


**HENRY**

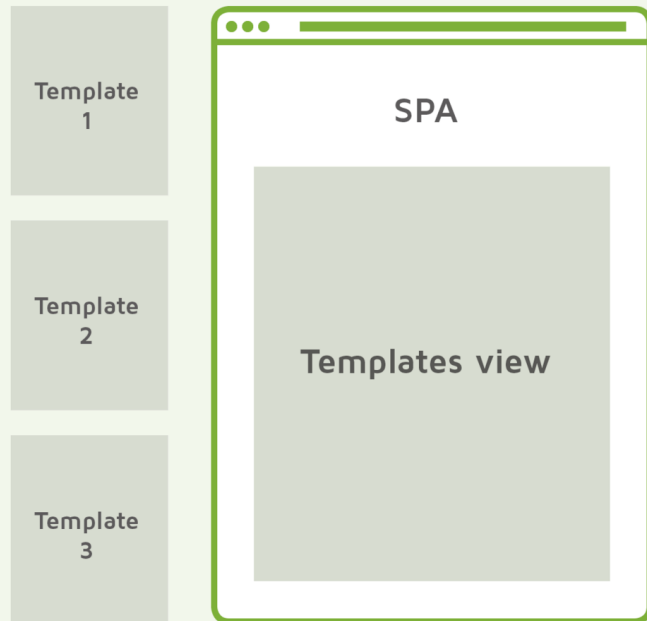
A bright yellow beam of light originates from the left edge of the frame and points towards the letter 'R' in the word 'HENRY'. The beam is wider on the left and tapers as it moves to the right. The word 'HENRY' is written in a bold, black, sans-serif font.



