

# React Router

## Módulo 6



Elaborado por: Jesua Luján

Jesua Hadai Luján

DEV.F.:





DEV.F.



## ● Módulo 4: “React Router Domina tu Navegación” 🔥

🎯 *Objetivo: Aprender a usar React Router para movernos entre páginas sin recargar la app.*

🎓 **Contenido:**

✓ ¿Qué es React Router?

Para qué sirve y cómo **mejorar nuestras apps** 🤔

✓ Instalación y configuración básica:

**`npm install react-router-dom`** 🧑

✓ Rutas simples:

**`<BrowserRouter>`, `<Routes>` y `<Route>`**

Cómo mostrar diferentes componentes según la URL 🧑

✓ Navegación entre páginas: **`<Link>` y `<Navigate>`**



# React Router



## ¿Qué es React Router?

**React Router** es una librería oficial para manejar rutas en aplicaciones React. Permite crear una navegación fluida tipo **SPA** (*Single Page Application*), donde el usuario puede moverse entre distintas "páginas" **sin recargar el navegador**.

En vez de tener un archivo **.html** por cada vista, usamos **componentes** que se muestran dinámicamente según la URL.



# React Router



## ¿Por qué usar React Router?

- ✓ Navegación sin recarga.
- ✓ URLs legibles y compartibles.
- ✓ Manejo de rutas dinámicas `/user/:id`.
- ✓ Rutas protegidas (ej. solo usuarios logueados).
- ✓ Rutas anidadas (ej. `/admin/users`).
- ✓ Redirecciones automáticas.
- ✓ Control total desde el código.

## vs ¿Qué diferencia a React Router v6 de v5?



Característica	v5	v6
Rutas	Anidadas con <b>Switch</b>	Anidadas con Routes
Redirección	<b>&lt;Redirect&gt;</b>	<b>&lt;Navigate&gt;</b>
Orden de rutas	Se revisaban manualmente	Automáticamente por coincidencia más específica
Render	<b>component, render</b>	Solo <b>element</b> (más claro)
Hooks	Menos integrados	Más hooks nativos ( <b>useRoutes, etc</b> )





# React Router Dom

React utiliza una biblioteca externa para manejar el enrutamiento; sin embargo, antes de que podamos implementar el enrutamiento con esa biblioteca, primero debemos instalarlo en nuestro proyecto, que se logra ejecutando el siguiente comando en la terminal **(dentro de su carpeta raíz)**

## Install


```
> npm i react-router-dom
```



## Repository

 [github.com/remix-run/react-router](https://github.com/remix-run/react-router)

## Homepage

 [github.com/remix-run/react-router#r...](https://github.com/remix-run/react-router#r...)

## Weekly Downloads

9.492.312



## I 2. Estructura Básica

### <BrowserRouter>

- Envoltorio principal que permite usar el historial del navegador.
- Solo se declara una vez en el nivel más alto (normalmente en **main.jsx**).

```
<BrowserRouter>
  <Switch>
    <Route exact path="/" component={DashResponsive} />
    <Route path="/login" component={Login} />
    <Route path="/signup" component={Signup} />
  </Switch>
</BrowserRouter>
```

old version



## I 2. Estructura Básica

[ 1.1 ]

### <BrowserRouter>

- Envoltorio principal que permite usar el historial del navegador.
- Solo se declara una vez en el nivel más alto (normalmente en **main.jsx**).

main.jsx o index.jsx:

Envuelve tu app con **<BrowserRouter>**, el componente que escucha cambios en la URL.

```
import { BrowserRouter } from 'react-router-dom'  
import App from './App'  
  
<BrowserRouter>  
  <App />  
</BrowserRouter>
```





## I Componentes clave

### <Routes> y <Route>

- Reemplazan al viejo <Switch>.
- <Routes> agrupa todas las rutas.
- <Route> define qué componente se muestra según el path.

```
import { Routes, Route } from 'react-router-dom'
import Home from './pages/Home'
import About from './pages/About'

function App() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
    </Routes>
  )
}
```



