos.

Cómo configurar la historia de hash

En este caso, en vez de importar browserHistory, hay que importar hashHistory:

```
//imports
import {Router, hashHistory} from "react-router";
…

//componente raíz
ReactDOM.render(
  <Router history={hashHistory}>…</Router>,
  document.getElementById("react-app")
);
```

Contexto del encaminador

Recordemos que el contexto (*context*) es un mecanismo a través del cual pasar datos de arriba abajo, es decir, de un componente a sus descendientes. De esta manera, se puede intercambiar datos entre un componente y sus descendientes sin necesidad de pasarlos a través de las propiedades o el estado.

React Router añade la propiedad router al contexto. Esta propiedad se utiliza principalmente en los

controladores de las vistas para transicionar, explícitamente, a otra vista. Por ejemplo, si acabamos de insertar un nuevo producto y deseamos remitir al usuario a la vista que lista los productos, lo podemos hacer mediante el objeto router del contexto.

Recordemos que para que un componente pueda hacer uso del contexto, hay que definir qué propiedades accederá mediante su propiedad estática contextTypes. En nuestro caso, en los componentes, tendremos algo como:

```
static get contextTypes() {
  return {
    router: React.PropTypes.any
  };
}
```

Transición explícita

Podemos cambiar de una vista a otra, básicamente de dos maneras. La primera, mediante un componente <Link>. Cuando el usuario pulse en el enlace, se producirá el cambio de vista. La otra opción es hacerlo mediante JavaScript explícitamente. En este segundo caso, se utiliza la propiedad router del contexto.

this.context.router.push()

Mediante el método push(), solicitamos al encaminador que cambie a la vista indicada:

```
push(path)
```

Parámetro	Tipo de datos	Descripción
path	string	Ruta a la que cambiar.

Veamos un ejemplo ilustrativo:

```
this.context.router.push("/ruta/a/la/vista");
```

Este método añade una nueva entrada a la historia de navegación.

this.context.router.replace()

Mediante el método replace(), podemos hacer lo mismo, pero ahora, no se añade una nueva entrada a la historia de navegación, sino que se reemplaza la actual por la nueva:

```
replace(path)
```

Parámetro	Tipo de datos	Descripción
path	string	Ruta a la que cambiar.

Ejemplo:

```
this.context.router.replace("/ruta/a/la/vista");
```

this.context.router.go()

Mediante el método go(), podemos ir adelante o atrás en la historia, según la entrada en la que nos encontramos actualmente:

```
go(n)
```

Parámetro	Tipo de datos	Descripción
n	number	A qué entrada, a partir de la actual, saltar. Si es negativo, el salto será hacia atrás; si es positivo, adelante.

A continuación, mostramos unos ejemplos ilustrativos:

```
//atrás
this.context.router.go(-2);

//adelante
this.context.router.go(2);
```

Si deseamos ir una entrada atrás, podemos utilizar go(-1) o bien goBack():


```
goBack()
```

Si lo que necesitamos es ir una entrada adelante, go(1) o goForward():

```
goForward()
```

Actualización de propiedades del documento

Como sabemos, una vista viene a representar un documento HTML, el cual *no* se solicita al servidor, como ocurre en las aplicaciones tradicionales. En su lugar, el encaminador se encarga de reproducirla atendiendo al URL solicitado por el usuario.

Como las vistas representan documentos HTML, es muy común realizar algunas tareas específicas como cambiar el título y/o la descripción del documento.

Título del documento

El título del documento se indica mediante el elemento HTML <title>. En una aplicación React, lo fija la página index.html. Pero como cada vista representa un supuesto documento HTML, en muchas ocasiones, se desea fijar otro título distinto al fijado en index.html.

Esto es muy fácil de hacer. Cada vez que se cargue una vista, no habrá más que modificar la propiedad title del objeto document. Recordemos que este objeto es específico de los navegadores y es global. Generalmente, se hace en los métodos de ciclo de vida render() o componentDidMount(), usándose por ejemplo algún dato almacenado en los objetos de propiedades o de estado.

He aquí un ejemplo ilustrativo:

```
componentDidMount() {
  document.title = this.props.name;
}
```

Descripción del documento

Otro metadato muy importante, al menos desde el punto de vista SEO, es la descripción de la página. Cada vez que se carga una vista, es posible que nuestra aplicación deba actualizar tanto el título como su descripción. Recordemos que la descripción la utilizan los motores de búsqueda y, por esta razón, es muy importante su actualización con cada vista.

Se suele utilizar el mismo método que el utilizado para actualizar el título. Pero ahora, hay que fijar el elemento <meta name="description">. No hay más que acceder a él, por ejemplo, mediante el método querySelector() del objeto document y modificar su propiedad content.

Veámoslo mediante un ejemplo:

```
componentDidMount() {
  document.title = `${this.props.name} | mi.portal.com`;
  document.querySelector("meta[name=description]").content = this.props.desc;
}
```