# ROUTING

# SPA

## Single Page Application



No page refresh on request

## Traditional Web Application



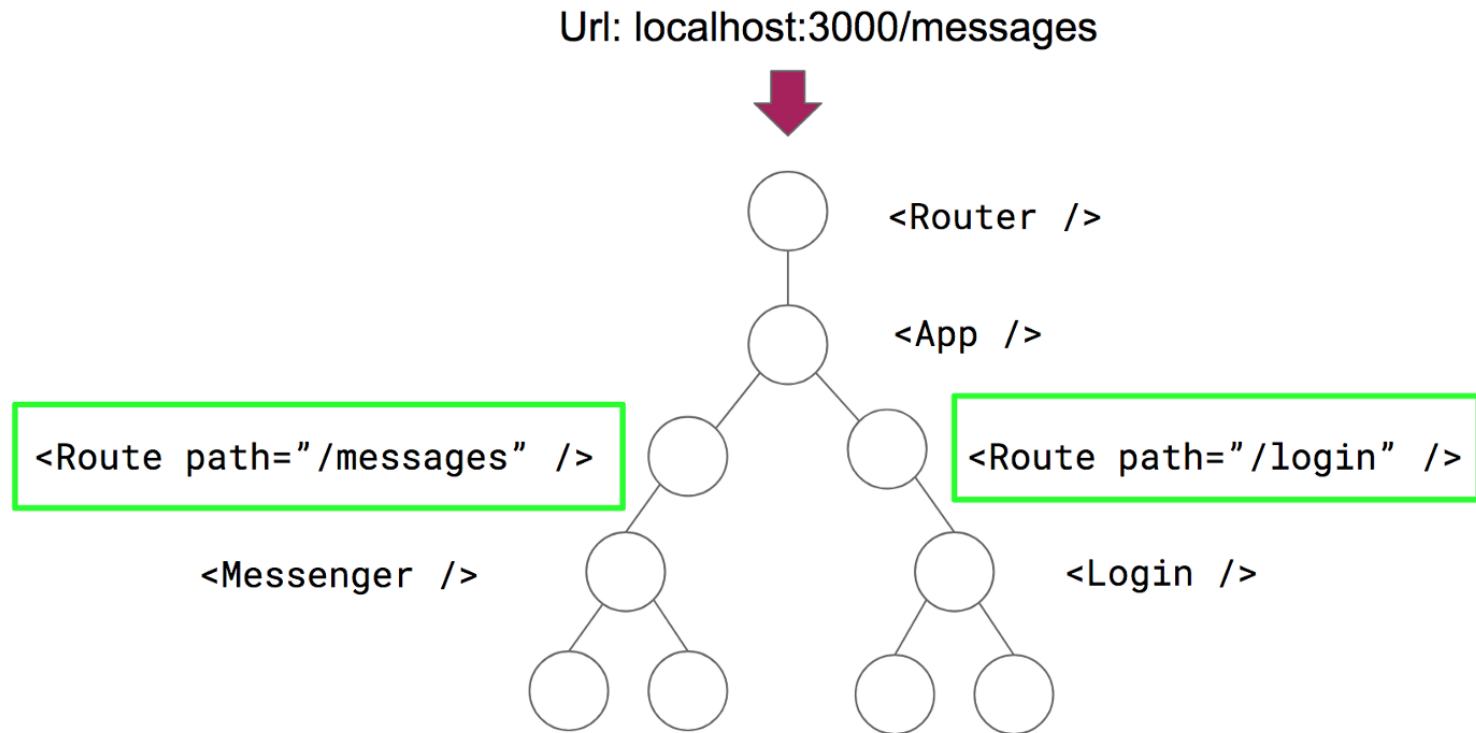
Whole page refresh on request

# SPA

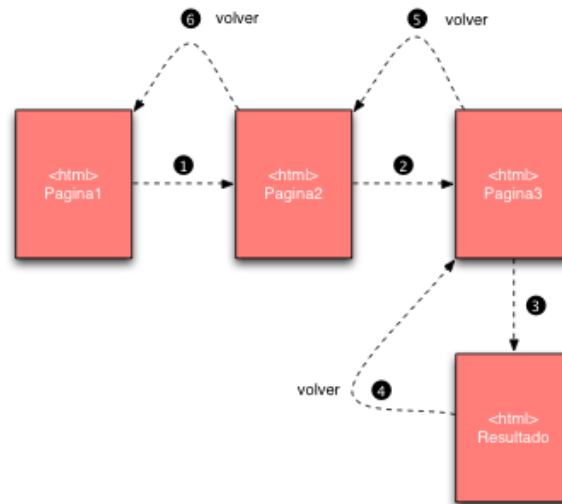


Librería para definir de forma declarativa la vistas que queremos renderizar dependiendo la URL

# SPA - Routing



# SPA - History API



El objeto DOM [window](#) proporciona acceso al historial del navegador a través del objeto [history](#) . Este da acceso a métodos y propiedades útiles que permiten avanzar y retroceder a través del historial del usuario, así como manipular el contenido del historial.

Documentación API

# Routing Config

Para configurar nuestra aplicación primero tenemos que decidir que tipo de router vamos a usar:



<HashRouter />

http://example.com/#/about



Configuración del servidor más simple ya que siempre el request es a la misma URL



Agrega un # a la URL

<BrowserRouter />

http://example.com/about



Configuración extra en ambiente de producción (DEV out of the box)



URL más prolija

```
1 // Instalar React-Router-DOM
2 npm i react-router-dom
3
4 // Utilización
5 import { HashRouter, Route } from 'react-router-dom';
6
7 ReactDOM.render((
8   <HashRouter>
9     <App />
10  </HashRouter>
11 ), document.getElementById('app'))
```

# <Route />

```
1 <Route
2   exact
3   strict
4   sensitive
5   path="..."
6   component={Component}
7   render={() => <Component />}
8   children={() => <Component />}
9 >
10 ...
11 </Route>
```

## Propiedades del componente <Route />

- **path**: URL o array de URLs que se van a comparar contra el path actual para ver si renderizar o no los componentes definidos en esa ruta
- **exact**: si se encuentra dentro de route, el path debe coincidir por completo y no únicamente su inicio
- **strict**: si se encuentra dentro de route, el path debe coincidir incluso en las barras (' / ') del path
- **sensitive**: si se encuentra dentro de route, el path sera case sensitive

< demo-route />



# Basic Routing

```
1 import React from 'react';
2 import { render } from 'react-dom';
3 import { Route, Switch, HashRouter as Router, useParams } from 'react-router-dom';
4
5 const Root = (
6   <Router>
7     <NavBar />
8     <Switch>
9       <Route exact path="/">
10         <Home />
11       </Route>
12       <Route path="/about">
13         <About />
14       </Route>
15       <Route path="/ejemplo">
16         <Ejemplo nombre="Toni" apellido="Tralice"/>
17       </Route>
18       <Route path="/">
19         <h2>Default if no match</h2>
20       </Route>
21     </Switch>
22   </Router>
23 );
24
25 render(Root, document.querySelector('#app'));
```

# <Link />



```
1 import React from 'react';
2 import { Link } from 'react-router-dom';
3
4 export default function NavBar() {
5   return (
6     <div className="nav-bar">
7       <h2>Barra de Navegación</h2>
8       <Link to="/">Home</Link>
9       <Link to="/about">About</Link>
10      <Link to="/ejemplo">Ejemplo</Link>
11    </div>
12  );
13 };
```

El componente <Link /> se va a traducir a un tag <a> que nos va a permitir redireccionar al usuario hacia una nueva URL

- **to**: nos indica el path hacia el cual debemos redirigir una vez clickeado el link

