React IV

Ecosistema npm, *React Router*, `useContext`

Javier Ribal del Río

2026-03-06

Table of contents

Contenido

- Ecosistema React y paquetes npm
- Librerías externas
- React Router
- Parámetros de URL
- `useContext`

1 Ecosistema Node

Hasta ahora hemos trabajado únicamente con **React y JavaScript**.

Sin embargo, en el desarrollo real de aplicaciones es muy común utilizar **librerías externas**, para ello podemos recurrir `Node.JS`.

React está diseñado para funcionar dentro de un **ecosistema de paquetes**.

Ejemplos habituales:

- React Router → navegación
- Axios → peticiones HTTP
- Chart.js → gráficas
- Zustand / Redux → gestión de estado
- Librerías de UI: Tailwind, Bootstrap, Booswatch

2 npm

npm (**Node Package Manager**) es el sistema que permite instalar y gestionar librerías de JavaScript. Como su nombre indica, está desarrollado por Node.JS

Cada proyecto tiene un archivo:

`package.json`

En este archivo se registran las **dependencias del proyecto**.

Ejemplo:

```
{
  "dependencies": {
    "react": "^18.0.0",
    "react-router-dom": "^6.0.0"
  }
}
```

2.1 Instalación de paquetes

Las librerías se instalan desde la terminal.

```
npm install react-router-dom
```

Esto hace tres cosas:

1. Descarga el paquete
2. Lo guarda en `node_modules` (capreta donde se guardan las librerías externas)
3. Añade la dependencia en `package.json`

3 React Router

En una aplicación web tradicional cada enlace carga **una página nueva**.

En React normalmente trabajamos con **Single Page Applications (SPA)**.

En una SPA:

- el navegador no recarga la página
- React cambia los componentes que se muestran

Para gestionar esto utilizamos **React Router**.

Realmente es la misma página, solo que el usuario lo percibe como distintas páginas

El usuario percibirá que el sitio web está divido en diferentes subpáginas

- `/`
- `pokemons`
- `pokemons/pikachu`

3.1 Componentes principales de React Router

Los elementos fundamentales son:

- `BrowserRouter`
- `Routes`
- `Route`
- `Link`

Ejemplo básico:

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

function App() {
  return (
    <BrowserRouter>

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="/pokemons" element={<PokemonList />} />

      </Routes>

    </BrowserRouter>
  );
}
```

Elementos del código

- **`BrowserRouter`**
  Es el componente que activa el sistema de navegación de React Router. Utiliza la API de historial del navegador para cambiar la URL sin recargar la página.
- **`Routes`**
  Es el contenedor donde se definen todas las rutas de la aplicación. React Router examina las rutas dentro de este componente para decidir qué componente mostrar.
- **`Route`**
  Define una ruta concreta de la aplicación.

- `path`
  Indica la URL que activa la ruta.
  Ejemplo: `/` corresponde a la página principal.
- `element`
  Es el componente de React que se renderiza cuando la URL coincide con el `path`.
- `<Home />`
  Componente que se muestra cuando el usuario accede a la ruta `/`.
- `<PokemonList />`
  Componente que se muestra cuando el usuario accede a la ruta `/pokemons`.

3.2 Navegación

Para navegar entre páginas utilizamos el componente `Link`.

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

<Link to="/">Inicio</Link>

<Link to="/pokemon">Pokemon</Link>
```

A diferencia de `<a>`:

- **no recarga la página**
- React cambia el componente visible

3.3 Parámetros dinámicos

Podemos crear rutas dinámicas.

Ejemplo:

```
/pokemon/25
/pokemon/7
```

Definición de la ruta:

```
<Route path="/pokemon/:id" element={<PokemonDetail />} />
```

3.3.1 useParams

Para acceder al parámetro utilizamos el hook `useParams` de *React Router*.

```
import { useParams } from "react-router-dom";

function PokemonDetail() {

  const { id } = useParams();

  return <p>Pokemon {id}</p>;

}
```

Este parámetro puede utilizarse para realizar peticiones a una API.

4 Context API

En aplicaciones grandes aparece un problema frecuente.

Muchos componentes necesitan acceder a la **misma información**.

Por ejemplo:

- usuario
- tema visual
- idioma
- configuración

Si pasamos la información mediante *props*, los datos deben atravesar muchos componentes.

Representación conceptual:

```
App
  Layout
     Page
        Component
```

Si todos necesitan `user`, debemos pasar la prop continuamente.

Este problema se conoce como **prop drilling**.

4.1 Context

React proporciona una solución llamada **Context**.

Context permite **compartir información entre múltiples componentes** sin pasar props manualmente.

El proceso tiene tres pasos:

1. Crear el contexto
2. Proveer el contexto
3. Consumir el contexto

4.1.1 Crear un contexto

```jsx
import { createContext } from "react";

const UserContext = createContext();
```

Esto crea un contenedor que puede almacenar información compartida.

4.1.2 Provider

El `Provider` permite que los componentes hijos accedan al contexto.

```
<UserContext.Provider value={user}>

  <App />

</UserContext.Provider>
```

Todos los componentes dentro del `Provider` pueden acceder al valor.

4.1.3 Consumir el contexto

Para acceder al contexto utilizamos el hook `useContext`.

```
import { useContext } from "react";

const user = useContext(UserContext);
```

El componente obtiene directamente el valor almacenado en el contexto.

4.1.4 Ejemplo completo

```
import { createContext, useContext } from "react";

const UserContext = createContext();

function App() {

  const user = { name: "Javier" };

  return (

    <UserContext.Provider value={user}>

      <Profile />

    </UserContext.Provider>

  );
}

function Profile() {

  const user = useContext(UserContext);

  return <h1>{user.name}</h1>;

}
```