PARTE II — NAVEGACION ENTRE VISTAS

El problema de la navegación en React

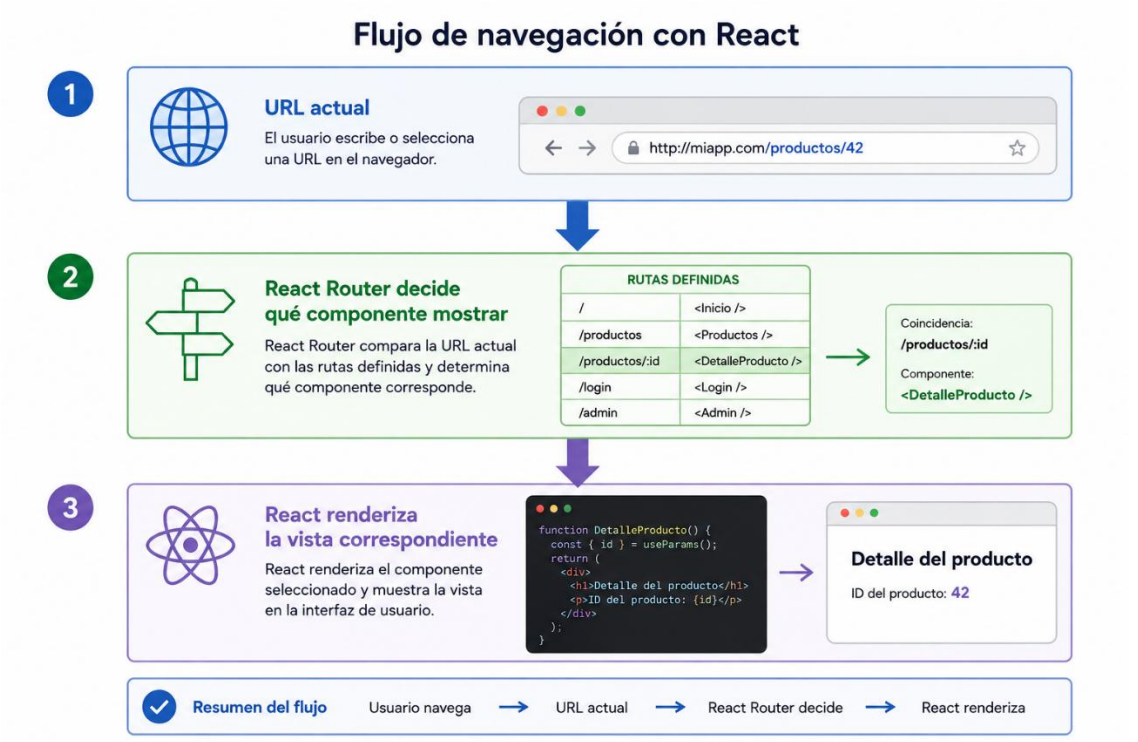
En una web tradicional, cada página suele ser un documento distinto:

```
index.html
productos.html
contacto.html
login.html
```

Cuando el usuario navega, el browser solicita un nuevo archivo al servidor. Esto provoca una recarga completa de la página.

En React, en cambio, generalmente trabajamos con una **SPA**: Single Page Application.

Esto significa que existe una única página HTML principal, pero React cambia la vista visible según la URL. El siguiente gráfico muestre este flujo:



React Router permite configurar rutas declarativas que asocian segmentos de URL con componentes de interfaz. En la documentación actual, las rutas pueden configurarse renderizando Routes y Route, vinculando segmentos de URL con elementos de UI.

Por tanto, React Router no carga páginas HTML nuevas. React Router cambia componentes visibles según la URL.

Instalación

En versiones actuales, la documentación de React Router muestra el uso del paquete react-router para aplicaciones React. Este paquete debe instalarse para poder ser usado:

```
npm install react-router
```

En muchas fuentes bibliográficas anteriores también se observa esta forma de instalación:

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

La diferencia responde a cambios de versión y organización de la librería. Para mantener coherencia con la documentación actual, podemos trabajar con react-router de tal forma que los componentes de navegación se importen de la siguiente manera:

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

Estructura de un proyecto React que soporta navegación

Se propone la siguiente estructura para un ejemplo de proyecto:

```
src/
├── components/
│   ├── Navbar.jsx
│   └── RutaProtegida.jsx
│
├── pages/
│   ├── Inicio.jsx
│   ├── Productos.jsx
│   ├── DetalleProducto.jsx
│   ├── Login.jsx
│   └── Admin.jsx
│
├── App.jsx
└── main.jsx
```

Esto permite establecer una forma de separar responsabilidades para que el proyecto sea escalable, mantenible y fácil de entender para los desarrolladores.

