React Router

"Client side routing"

KEEPCODING

Tech School

Principios básicos

- [React router docs]
- **Server side routing:**
  - Browser solicita una ruta al servidor, descarga una respuesta HTML con JS y CSS y renderiza esa respuesta
  - Cuando el usuario hace *click* en un enlace, el proceso empieza de nuevo
- **Client side routing:**
  - Cuando el usuario hace *click* en un enlace, la URL del browser se actualiza sin hacer una nueva petición al servidor
  - Un componente descargado previamente en el JS, el **router**, se encarga de *sincronizar* la interfaz mostrada con la URL del browser, es decir, **mostrar unos componentes React u otros, en función de la URL**
  - React Router nos ofrece una serie de utilidades y componentes React que se encargarán de este *client side routing*

Router: createBrowserRouter y RouterProvider

- Router recomendado para la mayoría de aplicaciones web
- Usa DOM History API para actualizar la URL y manejar el objeto history
- Usado conjuntamente con **RouteProvider** al que le pasaremos el **router**

```
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
// creamos el router que maneja la navegación en nuestra app
const router = createBrowserRouter([{
  path: '*',
  element: <App />,
}]);
// pasamos el router al router provider
ReactDOM.createRoot(document.getElementById("root")).render(
  <RouterProvider router={router} />
);
```

🔔 **Similar a <BrowserRouter />, pero con control del router usado**

Componentes principales

- **Route:** Componente que asocia una ruta con un elemento React
- **Routes:** Agrupación de componentes **Route.** Decide qué ruta dentro de la agrupación ofrece el mejor **match**
- **Outlet:** En rutas anidadas, es usado en la ruta padre para renderizar la ruta hija en su lugar. Una especie de *hueco* que el padre deja para ser sustituido por el resultado de la ruta hija
- **Navigate:** Componente que cuando es renderizado cambia la URL. Es como una redirección declarativa
- **Link:** Enlace que permite la navegación a otra URL
- **NavLink:** Como un Link, pero este componente “sabe” cual es la ruta activa y puede aplicar estilos basados en ello (útil para un menú)

Route

Route es la parte “principal” de React Router
Se encarga de renderizar una porción de UI cuando la URL coincide con la propiedad **path**, o null en caso contrario

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

// cuando la ruta sea /login se renderiza el componente <Login />
<Route element={<Login />} path="/login" />

// permite parámetros dinámicos en la url, teamId podrá ser leído por <Team />
<Route element={<Team />} path="/teams/:teamId" />

// permite definir una ruta index dentro de un grupo
<Route element={<Teams />} index />
```

Routes

Routes se encarga de renderizar el Route que tiene mejor **match** de entre todos los hijos

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

<Routes>
 <Route path="/" element={<Dashboard />}>
   <Route
     path="messages"
     element={<DashboardMessages />}
   />
   <Route path="tasks" element={<DashboardTasks />} />
 </Route>
 <Route path="about" element={<AboutPage />} />
</Routes>
```


Rutas anidadas: Outlet

React router permite **anidar** unas rutas dentro de otras
Cuando anidamos rutas, el componente padre debe dejar un “hueco” donde se renderiza la ruta hija elegida. Ese hueco es **<Outlet />**

```
import { Routes, Route, Outlet } from 'react-router-dom';
// <Dashboard /> es el componente que renderiza la ruta padre
<Route path="/" element={<Dashboard />}>
  <Route path="messages" element={<DashboardMessages />} />
  <Route path="tasks" element={<DashboardTasks />} />
</Route>

// <Dashboard /> debe dejar un <Outlet /> donde se renderiza la ruta hija
// el componente de la ruta hija “vencedora” sustituye a Outlet
// dashboard.js
<div><h1>Dashboard</h1><Outlet /></div>
```

Redirecciones: Navigate

El componente **<Navigate />** permite realizar redirecciones declarativas
Al renderizarse provoca una redirección a una nueva URL

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

// Si se renderiza Navigate, se producirá una redirección a /login
<Navigate to="/login" />

// con replace sustituimos la última entrada del histórico, en lugar de crear una
<Navigate to="/login" replace />

// Podemos guardar es state cualquier dato interesante para la redirección
// Ese estado podrá ser leído posteriormente en la ruta /login
<Navigate to="/login" state={...} />
```

Enlaces: Link

Componente para poner un enlace a otra página

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

// Enlace a /login
<Link to="/login" >Login<Link />

// reemplazando el histórico
<Link to="/login" replace />Login<Link />

// Guardando información en el state que puede ser leída en la ruta destino
<Link to="/login" state={...} />Login<Link />
```

🔔 **No usar elementos <a />, provocan navegación al server**

Enlaces: NavLink

Similar a un Link, pero NavLink sabe si la ruta del enlace es la **activa**
Esto permite modificar el estilo del enlace basado en la ruta activa

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

// El enlace recibe la clase “active” cuando la ruta sea /messages
<NavLink to="/messages" className={({isActive}) => (isActive ? 'active' : '')}>
  Messages
</NavLink>;

// Igualmente podemos asignar un style inline
<NavLink
  to="/messages"
  style={({isActive}) => ({color: isActive ? 'red' : 'black'})}>
  Messages
</NavLink>;
```

Hooks

- **useParams:** Devuelve un objeto con los **parámetros dinámicos**

```
const { teamId } = useParams();
```

- **useLocation:** Devuelve el objeto **location** actual

```
const location = useLocation();
```

- **useNavigate:** Devuelve una función con la que podremos realizar redirecciones imperativas (por código)

```
const navigate = useNavigate();
navigate('/login', options); // <Navigate /> options
```

- **useSearchParams:** Hook para leer y modificar la query string de la URL

```
// se usa de forma similar a useState
const [useSearchParams, setSearchParams] = useSearchParams();
```