## I Componentes clave

- **path:** Como su nombre lo indica, esto identifica la ruta que queremos que los usuarios tomen para alcanzar el componente establecido. Cuando configuramos el **path** a **/about**, por ejemplo, cuando el usuario agrega **/about** al enlace URL, se navega a esa página.

```
// App.js
import { Routes, Route } from 'react-router-dom';
import Home from './Pages/Home';
import About from './Pages/About';
import Products from './Pages/Products';

const App = () => {
  return (
    <>
      <Routes>
        <Route path="/" element={ <Home /> } />
        <Route path="/products" element={ <Products /> } />
        <Route path="/about" element={ <About /> } />
      </Routes>
    </>
  );
};

export default App;
```



## I Componentes clave

- **element:** Contiene el **componente** que queremos que se cargue en la **ruta establecida**. Recuerda importar cualquier componente que estemos usando aquí, o de lo contrario se producirá un error.

```
// App.js
import { Routes, Route } from 'react-router-dom';
import Home from './Pages/Home';
import About from './Pages/About';
import Products from './Pages/Products';

const App = () => {
  return (
    <>
      <Routes>
        <Route path="/" element={ <Home /> } />
        <Route path="/products" element={ <Products /> } />
        <Route path="/about" element={ <About /> } />
      </Routes>
    </>
  );
};

export default App;
```



## I Componentes clave

### <Link> vs <a>

- <Link> evita que la página recargue.
- Usar <a> rompe el comportamiento SPA.

```
<Link to="/dashboard">Ir al dashboard</Link>
```

```
<Link to="/about">Ir a About</Link>  
<NavLink to="/dashboard" className={({  
  isActive }) => isActive ? 'active' : ''}  
>
```



## 📌 Teoría: <Link> vs <NavLink> en React Router

### 🧪 Buenas prácticas

- Usa **<NavLink>** para menús o barras de navegación.
- Usa **<Link>** en cualquier otra parte donde no importe el "estado activo" (ej. botones o CTA).
- Evita usar **<a href="">** a menos que estés apuntando a URLs externas.

[https://api.reactrouter.com/v7/functions/react\\_router.NavLink.html](https://api.reactrouter.com/v7/functions/react_router.NavLink.html)



## Hooks útiles



Hook	¿Para qué sirve?
<code>useNavigate()</code>	Redirigir a otra ruta desde código
<code>useParams()</code>	Leer parámetros dinámicos de la URL ( <code>:id</code> , <code>:slug</code> )
<code>useLocation()</code>	Leer la URL actual y su estado
<code>useRoutes()</code>	Definir rutas desde un array JS (más dinámico)





## Patrón:

```
const PrivateRoute = ({ children }) => {  
  const isAuthenticated = // lógica de sesión  
  return isAuthenticated ? children : <Navigate to="/login" />  
}
```

## Uso:

```
<Route path="/dashboard" element={  
  <PrivateRoute>  
    <Dashboard />  
  </PrivateRoute>  
} />
```



## Rutas Protegidas (Private Routes)

Se usan para **restringir el acceso** a ciertas rutas según el estado del usuario.



## Buenas Prácticas



Práctica	Descripción
Agrupar tus rutas por tipo	<code>routes/PublicRoutes.jsx</code> , <code>PrivateRoutes.jsx</code>
Usa contexto para manejar sesión	<code>AuthContext</code> o <code>AuthProvider</code>
Evita el uso de <code>&lt;a&gt;</code> para navegación	Siempre <code>&lt;Link&gt;</code> o <code>useNavigate()</code>
Define componentes con nombres claros	<code>&lt;Login&gt;</code> , <code>&lt;Signup&gt;</code> , <code>&lt;Dashboard&gt;</code>
Separa lógica de autenticación	Componente envoltorio para rutas privadas





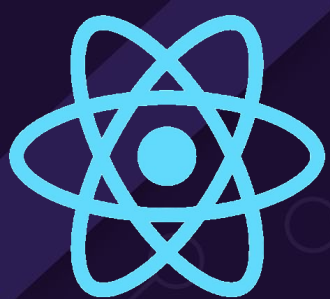


## Conclusión

**React Router** es el estándar de facto para navegación en React. Con él puedes:

- Tener un flujo de **usuario claro**.
- Controlar **acceso a contenido**
- Separar tus **componentes por responsabilidades**
- Crear UX modernas, **sin recargas y con navegación fluida**.