La clave es separar las “vistas” de los “componentes”. En la carpeta `pages/` estarán las vistas, que son pantallas completas, cada una de ellas asociada a una ruta:

- / → Inicio.jsx
- /productos → Productos.jsx
- /productos/:id → DetalleProducto.jsx
- /login → Login.jsx
- /admin → Admin.jsx

React Router va a renderizar según la URL la vista correspondiente. Esto ayuda a:

- visualizar rápidamente qué rutas tiene la app
- mantener cada pantalla aislada
- evitar mezclar componentes pequeños con pantallas completas

Entonces en la carpeta `components/` seguirán estando los componentes reutilizables, que no son páginas:

- Navbar
- Footer
- Tarjetas

- Botones
- Formularios
- RutaProtegida (un wrapper, no una página)

Son piezas que pueden aparecer en varias páginas. Separarlos evita que el directorio `pages/` se llene de cosas que no son rutas.

Cabe destacar que `RutaProtegida.jsx` es un componente lógico, no una página.

Este es un ejemplo típico que permite comprender el concepto:

- Si el usuario no está logueado → redirigir a /login
- Si está logueado → mostrar la página protegida

Por eso va en `components/` y no en `pages/`.

Configuración básica de las rutas

La navegación suele realizarse en `App.jsx` para dejar en `main.jsx` únicamente el montaje de la aplicación. Por ejemplo:

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

import Navbar from "./components/Navbar";
import Inicio from "./pages/Inicio";
import Productos from "./pages/Productos";
import DetalleProducto from "./pages/DetalleProducto";
import Login from "./pages/Login";
import Admin from "./pages/Admin";

const App = () => {
  return (
    <BrowserRouter>
      <Navbar />

      <Routes>
        <Route path="/" element={<Inicio />} />
        <Route path="/productos" element={<Productos />} />
        <Route path="/productos/:id" element={<DetalleProducto />} />
        <Route path="/login" element={<Login />} />
        <Route path="/admin" element={<Admin />} />
      </Routes>
    </BrowserRouter>
  );
};

export default App;
```

`<BrowserRouter>` envuelve la aplicación y permite que React Router controle la navegación.

`<Routes>` agrupa todas las rutas disponibles.

En el ejemplo, `<Route path="/" element={<Inicio />} />` indica que cuando la URL sea /, se debe mostrar el componente `Inicio`.

De manera similar, `<Route path="/productos" element={<Productos />} />` indica que cuando la URL sea /productos, se debe mostrar Productos.

Pero, esta ruta introduce una novedad:

```
<Route path="/productos/:id" element={<DetalleProducto />} />
```

Es la forma que tiene React Router de establecer una ruta dinámica. El valor después de /productos/ es un parámetro. Se puede identificar debido a que `id` es precedido por :.

Esto significa que ante una URL se invocará un componente. Para este ejemplo sería así:

Navegación con Link

¿Cómo logramos que React busque alguna de las rutas definidas para que cargue el componente establecido en una vista?

En HTML tradicional podríamos usar:

```
<a href="/productos">Productos</a>
```

Pero esto, no debe usarse en React ya que puede provocar una recarga completa, mientras que la folosfia de un SPA consiste en actualizar una porción de la página. Para lograrlo, React Router reemplaza la navegación tradicional del browser por una navegación interna al combinar los siguientes elementos:

1. Las rutas → se definen con `<Route>` dentro de `<Routes>`
2. La navegación → se realiza con `<Link>` o `<NavLink>`

Ya vimos que con `<Route>` definimos qué componente se mostrará según su URL. En cambio, con los componentes de navegación (`Link` o `NavLink`) permite cambiar la URL sin recargar la página. Al cambiarla se renderizará automáticamente el componente asociado.

Observe el siguiente ejemplo:

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

const Navbar = () => {
  return (
    <nav>
      <Link to="/">Inicio</Link>
      <Link to="/productos">Productos</Link>
      <Link to="/login">Login</Link>
      <Link to="/admin">Admin</Link>
    </nav>
  );
};

export default Navbar;
```

`Link` el es componente que permite navegar entre rutas sin recargar toda la página. Este componente especial reemplaza al `<a href="">` tradicional ya que cambia la URL y deja en manos de React Router la renderización de la vista correspondiente.

