

## Guía de estudio 3 - React Router II



¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

### ¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo recordar y repasar los contenidos que hemos visto en clase, dentro de los que se encuentran:

- Crear aplicaciones web en React que tengan sistemas de navegación, redirección y captura de parámetros en url.

**¡Vamos con todo!**



## Tabla de contenidos

<b>Guía de estudio 3 - React Router II</b>	<b>1</b>
¿En qué consiste esta guía?	1
Tabla de contenidos	2
La clase active	3
El componente NavLink	3
¡Manos a la obra! - La clase active	6
El hook useNavigate()	6
¡Manos a la obra! - Redireccionamiento programático	8
El hook useParams()	8
¡Manos a la obra! - Obteniendo parámetros de la URL	10
Preguntas de cierre	10



**¡Comencemos!**

## La clase active

Existen consideraciones visuales que aportan a una mejor experiencia de usuario, entre ellas, personalizar el estilo de la opción que representa la ruta actual.

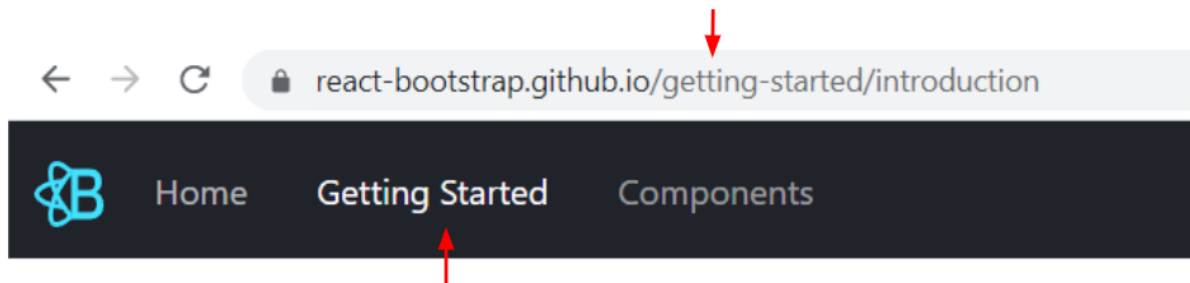


Imagen 1. Ejemplo de coincidencia entre ruta y opción

Fuente: Desafío Latam

Este pequeño, pero importante cambio visual, ayudará a nuestros usuarios a estar mejor ubicados mientras navegan en nuestra aplicación.

## El componente NavLink

React Router nos provee un componente que nos ayudará a saber cuál de las opciones de navegación corresponde a la ruta consultada.

```
<NavLink  
  className={ ({ isActive }) => (isActive ? "nombre-de-la-clase" : undefined) }  
  to="/" >  
  Home  
</NavLink>
```

A diferencia del ya conocido componente Link, este componente tiene integrado un atributo **isActive** que nos devolverá **true** o **false** si la ruta está siendo consultada.

Para acceder a este atributo, necesitamos definir una función como valor del atributo **className**, la cual tendrá como parámetro un objeto con la propiedad **isActive** y deberá retornar el nombre de la clase que será asignada a este elemento.

Para ver en la práctica el uso de este componente, crea una nueva aplicación de React e instala React Router y realicemos los siguientes pasos:

1. Agrega una clase `.active` en **style.css**

```
.App {  
  font-family: sans-serif;  
  text-align: center;  
}  
  
.active {  
  color: darkgreen;  
}
```

2. Crea un componente **Navbar** con 2 opciones: Home y Personajes

```
import { NavLink } from "react-router-dom";  
  
export default function Navbar() {  
  const setActiveClass = ({ isActive }) => (isActive ? "active" : undefined);  
  return (  
    <div>  
      <NavLink className={setActiveClass} to="/">  
        Home  
      </NavLink>  
      {" - "}  
      <NavLink className={setActiveClass} to="/personajes">  
        Personajes  
      </NavLink>  
    </div>  
  );  
}
```

3. Modifica el código de **App.js** para agregar el enrutador e incluir el Navbar creado

```
import "./styles.css";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Navbar from "./components/Navbar";

export default function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <Navbar />
      </BrowserRouter>
    </div>
  );
}
```

El resultado de lo anterior lo podremos apreciar levantando la aplicación y visitando las rutas

