React Router - Parte 1

Tiempo estimado: 25min

Ésta es la primera parte de dos dedicadas al paquete React Router, encargado de facilitar la escritura de aplicaciones webs SPA con React.

Comenzamos la lección presentando la diferencia entre aplicación de página única (SPA) y multipágina (MPA). A continuación, presentamos el concepto de encaminador y cuál es su responsabilidad en una aplicación React. Después, exponemos qué es una ruta y cómo definirlas mediante el encaminador React Router. Finalmente, los comandos del generador de Justo que nos asisten con las rutas y las vistas.

Al finalizar la lección, el estudiante sabrá:

- Qué es React Router.
- Cómo desarrollar aplicaciones de múltiples vistas mediante React.
- Cuál es la función del encaminador en una aplicación React.
- Cómo asociar vistas y rutas en una aplicación React.
- Cómo definir rutas simples, compuestas y de índice mediante React Router.

Introducción

Una página web (*web page*) es un recurso accesible a través de Internet, por ejemplo, mediante un navegador o cualquier otro tipo de aplicación. Todo recurso web dispone de su propio URL que lo identifica de los demás y, así, permite su acceso. De HTML, sabemos que cada página web se desarrolla mediante su propio documento HTML. En las aplicaciones SPA, las cosas cambian un poco. Veámoslo detenidamente.

Atendiendo a cómo se implementa la aplicación web, básicamente las podemos clasificar en aplicaciones de página única o multipágina.

Una aplicación de página única (SPA, *single-page application*) es aquella que aglutina todas sus páginas en un único documento con el objeto de mejorar la experiencia del usuario. Este tipo de aplicaciones se desarrollan con HTML, CSS, JavaScript y algún *framework* específico que facilita el desarrollo de este tipo de aplicaciones como, por ejemplo, Angular, Ember o React. Por su parte, una aplicación multipágina (MPA, *multipage application*) es aquella que separa cada página en un recurso aparte. Es la manera tradicional de escribir aplicaciones webs clientes.

Independientemente del tipo de aplicación, en las páginas webs se puede distinguir principalmente dos aspectos: la presentación y los datos. La presentación (*display*) es el código que contiene la interfaz de usuario, o sea, lo que ve el usuario y con lo que interactúa. En cambio, los datos (*data*) es la información que visualiza la capa de presentación.

En una aplicación MPA, cuando el usuario accede a una página web, el servidor remite tanto la presentación como los datos. Si el usuario accede a la misma página varias veces, para cada una de ellas el servidor remite tanto la presentación como los datos. Para mejorar el rendimiento, algunas aplicaciones utilizan Ajax para permitir la visualización de distintos datos. Con Ajax, se mantiene la parte de presentación, solicitándose sólo los nuevos datos, lo que mejora el rendimiento, al no tener que solicitarse de nuevo la presentación. El servidor tiene que procesar y transmitir menos información por lo

que puede atender más solicitudes por segundo. El problema es que si el usuario cambia de página, el navegador solicita tanto la página como los datos de la nueva página. Y si a continuación el usuario vuelve a una página ya visitada, el servidor tendrá que remitirle *de nuevo* su presentación y sus datos.

La idea que se esconde en las aplicaciones SPA es muy sencilla: reducir al máximo el ir y venir de la capa de presentación y reducir las solicitudes al servidor web sólo a los datos. Para ello, la aplicación se desarrolla de tal manera que la interfaz de usuario, esto es, la presentación, se remite una única vez. Y a medida que el usuario interactúa con la aplicación cambiará de una página a otra sin necesidad de volver a solicitar la presentación, pues ya la tiene.

Las aplicaciones SPA siguen teniendo varias páginas, al igual que las MPA, pero todas ellas se transmiten una única vez al cliente. Esto mejora enormemente la aplicación. Habrá una carga inicial más duradera que si sólo se transmitiera una única página, pero una vez cargada la aplicación, ésta irá más rápida pues los mensajes que intercambiará con el servidor son únicamente de datos.

Por desgracia, las aplicaciones SPA son más difíciles de desarrollar que las MPA, pero el rendimiento adicional que aportan es tan importante que cada vez son más populares. Para facilitar y mejorar el desarrollo de aplicaciones de página única, se utiliza *frameworks* específicos como es el caso de React, el objeto del presente curso. Con React, se oculta parte de la complejidad que de otra manera el desarrollador debería implementar. Concretamente, este soporte se proporciona mediante el encaminador, en nuestro caso, el paquete react-router.