# <NavLink />



```
1 export default function NavBar() {
2   return (
3     <div className="nav-bar">
4       <h2>Barra de Navegación</h2>
5       {/* <NavLink to="/">Home</NavLink> */}
6       <NavLink exact to="/">Home</NavLink>
7       <NavLink to="/about" activeClassName="selected">About</NavLink>
8       <NavLink to="/ejemplo" activeStyle={{
9         color: 'green'
10      }}>Ejemplo</NavLink>
11     </div>
12   );
13 };
```

El componente `<NavLink />` mantiene la misma funcionalidad que `<Link>` pero permite dar estilos dependiendo la ruta actual

- **activeClassName**: nos indica la clase que se va a asignar a ese link cuando la ruta coincida con la especificada
- **exact**: si lo agregamos solo va a aplicar los estilos cuando la ruta coincida en su totalidad y no solo en su comienzo
- **activeStyle**: permite definir un style de forma inline para cada link

# <Route /> - Legacy Methods

```
1 <Route
2   exact
3   strict
4   sensitive
5   path="..."
6   component={Component}
7   render={() => <Component />}
8   children={() => <Component />}
9 >
10 ...
11 </Route>
```

Métodos legacy del componente <Route />

- **component**: toma un componente y lo crea usando `React.createElement`
- **render**: toma una función que retorna UI. Útil para hacer inline render y pasar props al componente a renderizar.
- **children**: similar a render, pero siempre renderiza.

Si el path y la ubicación actual coinciden se crea un objeto conteniendo tres propiedades (match, location y history) el cual se pasa como prop del componente renderizado o como parámetro de la inline function.

# <Route /> - Route Props

```
1 <Route
2   path="/home"
3   component={Home}
4 />
5
6
7 <Route
8   path("/:id"
9   render=(({match, location, history}) => (
10     <Home match={match} Ejemplo />
11   ))
12 />
13
14 <Route
15   path("/:id"
16   children={({props}) => (
17     <Home {...props} Ejemplo />
18   ))
19 />
```

Home

props

- ▶ **history**: {action: "PUSH", block: f block() {}, createHref: f...}
- ▶ **location**: {hash: "", pathname: "/component", search: "", stat...}
- ▶ **match**: {isExact: true, params: {...}, path: "/component", ur...}

# <Route /> - Route Props

## match

Propiedades del objeto match

- **isExact**: boolean que será true si la ruta matchea exactamente, false de lo contrario
- **params**: objeto con los valores de los parametros de la ruta
- **path**: string que contiene el patrón utilizado para comparar contra la URL actual
- **url**: string que contiene la porción matcheada del url.

```
▼ match: {isExact: true, params: {...}, path: "/component/:fir...}
  isExact: true
  ▶ params: {firstParam: "one", secondParam: "two"}
  path: "/component/:firstParam/:secondParam"
  url: "/component/one/two"
```

# <Route /> - Route Props

## location

Propiedades del objeto location

- **pathname**: string indicando el path actual
- **search**: string que contiene los parámetros pasados por query a la ruta actual. Si no se pasa ninguno, será un string vacío
- **state**: propiedad que por default es undefined pero en la cual podemos pasar datos extras a la nueva ruta

```
▼ location: {hash: "", pathname: "/component/one/two", search: ...}  
  hash: ""  
  pathname: "/component/one/two"  
  search: ""  
  state: undefined
```

# <Route /> - Route Props

## history

### Propiedades del objeto history

- **action**: string que indica la acción que se realizó para llegar a la ruta actual
- **block**: función que va a bloquear la posibilidad de navegación hacia otras rutas
- **go**: función que nos permite movernos n lugares en nuestro stack de navegación. Pueden ser números negativos para ir hacia atras
- **goBack**: función que permite navegar una posición hacia atras. Es equivalente a "go(-1)"
- **goForward**: función que permite navegar una posición hacia adelante. Es equivalente a "go(1)"
- **length**: número que representa la cantidad de entradas en el stack de navegación
- **push**: función que nos permite agregar otra entrada al stack de navegación
- **replace**: función que nos permite reemplazar la entrada actual en el stack de navegación por otra

```
▼ history: {action: "PUSH", block: f block() {}, createHref: f...}
  action: "PUSH"
  block: f block() {}
  createHref: f createHref() {}
  go: f go() {}
  goBack: f goBack() {}
  goForward: f goForward() {}
  length: 50
  listen: f listen() {}
  ▶ location: {hash: "", pathname: "/component/one/two", search: ...}
  push: f push() {}
  replace: f replace() {}
```



