React: Rutas

Una aplicación de React está formada por muchos componentes. Muchos de ellos estarán anidados dentro de otros, pero muchos otros componentes estarán al mismo nivel. Hasta ahora, todos los componentes se renderizaban y eran visibles en todo momento, pero puede que nosotros queramos que no sea así: queremos que, en función de unas condiciones u otras, se muestren unos componentes u otros.

Es decir, si estoy en el componente App, quiero que se vean ciertos componentes (Cabecera, Footer), pero no quiero que se vean todos los barcos. Queremos que los barcos se vean únicamente cuando entremos a un componente concreto.

App.js

```
function Cabecera() {
  return (
    <header>
      <h1>Muelle de Naves</h1>
      <nav>
        <ul>
          <li>
            <a href="/inicio">Barcos</a>
          </li>
          <li>
            <a href="/contacto">Contacto</a>
          </li>
        </ul>
      </nav>
    </header>
  );
}
function Barcos() {
  return (
    <>
      <section>
        <div>
          <h1>Bar Quito</h1>
          <p>Eslora: 5m</p>
          <p>Tripulantes: 4</p>
```

```
          <p>Tipo: Recreativo</p>
        </div>
        <div>
          <h1>Imperioso</h1>
          <p>Eslora: 25m</p>
          <p>Tripulantes: 2</p>
          <p>Tipo: Recreativo</p>
        </div>
      </section>
    </>
  );
}
function Contacto() {
  return (
    <section>
      <p>
        Para contactar preguntar en la nave al lado de la entrada o llamar al
        94.372.28.23
      </p>
    </section>
  );
}
function Footer() {
  return (
    <footer>
      <p>&copy;2020</p>
    </footer>
  );
}
function App() {
  return (
    <>
      <Cabecera />
      <Barcos />
      <Contacto />
      <Footer />
    </>
  );
}
```

Podríamos añadir un enlace usando la etiqueta **<a>** para crear rutas para los diferentes componentes: con **Barcos** podríamos definir un enlace al componente Barcos. El problema con React es que tenemos un único html, por lo que si visitamos un enlace, la página se actualiza y todas las variables que habíamos definido (en el estado) vuelven al valor inicial. Lo que queremos conseguir con React es dar la sensación de navegación entre páginas, pero sin actualizar la página: únicamente cambiamos el HTML que se renderiza en cada momento.

Para conseguir eso, tendremos que instalar el paquete **react-router-dom** Es un paquete npm que nos dará la posibilidad de implementar rutas en el DOM. En nuestro componente principal, importamos el módulo:

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

En el return del componente **App**, ponemos etiquetas **<BrowserRouter> </BrowserRouter>** alrededor de todos los demás componentes que vamos a escribir. Es importante que esta etiqueta envuelva a todos los componentes sobre los que queremos navegar.

La etiqueta **<Route></Route>** la usaremos para envolver todo aquel(los) componente(s) que queremos mostrar según el valor de la ruta. Para ello, daremos un atributo **path**='/valorDeLaRuta' para decirle a React que, cuando vayamos a esa ruta, nos muestre el componente que está entre las etiquetas **<Route></Route>**. Importamos lo que necesitamos y modificamos el componente **App**

App.js

```
import React, {useState} from 'react';
import { BrowserRouter, Route } from 'react-router-dom';
...
function App() {
    let [visitantes, setVisitantes] = useState(0);

    function sumarVisitante() {
        setVisitantes(visitantes + 1);
    }
    return (
        <BrowserRouter>
        <Cabecera />
        <section>
          <p>Hemos tenido {visitantes} visitantes</p>
          <button onClick={sumarVisitante}>Sumar visitante</button>
        </section>
        <Routes>
```

```
          <Route path="/inicio" element={<Barcos />} />

          <Route path="/contacto" element={<Contacto />} />
        </Routes>

        <Footer />
      </BrowserRouter>
    );
}

export default App;
```

Cuando entremos a direccionDeLaPágina/inicio, React nos mostrará el componente Barcos, además de todo lo que tengamos fuera de etiquetas **<Router>**