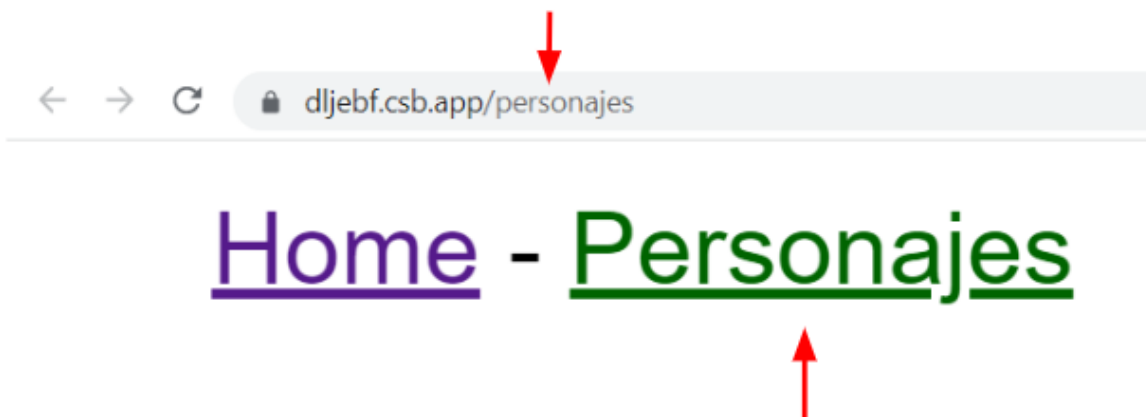


Imagen 2. Ejemplo de la clase active  
Fuente: Desafío Latam



## ¡Manos a la obra! - La clase active

Continuando con nuestra aplicación:

1. Agrega una nueva opción en el Navbar.
2. Crea una nueva clase para definir los estilos de la ruta activa.
3. Crea una nueva clase para definir los estilos de las rutas que no estén activas.
4. Comprueba que las nuevas clases son asignadas correctamente al consultar las rutas.

## El hook `useNavigate()`

En algunos casos podríamos necesitar redireccionar a nuestros usuarios sometiendo esta acción a una condición o simplemente agregar en la URL de destino alguna variable.

A esto se le conoce como un redireccionamiento programático y en React Router podemos realizarlo utilizando el hook `useNavigate()`

Veamos un ejemplo del redireccionamiento programático creando una vista que reciba del usuario un id numérico y redirija a una ruta `/personajes/:id`

```
import { useState } from "react";
import { useNavigate } from "react-router-dom";

export default function Home() {
  const [id, setId] = useState("");
  const navigate = useNavigate();
  const irAPersonajes = () => {
    navigate(`/personajes/${id}`);
  };
  return (
    <div className="mt-5">
      <h1>React Router II</h1>
      <input
        type="number"
        value={id}
        onChange={({ target }) => setId(target.value)}
      />
      <button onClick={irAPersonajes}>buscar</button>
    </div>
  );
}
```

Ahora creemos una vista Personajes que simplemente retorne un `OK`

```
export default function Personajes() {  
  return (  
    <div className="mt-5">  
      <h1>OK</h1>  
    </div>  
  );  
}
```

Finalmente hay que agregar en el componente App las rutas relacionadas a las vistas creadas

```
import "./styles.css";  
import { BrowserRouter, Routes, Route } from "react-router-dom";  
import Navbar from "./components/Navbar";  
  
import Home from "./views/Home";  
import Personajes from "./views/Personajes";  
  
export default function App() {  
  return (  
    <div className="App">  
      <BrowserRouter>  
        <Navbar />  
        <Routes>  
          <Route path="/" element={<Home />} />  
          <Route path="/personajes/:id" element={<Personajes />} />  
        </Routes>  
      </BrowserRouter>  
    </div>  
  );  
}
```

