ThePower business school

🇪🇸 React Router ReactJS

Introducción

Para navegar entre nuestras páginas, React nos permite usar otras librerías, pero la que más se utiliza es react-router. Debes tener en cuenta que en este post se explicará la versión **v6**, que corresponde a la última versión en el momento que escribimos este post.

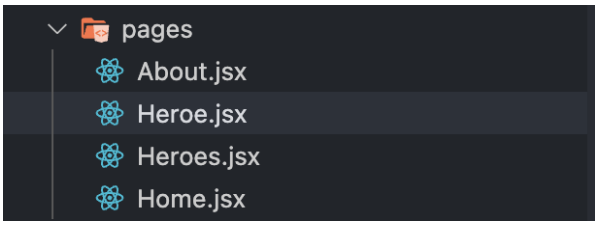
React Router

Para añadir navegación a nuestros proyectos de React es tan sencillo como ejecutar el siguiente comando sobre la terminal.

```
npm create vite@latest
cd react-router
npm install
npm install react-router-dom@6
```

Páginas

Para este proyecto vamos a crear 4 componentes página que contengan el contenido de nuestra aplicación. En cada uno de ellos trabajaremos más adelante.



Definición del mapa de rutas

En nuestro fichero `main.jsx` tenemos que definir el componente `<BrowserRouter>` , que contiene todos los componentes que forman el mapa de todas las rutas que vamos a ir habilitando en nuestra aplicación.

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import App from './App'
import './index.css'

// Import Pages
import Home from './pages/Home'
import About from './pages/About'
import Heroes from './pages/Heroes'
import Heroe from './pages/Heroe'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter basename="/">
      <Routes>
        <Route path="/" element={<App />}>
          <Route index element={<Home />} />
          <Route path="heroes" element={<Heroes />} />
          <Route path='/heroe/:id' element={<Heroe />} />
          <Route path="about" element={<About />} />
          <Route
            path="*"
            element={
              <main>
                <p>404 - No existe la ruta!</p>
              </main>
            }
          ></Route>
        </Route>
      </Routes>
    </BrowserRouter>
  </React.StrictMode>
)
```

Las rutas “hijas” con `index element` son rutas sin path, que se renderizarán sobre el componente “Outlet” del elemento padre al acceder a su URL (la del padre), este componente lo veremos más adelante. En este caso, el componente `Home` se renderizará en el path vacío sobre el `Outlet` del componente `App`.

Nuestros componentes `<Routes>` y `<Route>` se utilizan para renderizar un element dependiendo del location actual en la URL. Realmente podemos ver un `<Route>` como un `if` o un `switch` en el que si el path coincide con la URL actual renderizará el element.

Route	Page - Component
/	Home
/heroes	Heroes
/herores/:heroesid	Heroe
/about	About
/*	No existe la ruta!

Definición del Layout

Podemos usar cualquier componente propio pero también el componente de App, en este ejemplo lo vamos a usar a modo de Layout. Para ello vamos a dividir nuestro componente en tres partes. Encabezado → `header` || Menú de navegación → `nav` || Contenido → `content`

```
import { NavLink, Outlet } from 'react-router-dom'
import './App.css'

function App() {

  return (
    <div className="App">
      <header className="header">
        <h1>React Router v6 🧪</h1>
      </header>
      <div>
        <nav>
          <NavLink to="">Home</NavLink>
          <NavLink to="heroes">Heroes</NavLink>
          <NavLink to="about">About</NavLink>
        </nav>
        <main>
          <Outlet />
        </main>
      </div>
    </div>
  )
}
```

```
export default App
```

De este modo ya tenemos Navegación dentro de nuestro proyecto de React.

Componente Link

Podemos decir que nuestro componente `<Link>` es un elemento que permite al usuario navegar a otra parte de la App, similar a la etiqueta `<a>` en HTML.

1. **Componente NavLink.**

   Un `<NavLink>` es una especie de `<Link>` que sabe si la ruta que contiene es la activa en ese momento. Se comporta de una forma muy parecida al `Link` con la diferencia de que propaga la clase `active` al elemento `a` que se corresponda con el Link cuyo path está activo. Por ejemplo, en la ruta `/heroes` el elemento `<NavLink to="/heroes" />` tendrá la clase `.active`.

2. **Componente Outlet.**

   El componente Outlet lo usamos dentro de del del componente utilizado en la ruta padre como `Route`, de este modo nos permite renderizar sus `<Route>` hijos. Esto permite a la interfaz anidada mostrar las rutas hijas cuando son renderizadas. Si la ruta seleccionada es la raíz, se renderizará la `<Route index>` hija. Si la ruta no está mapeada, se renderizará la `<Route path='*'>` hija.

Definiendo components de página

En Home usaremos el `Link` para ir a nuestro Heroes. Y así poder nevegar hasta nuestros Heroes.

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

function Home() {
  return (
    <>
      <h2>Home Page</h2>

      <p>App ejemplo sobre React Router</p>

      <ul>
        <li>
          <p>
            <span>Visita la página de héroes 🦸:</span>
            <Link to="heroes">Heroes</Link>
          </p>
        </li>
      </ul>
    </>
  );
}

export default Home;
```