# Ejemplo de Clase

**DEV.F**  
DESARROLLAMOS(PERSONAS);



# | Objetivo del Proyecto Introductorio



Crear una app React básica con:

1. 3 componentes clave: **Navbar, Login, Signup**
2. Formularios profesionales con Tailwind CSS
3. Simulación de sesión simple usando **useState y localStorage**
4. Rutas organizadas en una carpeta
5. Renderizado condicional: mostrar diferentes opciones si el usuario está logueado o no

### Iniciar Sesión

Correo electrónico

Contraseña















Entrar



# | Objetivo del Proyecto Introductorio



## Setup del proyecto

- ✓  src
  - >  assets
  - ✓  components
    -  Navbar.jsx
  - ✓  pages
    -  Login.jsx
    -  Signup.jsx
  - ✓  routes
    -  AppRoutes.jsx
- ✓  styles
  -  index.css
  -  App.jsx
  -  index.css
  -  main.jsx



# | VITE + REACT + TAILWIND



## Create your project

Start by creating a new Vite project if you don't have one set up already. The most common approach is to use Create Vite.



<https://v3.tailwindcss.com/docs/installation/using-postcss>

# VITE + REACT + TAILWIND

## Install Tailwind CSS

Install `tailwindcss` and its peer dependencies,  
then generate your `tailwind.config.js` and  
`postcss.config.js` files.

Terminal

```
> npm install -D tailwindcss@3 postcss autoprefixer  
> npx tailwindcss init
```



<https://v3.tailwindcss.com/docs/guides/vite>



# | VITE + REACT + TAILWIND

Add the Tailwind directives to your CSS

Add the @tailwind directives for each of Tailwind's layers to your main CSS file.

main.css

```
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

postcss.config.js