Las aplicaciones SPA desarrolladas con React contienen en el archivo empaquetado o *bundle* toda la aplicación, salvo la página de inicio, index.html, las hojas de estilo, las imágenes, etc. La página de inicio, recordemos, es la encargada de importar el archivo empaquetado de la aplicación React.

Es importante tener claro que aunque sólo se remite un documento, la aplicación dispone de varios URLs. Cada uno de ellos representa o identifica de manera única una única página web. La diferencia con las múltiples, es que en una aplicación SPA, es la propia aplicación la que se encarga de presentar la página en cuestión cuando el usuario acceda a ella. Como la aplicación ya tiene, en el *bundle*, el contenido de la página, debe monitorizar los cambios y, cuando el usuario haga clic en un enlace a una nueva página, internamente debe procesarlo y presentar la nueva página al usuario. Esto se consigue gracias al encaminador.

Generalmente, en una aplicación SPA no se habla de página web, sino de vista. Donde una vista (*view*), también conocida como vista de página (*page view*), no es más que una página virtual o una representación dinámica de una página. Tiene su propia ruta como, por ejemplo, /usuarios/, /usuario/1234, /articulo/quien-esta-usando-node, etc. En React Router, las vistas se implementan mediante componentes compuestos y, por convenio y buenas prácticas, se ubican en su propia carpeta /app/views/.

Para gestionar qué vista debe presentar la aplicación al usuario, se utiliza el encaminador (*router*), un elemento que presenta una vista u otra atendiendo al URL actual. Para ello, monitoriza los cambios de URL que se producen cuando el usuario navega a lo largo y ancho de la aplicación y se encarga de determinar y mostrar la vista que corresponde a cada uno de ellos.

La idea del encaminador es mantener sincronizado lo que ve el usuario, la vista, con lo que desea ver, la ruta. Cada vez que el usuario cambia de URL es porque desea se le muestre otra vista o página. Sin este soporte, la aplicación tendría una única vista que presentar al usuario y una única ruta. Gracias a él, la aplicación dispone de varias rutas y vistas.

Existe varias implementaciones de encaminadores para React, siendo React Router el recomendado por el equipo de React. Su sitio web oficial es github.com/reactjs/react-router.

Instalación de React Router

React Router se encuentra implementado en el paquete react-router, el cual se instala mediante NPM. Se trata de una dependencia de producción, por lo que habrá que indicarlo en la propiedad dependencies del archivo package.json:

```
"dependencies": {
  "react": "*",
  "react-dom": "*",
  "react-router": "*"
}
```

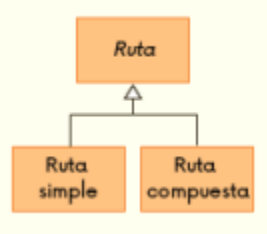

Encaminador en el navegador

El encaminador crea la propiedad ReactRouter en el objeto window para que podamos acceder directamente a él.

Rutas

Un registro de ruta (*route*) es un objeto que empareja URLs y vistas. Cada vez que el usuario acceda al URL indicado, el encaminador lo detectará y entonces mostrará la vista asociada. Simulándose así el concepto de página web.

Una ruta no es más que un identificador de recurso de la aplicación. Atendiendo a si pueden anidarse, se clasifican en simples y compuestas.



Una ruta compuesta (*composite route*) es aquella que actúa a modo de espacio de nombres de otras. Una especie de contenedor o recipiente de otras rutas. Tal como hacen los directorios. En cambio, una ruta simple (*simple route*) es aquella que es indivisible y de la que no cuelga ninguna otra ruta. De manera similar a los archivos.

Rutas compuestas

Una ruta compuesta (*composite route*), también conocida formalmente como ruta anidada (*nested route*), no es más que un recipiente o contenedor de rutas. Tiene los siguientes elementos:

- El punto de montaje (*mount point*) o URL del que cuelga su contenido como, por ejemplo, /usuarios/ o /noticias/.

  Este URL lo contendrá toda ruta que cuelgue de ella.

- La vista envoltorio (*layout view*) que debe mostrar el encaminador cuando el usuario solicite el URL de la ruta o de una de sus subrutas. Define el *look & feel* de las vistas, disponiendo de huecos donde se reproducirán las vistas de las subrutas solicitadas por el usuario.