Específicamente en el ejemplo, actúa de la siguiente manera:

- <Link to="/"> → cambia la URL a / y React Router muestra <Inicio />
- <Link to="/productos"> → muestra <Productos />
- <Link to="/login"> → muestra <Login />
- <Link to="/admin"> → muestra <Admin /> (si está protegida, pasará por RutaProtegida)

Todo esto sin recargar la página.

Parámetros dinámicos

Un parámetro dinámico permite representar una ruta variable.

Si por ejemplo tenemos:

`/productos/1`
`/productos/2`
`/productos/25`

Pero no queremos crear una ruta distinta para cada producto es posible generar una única ruta de la siguiente manera:

```
<Route path="/productos/:id" element={<DetalleProducto />} />
```

Donde `:id` representa la definición de un parámetro dinámico. El nombre del parámetro es `id`, y se puede reconocer porque posee : antes del identificador.

React Router permite leer esos parámetros con `useParams`, que devuelve un objeto con los valores dinámicos de la URL.

Veamos su aplicación en el componente <DetalleProducto> invoca en el ejemplo de ruta anterior:

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

const DetalleProducto = () => {
  const { id } = useParams();

  return (
    <section>
      <h1>Detalle del producto</h1>
      <p>Producto seleccionado: {id}</p>
    </section>
  );
};

export default DetalleProducto;
```

En primer lugar, se importa useParams.

En segundo lugar, el siguiente código:

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

obtiene el parámetro `id` definido en la ruta:

```
/productos/:id
```

Supongamos que que el usuario entra a `/productos/7`

Entonces `id = "7"`

Y mostrará en pantalla Producto seleccionado: 7

Redirecciones con Navigate

Una redirección ocurre cuando el sistema decide enviar al usuario a otra ruta. Ejemplo típico:

Para lograrlo, React router utiliza Navigate. Por ejemplo:

```
import { Navigate } from "react-router";

const Admin = () => {
  const autenticado = false;

  if (!autenticado) {
    return <Navigate to="/login" />;
  }

  return (
    <section>
      <h1>Panel de administración</h1>
    </section>
  );
};

export default Admin;
```

Se define una variable `autenticado`. En un caso real, el valor estaría determinado por un contexto (ejemplo resultado de validación contra una base de datos, o un token de autenticación), un hook o un estado global. Acá está en `false` para simular que el usuario NO inició sesión.

Posteriormente, se evalúa la condición

```
if (!autenticado) { ... }
```

Si el usuario NO está autenticado, React devuelve:

Esto provoca una redirección inmediata a la ruta `/login`.

En cambio, si el usuario sí estuviera autenticado, se renderiza:

```
<h1>Panel de administración</h1>
```

Protección de rutas

La protección de rutas permite encapsular la lógica de autorización. Observe el siguiente ejemplo:

```
import { Navigate } from "react-router";

const RutaProtegida = ({ children }) => {
  const autenticado = false;

  if (!autenticado) {
    return <Navigate to="/login" />;
  }

  return children;
};

export default RutaProtegida;
```

En esta versión de código sucede esto:

- Se envuelve a un componente (por ejemplo <Admin />)
- Se evalúa si el usuario está autenticado
- Si NO lo está → redirige a /login
- Caso contrario muestra el contenido protegido

Si hacemos una lectura conceptual más amplia:

```
<RutaProtegida>
  <Admin />
</RutaProtegida>
```

Expresa lo siguiente:

1. Antes de mostrar <Admin />, se debe ejecutar la lógica de autorización.
2. ¿El usuario está autenticado?
   - Sí → mostrar <Admin />
   - No → redirigir a /login

Como se lo usa en las rutas

```
<Route
  path="/admin"
  element={
    <RutaProtegida>
      <Admin />
    <</RutaProtegida>
  }
/>
```

El siguiente gráfico resume el flujo de protección y ejecución