```
module.exports = {  
  plugins: {  
    tailwindcss: {},  
    autoprefixer: {},  
  },  
}
```

tailwind.config.js

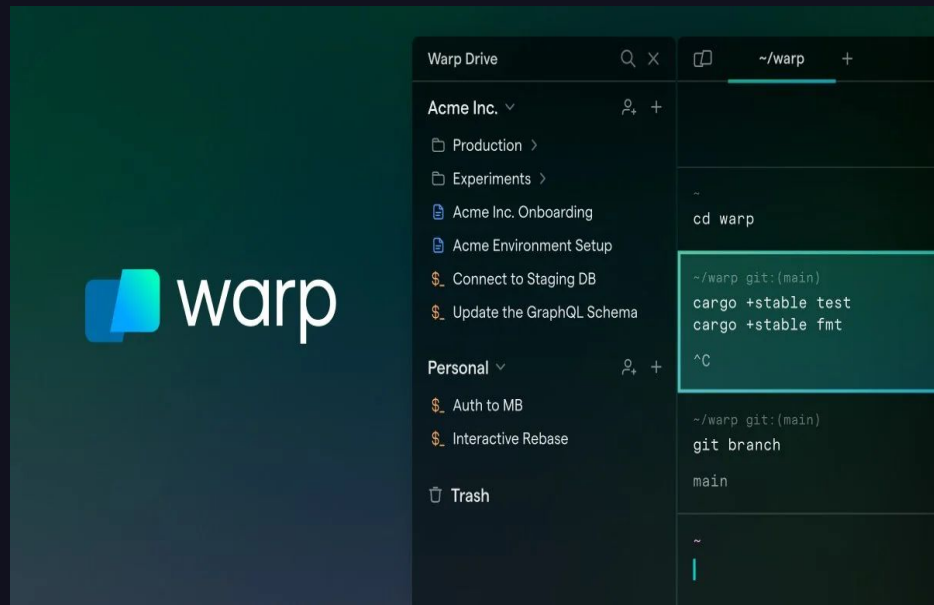
```
/** @type {import('tailwindcss').Config} */  
module.exports = {  
  content: ["./src/**/*.html,js"],  
  theme: {  
    extend: {},  
  },  
  plugins: [],  
}
```

<https://v3.tailwindcss.com/docs/guides/vite>





**Warp** es un **terminal moderno para desarrolladores**, creado para reemplazar el clásico terminal de texto (como el de macOS, Linux o el CMD/PowerShell de Windows) con una experiencia mucho más interactiva, rápida y centrada en productividad.



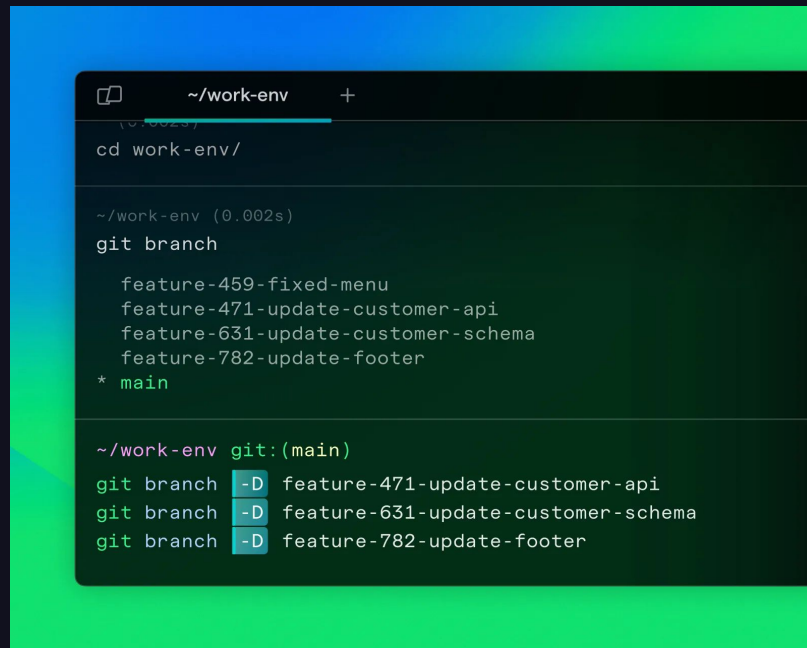
<https://www.warp.dev/>





# I WARP

Si vives en la terminal y sientes que iTerm, gnome-terminal, o el clásico Terminal.app ya se quedaron en el pasado, Warp es como pasar de escribir en bloc de notas a usar VSCode.



```
~/work-env +
(0.002s)
cd work-env/

~/work-env (0.002s)
git branch

feature-459-fixed-menu
feature-471-update-customer-api
feature-631-update-customer-schema
feature-782-update-footer
* main

~/work-env git:(main)
git branch -D feature-471-update-customer-api
git branch -D feature-631-update-customer-schema
git branch -D feature-782-update-footer
```

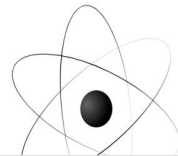
<https://www.warp.dev/>



# I **BONUS**

Imagina que tienes una duda que hemos visto durante el curso, por ejemplo que son las **props en react**, en [reactjs.wiki](https://reactjs.org/docs/faq-props.html) puedes consultar teoría y ejemplos prácticos.

## Preguntas típicas de React.js



¿Qué son las props en React?

### Las preguntas más buscadas...

#### PRINCIPIANTE

##### ¿Para qué sirve useEffect?

El hook useEffect se usa para ejecutar código cuando se renderiza el componente o cuando cambian las dependencias del efecto.

[Leer más...](#)

#### INTERMEDIO

##### Cómo cancelar una petición fetch

Cuando hacemos una petición a una API, podemos cancelarla para evitar que se ejecute cuando el componente se desmonte usando AbortController como hacemos en este ejemplo

[Leer más...](#)

#### INTERMEDIO

##### ¿Qué es la hidratación?

La hidratación convierte el HTML estático que devuelve el servidor en HTML interactivo que puede responder a eventos del usuario en el cliente.

[Leer más...](#)

<https://www.reactjs.org/docs/faq-props.html>