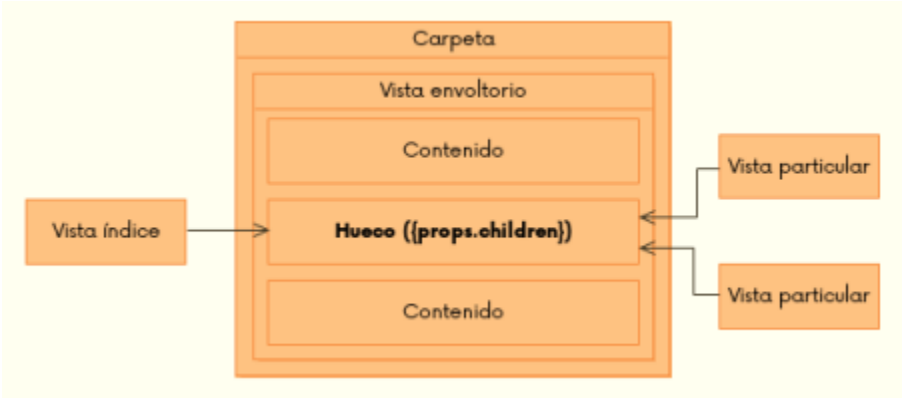
  Las vistas se implementan como componentes React. Pero por convenio y buenas prácticas, las vistas se encuentran en una carpeta aparte del resto, app/views. Por lo que esta propiedad referenciará a un componente de esta carpeta.

- La vista índice (*index route*) que el encaminador debe mostrar si el usuario accede directamente a la ruta compuesta, sin acceder a ninguna de sus subrutas.

  Referenciará a un componente React de la carpeta app/views. Generalmente, este tipo de vistas se definen con el sufijo Index.

- Las rutas hijas (*child routes*), aquellas que forman parte de la ruta compuesta. Tienen como parte inicial la ruta actual. O sea, aquellas cuya ruta es relativa a esta ruta compuesta.

Una ruta compuesta muestra siempre su vista envoltorio, la cual contiene un hueco donde se mostrará la vista particular que solicite el usuario, ya sea la vista índice del contenedor u otra particular.



El contenedor tiene un carácter organizativo y de composición de interfaz de usuario. Cada vez que se accede a la ruta o a una subruta del contenedor, el encaminador mostrará su vista envoltorio. Siempre. Es esta vista la que contiene un hueco en la que se depositará la vista particular que está solicitando el usuario.

La vista envoltorio se implementa mediante un componente compuesto, su método de representación es muy parecido a:

```
render() {
  return (
    <div>
      Contenido común para todas las vistas del contenedor.
      {this.props.children}  //hueco para la vista actual
      Más contenido común.
    </div>
  );
}
```

El encaminador mostrará siempre la vista envoltorio de la ruta compuesta a la que pertenezca el URL solicitado por el usuario. Dentro de ella, se encuentra un hueco (*placeholder*), un espacio en el que desplegar la vista particular solicitada por el usuario. Este hueco se representa mediante la expresión {this.props.children}.

Como se puede observar, la vista envoltorio contiene el código común de todas sus vistas hijas, además del contenedor donde se deben desplegar o reproducir. Es el sitio ideal para definir, por ejemplo, un menú de navegación común a la carpeta y, por ende, a todas sus vistas hijas.

Rutas simples

Por su parte, una ruta simple (*simple route*) es aquella que se asocia a un recurso indivisible, del que no cuelga ningún otro adicional. Este tipo de rutas tiene los siguientes elementos:

- Una ruta (*path*) o URL que la identifica de manera única de las demás como, por ejemplo, /usuarios/:id o /noticias/:id.

  Es relativa a la ruta compuesta donde se define.

- La vista (*view*) que debe mostrar el encaminador, en el hueco de la vista envoltorio, cuando el usuario solicite su URL.

  Será una de las indicadas en la carpeta app/views.

Llegado este punto, veamos un ejemplo para ir abriendo boca, antes de adentrarnos en cómo definirlas mediante React Router:

```
<Route path="usuario" component={require("../views/users/Layout").default}>
  <IndexRoute component={require("../views/users/Index").default} />
  <Route path="__nuevo__" component={require("../views/users/New").default} />
  <Route path=":id" component={require("../views/users/Edit").default} />
</Route>
```

Observe los componentes React a través de los cuales definir las rutas: Route e IndexRoute. Mediante un componente Route compuesto, se define una ruta compuesta. Mediante un componente Route simple, una simple. El atributo path indica la ruta de la aplicación asociada a la ruta. Así, por ejemplo, la ruta compuesta es /usuario. Y la de sus rutas hijas: /usuario/, /usuario/__nuevo__ y /usuario/:id. Las vistas

asociadas a cada ruta se indican mediante el atributo component. La ruta índice de una compuesta se define mediante un componente <IndexRoute>.