# <Route /> - Route Props

location vs history.location



El objeto history es mutable por lo que la recomendación es que si necesitamos utilizar 'location' lo hagamos desde esa propiedad y no desde

```
1 class Comp extends React.Component {
2   componentDidUpdate(prevProps) {
3     // will be true
4     const locationChanged =
5       this.props.location !== prevProps.location;
6
7     // INCORRECT, will *always* be false because history is mutable.
8     const locationChanged =
9       this.props.history.location !== prevProps.history.location;
10  }
11 }
12
13 <Route component={Comp} />;
```

# useParams

```
1 import React from 'react';
2 import { useParams } from 'react-router-dom';
3
4 export default function Mostrar() {
5   let params = useParams();
6   return (<span>Estoy en la ciudad {params.ciudadId}</span>)
7 };
```

Params: ▼ Object { ciudadId: "5" }  
    ciudadId: "5"  
    ▶ <prototype>: Object { ... }

Cuando la URL especificada dentro de <Route> es dinámica, es decir, que puede variar podemos obtener el o los valores pasados como parámetro dentro del path mediante el hook **useParams**

# useHistory

```
1 import React from 'react';
2 import { useHistory } from "react-router-dom";
3
4 export default function History() {
5   let history = useHistory();
6
7   ...
8 };
```

History:

- Object { length: 31, action: "PUSH", location: {...},  
createHref: createHref(location) <sup>⌵</sup>, push: push(path, state) <sup>⌵</sup>,  
replace: replace(path, state) <sup>⌵</sup>, go: go(n) <sup>⌵</sup>, goBack: goBack() <sup>⌵</sup>,  
goForward: goForward() <sup>⌵</sup>, block: block(prompt) <sup>⌵</sup>, ... }
- action: "PUSH"
- block: function block(prompt) <sup>⌵</sup>
- createHref: function createHref(location) <sup>⌵</sup>
- go: function go(n) <sup>⌵</sup>
- goBack: function goBack() <sup>⌵</sup>
- goForward: function goForward() <sup>⌵</sup>
- length: 31
- listen: function listen(listener) <sup>⌵</sup>
- location: Object { pathname: "/history", search: "", hash: "", ... }
- push: function push(path, state) <sup>⌵</sup>
- replace: function replace(path, state) <sup>⌵</sup>

Hook que nos permite acceder a la instancia de History sin necesidad de pasarla explícitamente como vimos con las propiedades del componente <Route /> previamente

# useLocation

```
1 import React from 'react';
2 import { useLocation } from 'react-router-dom';
3
4 export default function Location() {
5   let location = useLocation();
6
7   ...
8 };
```

```
Location:
▼ Object { pathname: "/location", state: {...}, search: "",
hash: "" }
  hash: ""
  pathname: "/location"
  search: ""
  ▼ state: Object { extraData: "Henry" }
    extraData: "Henry"
```

Hook que nos permite acceder al objeto Location sin necesidad de pasarla explícitamente como vimos con las propiedades del componente `<Route />` previamente

# Nested Routing

```
1  const Root = (  
2    <Router>  
3      <NavBar />  
4      <Switch>  
5        <Route path="/home/linkRelative">  
6          <h2>Estoy en /home/linkRelative</h2>  
7        </Route>  
8        <Route path="/home">  
9          <Home />  
10       </Route>  
11       <Route path="/">  
12         <h2>Default</h2>  
13       </Route>  
14     </Switch>  
15   </Router>  
16 );
```

```
1  function Home() {  
2    let match = useRouteMatch();  
3    return (  
4      <div>  
5        <h2>Home, Soy Henry!!</h2>  
6        <Link to="/linkAbsolute">Link Absolute</Link>  
7        <br></br>  
8        <Link to={`/${match.url}/linkRelative`}>Link Relative</Link>  
9      </div>  
10    );  
11  };
```

# <Prompt />

Componente que nos va a permitir mostrar un pop-up de confirmación si el usuario quiere irse del path actual.

```
1 <Prompt
2   when={condition}
3   message="Are you sure you want to leave?"
4 />
```

- **when**: booleano que va a determinar si el prompt debe mostrarse o no cuando se intente navegar hacia otro path
- **message**: podemos pasarle un string o una función. En el primer caso será simplemente el mensaje del pop-up y el segundo permite mayor customización, posibilitando el acceso a 'location' y 'action'

# <Redirect />

Componente que al renderizarse automáticamente nos redirigirá hacia un nuevo path reemplazando la ubicación actual en el stack de navegación.

```
1 <Route exact path="/">
2   {loggedIn ? <Redirect to="/dashboard" /> : <PublicHomePage />}
3 </Route>
```

- **to**: puede ser un string indicando la URL a la cual debe redirigir o un objeto con mayor información (pathname, search y state)
- **from**: string indicando una URL que si la aplicación intenta acceder automáticamente redirigirá hacia la URL indicando en el **to**

**exact, strict, sensitive**



**< demo-extra />**