Pero ahora, si hacemos click en un enlace **<a>**, la página se seguirá actualizando y los valores guardados en el estado volverán a sus valores por defecto. Para eso, en vez de usar etiquetas **<a>** usaremos la etiqueta **<Link>** que hemos importado del paquete **react-router-dom**. Lo que conseguiremos con esta etiqueta será escribir en la URL un valor, pero no actualizaremos la página. De este modo, con **<Link>** escribiremos un valor en la URL y **<Route>** nos pintará el componente correspondiente. Para indicar qué es lo que va a escribir en la URL, asignaremos el atributo **to=**'ruta' a la etiqueta **<Link>.** Modificamos el componente **Cabecera**

App.js
```
function Cabecera() {
    return (
      <header>
        <h1>Muelle de Naves</h1>
        <nav>
          <ul>
            <li>
              {/* <a href="/inicio">Barcos</a> */}
              <Link to="/inicio">Barcos</Link>
            </li>
            <li>
              {/* <a href="/contacto">Contacto</a> */}
              <Link to="/contacto">Contacto</Link>
            </li>
          </ul>
        </nav>
      </header>
    );
```

```
  }
```

Subir el estado

Vamos a modificar el componente Barcos. Crearemos una variable **barcos**, que será un array con los objetos barco dentro de él. En el return escribiremos un JSX en el que añadimos los valores de los objetos del array:

```
function Barcos() {
  const barcos = [
    {
      nombre: 'Bar Quito',
      eslora: 5,
      tripulantes: 4,
      tipo: 'recreativo',
    },
    {
      nombre: 'Imperioso',
      eslora: 25,
      tripulantes: 2,
      tipo: 'recreativo',
    },
  ];
  return (
    <>
      <section>
        <div>
          <h1>{barcos[0].nombre}</h1>
          <p>Eslora: {barcos[0].eslora}m</p>
          <p>Tripulantes: {barcos[0].tripulantes}</p>
          <p>Tipo: {barcos[0].tipo}</p>
        </div>
        <div>
          <h1>{barcos[1].nombre}</h1>
          <p>Eslora: {barcos[1].eslora}m</p>
          <p>Tripulantes: {barcos[1].tripulantes}</p>
          <p>Tipo: {barcos[1].tipo}</p>
```

```
        </div>
      </section>
    </>
  );
}
```

Podríamos reducir el código usando la función **.map**. Creamos una variable **barcosHTML** en la que guardaremos un array de elementos JSX con los datos de los objetos del array **barcos.** En el return del componente pondremos el array de elementos JSX:

```
function Barcos() {
    const barcos = [
        {
            nombre: 'Bar Quito',
            eslora: 5,
            tripulantes: 4,
            tipo: 'recreativo',
        },
        {
            nombre: 'Imperioso',
            eslora: 25,
            tripulantes: 2,
            tipo: 'recreativo',
        },
    ];
    const barcosHTML = barcos.map(function (barco) {
        return (
            <div>
                <h1>{barco.nombre}</h1>
                <p>Eslora: {barco.eslora}m</p>
                <p>Tripulantes: {barco.tripulantes}</p>
                <p>Tipo: {barco.tipo}</p>
            </div>
        );
    });
    return (
        <>
            <section>{barcosHTML}</section>
        </>
    );
}
```

Podríamos crear un componente **Barco** que reciba **props** y que pinte esa información con la misma estructura que tenemos arriba:

```
function Barco(props){

    return (
        <div>
        <h1>{props.nombre}</h1>
        <p>Eslora: {props.eslora}m</p>
        <p>Tripulantes: {props.tripulantes}</p>
        <p>Tipo: {props.tipo}</p>
    </div>
    )
}
```

De este modo, en el componente **Barcos**, en la función map, en vez de escribir esa estructura de JSX, pondremos un componente **Barco** al que le pasamos la información de cada barco:

```
function Barcos() {
    const barcos = [
        {
            nombre: 'Bar Quito',
            eslora: 5,
            tripulantes: 4,
            tipo: 'recreativo',
        },
        {
            nombre: 'Imperioso',
            eslora: 25,
            tripulantes: 2,
            tipo: 'recreativo',
        },
    ];
    const barcosHTML = barcos.map(function (barco) {
        return (
                            <Barco  nombre={barco.nombre}  eslora={barco.eslora}
tripulantes={barco.tripulantes} tipo={barco.eslora} />
        );
    });
    return (
```

```
        <>
            <section>{barcosHTML}</section>
        </>
    );
}
```