En breve, haremos más hincapié en estos componentes. Primero, terminemos con la parte más teórica y conceptual.

Mapa de rutas

Toda aplicación tiene un mapa de rutas (*route map*), un archivo JSX que contiene la definición de todos los registros de ruta de la aplicación. Este mapa no es más que el componente Route que indica el componente raíz o principal de la aplicación, así como sus rutas hijas. Veamos un ejemplo ilustrativo:

```
//imports
import React from "react";
import {Route, IndexRoute} from "react-router";

//route map
export const map =
<Route path="/" component={require("../views/App").default}>
  <IndexRoute component={require("../views/AppIndex").default} />
  <Route path="usuario" component={require("../views/users/Envelope").default}>
    <IndexRoute component={require("../views/users/Index").default} />
    <Route path="__nuevo__" component={require("../views/users/New").default} />
    <Route path=":id" component={require("../views/users/Edit").default} />
  </Route>
</Route>;
```

Por convenio, el mapa de rutas se ubica en el archivo app/routes/map.jsx.

Archivos de rutas

Para facilitar el mantenimiento del mapa de rutas de la aplicación, se suele crear un archivo específico para cada carpeta que representa un elemento reutilizable de la aplicación. En lugar de hacer que el mapa de rutas contenga todas las rutas, se le delega sólo la de la ruta raíz y cada componente se asocia a un archivo. Mejor mediante un ejemplo:

```
//imports
import React from "react";
import {Route, IndexRoute} from "react-router";

//route map
export const map =
<Route path="/" component={require("../views/App").default}>
  <IndexRoute component={require("../views/AppIndex").default} />
  {require("./user").default}
  {require("./news").default}
</Route>;
```

Cada uno de los elementos reutilizables tendrá su propio archivo de rutas. Por ejemplo, el de usuarios podría ser:

```
//archivo: app/routes/user.jsx
//imports
import React from "react";
import {Route, IndexRoute} from "react-router";

//route map
export const map =
<Route path="usuario" component={require("../views/users/Envelope").default}>
  <IndexRoute component={require("../views/users/Index").default} />
  <Route path="__nuevo__" component={require("../views/users/New").default} />
  <Route path=":id" component={require("../views/users/Edit").default} />
</Route>;
```

De esta manera, el mantenimiento y la reutilización es más sencilla. Cada vez que añadamos un elemento reutilizable, bastará con añadir su archivo de rutas a la carpeta /app/routes y su entrada a /app/routes/map.jsx.

Componente raíz de la aplicación

Recordemos que el componente raíz o principal es aquel que usa React como punto de partida o inicio de la aplicación. Se indica mediante el método render() del paquete react-dom. Recordémoslo también:

```
ReactDOM.render(<App />, document.getElementById("react-app"));
```

Cuando se utiliza el encaminador React Router para desarrollar aplicaciones SPA con distintas vistas, lo más habitual, el componente raíz debe ser necesariamente una instancia de un componente Router. He aquí un ejemplo ilustrativo:

```
//imports
import React from "react";
import ReactDOM from "react-dom";
import {Router, browserHistory as history} from "react-router";
import {map} from "./routes/map";

//root component
ReactDOM.render(
  <Router history={history}>{map}</Router>,
  document.getElementById("react-app")
);
```

Componente Router

El componente Router, ubicado en el paquete react-router, crea el encaminador y actúa como sincronizador de URLs y vistas. Inicialmente, mostrará la ruta raíz de la aplicación, o sea, la vista asociada a la ruta /. Consiste en un componente compuesto, donde las rutas asociadas se indican como su contenido. Observemos, del ejemplo anterior, que lo que hay que indicarle es el objeto devuelto por el archivo de mapa de rutas, /app/routes/map.jsx.

Componente Route

Mediante el componente Route del paquete react-router, se asocia rutas y vistas. Se puede usar para definir rutas simples y compuestas. Cuando se trata de una ruta compuesta, se usa como un componente compuesto. Mientras que para rutas simples, como uno simple.

Atributo path

Mediante el atributo path se indica el *path* de la ruta. Siempre es relativo a su ruta madre. Por ejemplo, la ruta raíz de la aplicación es / y se indica mediante path="/". En cambio, la de la carpeta /noticia como path="noticia", o sea, como relativa a su contenedor /.

Atributo component

El atributo component se usa para indicar el componente vista asociado a la ruta. En el caso de una ruta compuesta, su vista de envoltorio. Para una ruta simple, su vista asociada.