El About podemos simplemente marcar un mensaje sencillo.

```
import React from 'react'

const About = () => {
  return (
    <>
      <h2>Sobre esta app...📍</h2>
      <p>Es un ejemplo para aprender el funcionamiento de React Router</p>
      <p>Usa la barra de navegación de arriba para cambiar de rutas! 🔝</p>
    </>
  )
}

export default About
```

Definiendo nuestra Data

Como queremos simular que los datos vengan de una API. Vamos a simular estas llamadas con unos **mocks** a través de un fichero `data/data.js` que nos permita recoger los héroes, recoger un héroe por id y eliminar héroes por id.

```
let heroes = [
  {
    id: 1,
    name: "Superman",
    age: 45,
    alias: "Clark Kent"
  },
  {
    id: 2,
    name: "Batman",
    age: 55,
    alias: "Bruce Wayne"
  },
  {
    id: 3,
    name: "Wonder Woman",
    age: 1555,
    alias: "Diana"
  },
  {
    id: 4,
    name: "Green Latern",
    age: 31,
    alias: "Jal Jordan"
  },
  {
    id: 5,
    name: "Aquaman",
    age: 42,
    alias: "Arthur Curry"
  }
];

export const getHeroes = () => heroes;


export const getHeroe = id => heroes.find(
  heroe => heroe.id.toString() === id
);

export const deleteHeroe = async (id) => heroes = heroes.filter(
```

```
  heroe => heroe.id !== id
);
```

Definiendo componente Heroe Detail → Listado y Detalle

Vamos a definir un componente sencillo que recibe por props un heroe y retorna los valores que tenemos que pintar.

```
import React from 'react'

const HeroeDetail = ({ heroe }) => {
  return (
    <>
      <h1>name: {heroe.name}</h1>
      <p>alias: {heroe.alias}</p>
      <p>age: {heroe.age}</p>
    </>
  )
}

export default HeroeDetail
```

Definición de Rutas dinámicas

Dado que tenemos nuestros supers vamos a renderizarlos en nuestra página heroes pero teniendo en cuenta que cada super debe ser clickable y cuando lo hagamos vayamos al detalle de este en la página heroe recogiendo un id de los params de la url.

Es decir, en nuestra aplicación vamos a poder navegar y mostrar los datos de cada super. Dicha navegación se va construir dinámicamente a partir de los supers existentes en nuestro `data/data.js`.

```
import React from 'react'
import { Link, Outlet } from 'react-router-dom'
import HeroeDetail from '../components/HeroeDetail'
import { getHeroes } from '../data/data'

const Heroes = () => {
```

```
  const heroes = getHeroes();
  return (
    <>

      <div>
        <h1>All heroes 🦸🦸</h1>
        <ul>
          {heroes.map((heroe) => (
            <li key={heroe.id}>
              <Link to={`/heroes/${heroe.id}`}>
                <HeroeDetail heroe={heroe} />
              </Link>
            </li>
          ))}
        </ul>
      </div>
      <Outlet />
    </>
  )
}

export default Heroes
```

Te dejamos aquí el componente `Heroe.jsx` .

```
import { useParams, useNavigate } from 'react-router-dom'
import HeroeDetail from '../components/HeroeDetail';
import { getHeroe, deleteHeroe } from '../data/data';

export default function Heroe() {
  const params = useParams();
  const navigate = useNavigate();
  const heroe = getHeroe(params.id);

  if (!heroe) return <p>No existe el héroe que buscas 😭</p>;

  return (
    <div>
      <h1>My heroes 🦸🦸</h1>
      <HeroeDetail heroe={heroe} />
      <button
        onClick={() => {
          deleteHeroe(heroe.id).then(() => {
            navigate('/heroes');
          });
        }}
      >
        Borrar a {heroe.name}
      </button>
```

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

Mini-ejercicio

Ha llegado el momento de ponerse a trabajar con ReactJS, para ello os proponemos una pequeña práctica que os ayude afianzar el uso de la librería react-router.

1. Crea una aplicación de ReactJS con vite → name: project-basic-router.
2. Crea tu carpeta de `components` dentro de `src`.
3. Realizamos algunos componentes de ReactJS:
    a. Componente Header ⇒ Crea un componente que reciba como children el componente Title y retorne un `<header> + Children`.
    b. Componente Main ⇒ Crea un componente que reciba como children los algunos componentes creados y retorne un `<main> + Children`.
    c. Componente Footer ⇒ Crea un componente que reciba como children los algunos componentes creados y retorne un `<footer> + Children`.
    d. Crea los componentes necesarios para representar la información en tus diferentes páginas.
        i. Componentes de páginas mínimos ⇒ Home || Listado || About
        ii. Componentes mínimos ⇒ Navbar || Header || Navbar || Main || Footer
4. Estila cada uno de ellos haciendo uso de CSS Modules → hoja de estilo asociada al componente.
5. Exporta los componentes en un `index.js` e importalos en `App.jsx`.
6. Comprueba que la visualización es correcta.