Y si queremos reducir aún más, podríamos usar una función flecha junto con el **.map** para mostrar los componentes **Barco** directamente en el **return** del componente **Barcos:**

```
function Barcos() {
    const barcos = [
        {
            nombre: 'Bar Quito',
            eslora: 5,
            tripulantes: 4,np
            tipo: 'recreativo',
        },
        {
            nombre: 'Imperioso',
            eslora: 25,
            tripulantes: 2,
            tipo: 'recreativo',
        },
    ];

    return (
        <>
            <section>
                {barcos.map((barco) =>
                    <Barco
                    nombre={barco.nombre}
                    eslora={barco.eslora}
                    tripulantes={barco.tripulantes}
                    tipo={barco.eslora} />
                )}
            </section>
        </>
    );
}
```

Hemos visto que en esta aplicación tenemos un contador de visitas y un botón que incrementa el contador, todo ello en el componente **App**. Queremos ahora que ese botón solo se muestre cuando entramos a ver los barcos, es decir, en el componente **Barcos**. Pero también queremos que el contador sea visible desde cualquier parte de la aplicación, por lo que la variable de estado que controla el contador la tendremos en el componente **App**, pero el botón en el componente hijo.

Para ello, en el componente **App**, pasamos al componente hijo la variable de estado y la función que queremos que se ejecute cuando hagamos click en el botón:

```
function App()
<Barcos                barcos={barcos}              visitantes={visitantes}
sumarVisitante={sumarVisitante} />
```

En el componente **Barcos**

Para ello, en el componente **Barcos** creamos un botón y le decimos que cuando hagamos click en él, ejecute la función **sumarVisitante** que recibe por props. Cuando hagamos click ejecutaremos la función que hemos definido en el componente padre **App** y ésta incrementará en 1 el contador de visitantes.

```
function Barcos(props) {
    return (
        <>
            <section>
             <button onClick={props.sumarVisitante}>Sumar visitantes</button>
            </section>
            <section>
                {props.barcos.map((barco) =>
                    <Barco
                        nombre={barco.nombre}
                        eslora={barco.eslora}
                        tripulantes={barco.tripulantes}
                        tipo={barco.eslora} />
                )}
            </section>
        </>
    );
}
```

Podríamos pasar aún más hacia abajo la función de sumar 1 al contador de visitantes: podríamos hacer que cada componente **Barco** tenga un botón que ejecute la función que tenemos en el componente principal y que sume 1 al contador total y al contador propio de cada barco.

Para conseguirlo, en cada componente **Barco** tendremos una variable de estado y un botón que ejecute una función cuando se haga click. Esa función actualizará el estado del propio **Barco** y también el estado del componente **App**, ejecutando la función que recibe por props del componente **Barcos** que, a su vez, ha recibido del componente **App**:

```
function Barco(props) {
  let [guests, setGuests] = useState(0);

  function sumarGuest(){
    setGuests(guests+1);
    props.sumarVisitante();
  }

    return (
        <div>
            <h1>{props.nombre}</h1>
            <p>Eslora: {props.eslora}m</p>
            <p>Tripulantes: {props.tripulantes}</p>
            <p>Tipo: {props.tipo}</p>
            <p>Visitantes: {guests}</p>
            <button onClick={sumarGuest}>Sumar visitante</button>
        </div>
    )
}
```

Resumiendo, tenemos un estado definido en un componente y queremos que ese estado pueda ser modificado desde sus componentes hijos. Por eso, en el componente padre definimos la función que modificará el estado. A los componentes hijo les pasamos esa función dentro del objeto **props**. Si el botón que modificará el estado del componente padre está muy abajo en la estructura de componentes padre/hijo, tendremos que pasar esa función de unos a otros hasta llegar al componente donde queremos utilizarlo.

Una vez en el componente indicado, no será necesario crear una función concreta para cuando hagamos click (o cualquier otro evento). Simplemente ejecutaremos la función que nos llega en **props**, que será la que hemos definido en el componente padre.