Recordemos que por convenio y buenas prácticas, los componentes que definen vistas se ubican en la carpeta app/views, en vez de hacerlo en app/components.

Sección de contenido

Si estamos ante una ruta compuesta, sus rutas hijas se definirán en su sección de contenido. Ejemplo:

```
<Route path="usuario" component={require("../views/users/Envelope").default}>
  <IndexRoute component={require("../views/users/Index").default} />
  <Route path="__nuevo__" component={require("../views/users/New").default} />
  <Route path=":id" component={require("../views/users/Edit").default} />
</Route>
```

Observe que las rutas hijas se definen relativas a su ruta madre. Por esta razón, la ruta usuario/__nuevo__ se define sólo como __nuevo__.

Component IndexRoute

Las rutas compuestas se comportan como carpetas. De manera similar a una carpeta, se puede acceder a la propia carpeta o a una de sus entradas. Pues a la vista que debe mostrar el encaminador cuando se acceda a la propia ruta compuesta, o sea, a la carpeta, se le conoce formalmente como ruta índice (*index route*).

Este tipo de rutas se indican mediante el componente IndexRoute del paquete react-router. A diferencia de Route, no presentan atributo path, porque su valor siempre es /, pero sí component. Ejemplo:

```
<IndexRoute component={require("../views/users/Index").default} />
```

Rutas dinámicas

Las rutas se pueden clasificar en estáticas o dinámicas.

Una ruta estática (*static route*) es aquella que tiene un *path* que identifica un único recurso posible. Mientras que una ruta dinámica (*dynamic route*), aquella que se puede asociar a varios *paths* mediante el uso de comodines de manera muy parecida a las expresiones regulares de JavaScript. React Router permite el uso de los siguientes comodines:

- *. Representa cero, uno o más caracteres.
- **. Representa cualquier número de caracteres hasta el siguiente carácter /, ? o #.
- (). Delimita una porción opcional.

He aquí unos ejemplos ilustrativos:

```
/proyectos/*
/proyectos/**/team
/proyectos(/:id)
```

Rutas parametrizadas

Una ruta parametrizada (*parameterized route*) es un tipo especial de ruta dinámica que puede contener parámetros. Cada parámetro (*parameter*) representa un fragmento variable del *path*, cuyo valor podemos acceder en la vista. Su nombre viene precedido por dos puntos (:). Una ruta puede tener tantos parámetros como sea necesario.

A continuación, se muestra algunos ejemplos ilustrativos de rutas parametrizadas:

```
/proyecto/:id
/proyecto/:idPro/team/:idTeam
```

El encaminador crea la propiedad params dentro del objeto props de la vista para que así pueda acceder a los valores de los parámetros. Cada parámetro tendrá una propiedad homónima dentro de params. Así pues, para acceder al valor del parámetro id de la ruta, tendremos que usar algo como this.props.params.id en el componente vista.

Generalmente, se usa los parámetros para conocer el identificador del objeto de datos al que desea acceder el usuario. Así, por ejemplo, si tenemos una vista Info que muestra la información de un determinado producto, podremos indicar cuál en el *path*. Ejemplos: /producto/bq-x5-plus o /producto/rapsberrypi3. Bajo esta sintaxis, la ruta sería algo como /producto/:id. Y así podríamos definir una única vista para todos los productos. La vista determinará cuál de ellos, mediante

this.props.params.id.

Generador de Justo

Para facilitarnos el trabajo con las rutas, el generador de Justo proporciona varios comandos.

Hay que decir que cuando generemos la estructura de directorios de la aplicación con el generador, para añadir soporte de encaminador al proyecto, hay que indicar que usaremos React Router. Es raro que, en el momento de crear la estructura de directorios del proyecto, no sepamos todavía si lo usaremos.

Comando route file

Mediante el comando route file, crearemos un archivo de rutas en /app/routes y lo registraremos en el archivo mapa, /app/routes/map.jsx.

Por lo general, se utiliza un archivo de ruta para cada elemento de la aplicación que, digamos, puede ser desarrollado de manera independiente. A su vez, a este elemento le asignamos su propio URL, dentro del cual se encontrarán todas las vistas que tiene asociadas. Por esta razón, el comando route file crea una vista envoltorio y una vista índice para la ruta compuesta que se define en el archivo de rutas.

Comando route

Mediante el comando route, añadiremos una ruta simple a un archivo encaminador. Antes de invocar el comando, asegúrese de que la vista ya existe en app/views. Se la solicitará.

Comando view

Mediante el comando view, se añade un componente vista de página a la carpeta /app/views.