Probemos nuestra aplicación escribiendo un id en el input y presionando el botón

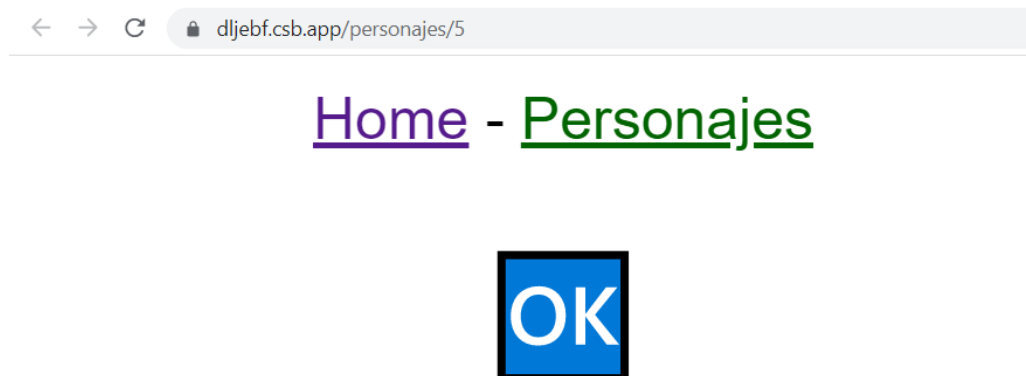


Imagen 3. Redireccionamiento programático  
Fuente: Desafío Latam



## ¡Manos a la obra! - Redireccionamiento programático

Continuando con nuestra aplicación:

1. Cambia el input por un selector del 1 al 10.
2. Condiciona el redireccionamiento en función de que la variable **id** contenga algún valor.
3. Comprueba los cambios en el navegador.

## El hook `useParams()`

Con React Router podemos definir en el **path** de una ruta el reconocimiento de parámetros

```
<Route path="/categoria/:nombre" element={<Componente />} />
```

Por ejemplo, si se consulta: `/categoria/pib`


El parámetro **nombre** tomará como valor: `"pib"`



Para que un componente acceda a los parámetros de la URL, se necesita ocupar el hook **useParams()**

Al ser ejecutado, este hook nos devuelve un objeto con todos los parámetros.

Veamos un ejemplo capturando el **id** del personaje que logramos incluir en la URL a través de un redireccionamiento programático

Modifica el código de la vista *Personajes* para que reciba y muestre el parámetro en lugar del 

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

export default function Personajes() {

  const { id } = useParams();

  return (
    <div className="mt-5">
      <h1>{ id }</h1>
    </div>
  );
}
```

Si probamos nuevamente el redireccionamiento programático, ahora veremos que se visualiza el valor del id en el componente

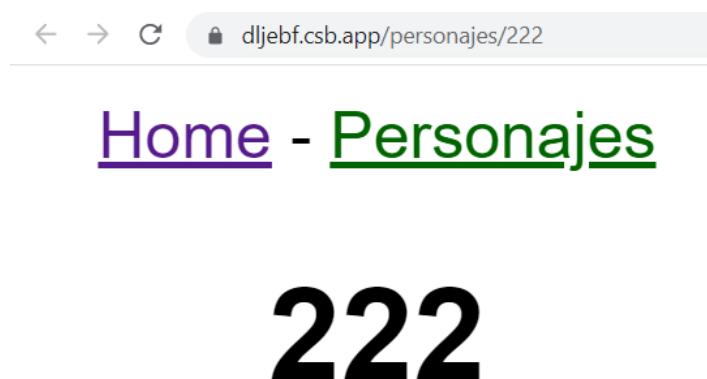


Imagen 4. El id recibido como parámetro en el componente  
Fuente: Desafío Latam



## ¡Manos a la obra! - Obteniendo parámetros de la URL

En la vista Personajes:

1. Utiliza el **id** recibido en parámetros para consultar la [API de Rick And Morty](#) y obtener la información del personaje que coincida con el id.
2. Agrega React Bootstrap al proyecto para ocupar el componente Card y mostrar la información del personaje.
3. Condiciona el componente Card a solo mostrarse en caso que haya obtenido un personaje con el id obtenido desde la vista Home, de lo contrario, muestra en el componente un texto indicando que no se encontró ningún personaje con el id ingresado.

## Preguntas de cierre

### Reflexiona:

- ¿Qué diferencia tienen los componentes **Link** y **NavLink**?
- ¿Qué utilidades nos provee el redireccionamiento programático?
- ¿Cómo se definen los parámetros al momento de declarar un **path** en una ruta?
- ¿Qué nos devuelve la ejecución del hook **useParams()**?

