

**Universidad Tecnológica Nacional**  
**Facultad Regional Rosario**  
**Tecnicatura Universitaria en Programación**



**Programación III**

Unidad 3.2

**Enrutado**

<b>Enrutado en aplicaciones React</b>	<b>3</b>
Error de código 404 o “página no encontrada”	4
Rutas protegidas	5
Ejercicio	7
Rutas anidadas	8
Rutas relativas	9
Rutas dinámicas y navegación con parámetros opcionales	12

## Enrutado en aplicaciones React

Si recordamos en la introducción de este curso, las aplicaciones React son del tipo SPA (*single-page application*) ¿Qué significa esto? Significa que nuestra aplicación no va a recorrer “páginas” diferentes a medida que nos movemos por ella (por ejemplo, yendo de la sección *home* a *contacts*) sino que se va a mover a través de **componentes**, lo que quiere decir, que nuestra app va a ser insertada en un **solo archivo HTML**.

Para lograr este tipo de navegación, se utiliza la librería [react-router](#), que nos permitirá definir rutas (o también conocidas como *paths*) para cada uno de los componentes.

Supongamos que quisiéramos hacer que cuando cargue la aplicación, nos levante el componente Login.js, y, una vez que el usuario inició sesión, lo redirigimos a nuestra página principal en *books champions*.

Bueno, para ello primero debemos instalar la librería, colocando esto en la terminal de nuestro proyecto:

*npm i react-router*

Luego, vamos crear un componente llamado Dashboard, el cuál contendrá todo el código referido a los libros:

1. Los componentes de App que hacían referencia a los libros (NewBook y BookItem)
2. Los manejadores de estados
3. Los estados
4. El arreglo de libros *hardcoded*

Lo primero que haremos es agregar un BrowserRouter a toda la SPA de React, de manera que toda la aplicación escuche a la navegación por rutas. Imaginemos que deseamos crear por ahora solo dos rutas en el lado cliente, una que levante el componente Login (podemos usar la ruta */login*) y una que levante el componente Dashboard (podemos ponerle, */library*).

```
10 const App = () => {
11
12   return (
13     <div className="d-flex flex-column align-items-center">
14       <BrowserRouter>
15         <Routes>
16           <Route path='login' element={<Login />} />
17           <Route path='library' element={<Dashboard />} />
18         </Routes>
19       </BrowserRouter>
20     </div>
21   )
22 }
```

## Error de código 404 o “página no encontrada”

React router internamente ya viene con un manejador de rutas en el caso de que la ruta ingresada no sea ninguna de las declaradas en el código:



Como allí nos especifica la librería, sería una mejor práctica crear nuestro propio componente de error ante una página no encontrada que utilizar ese *default*.

Lo que deseamos realizar entonces es un componente llamado `PageNotFound`, que sea básicamente un `h2` con el texto “¡Oops! La página solicitada no fue encontrada” y un botón de regreso al Login con el texto “Volver a Iniciar sesión”.

Para ello debemos agregar una ruta que apunte a `‘*’` (es decir, que tome todas las rutas posibles que no sean las ya definidas) y nos renderice el componente deseado:

```
36 <Route path='*' element={<NotFound />} />
```

Luego creamos el componente NotFound.

```
1  import { Button } from "react-bootstrap";
2  import { useNavigate } from "react-router";
3
4  const NotFound = () => {
5    const navigate = useNavigate();
6
7    const goBackLoginHandler = () => {
8      navigate("/login");
9    };
10   return (
11     <div className="text-center mt-3">
12       <h2> ¡Ups! La página solicitada no fue encontrada</h2>
13       <Button className="text-center" onClick={goBackLoginHandler}>
14         Volver a iniciar sesión
15       </Button>
16     </div>
17   );
18 };
19
20
21 export default NotFound;
```

Comprobamos su funcionamiento escribiendo cualquier cadena de caracteres luego de / en la barra de navegación.

## Rutas protegidas

Las rutas protegidas (*protected routes*) son aquellas rutas o *paths* que el usuario solo puede acceder mediante determinadas condiciones, sean estas por ejemplo un rol de usuario específico, poseer un permiso específico o al menos, estar logueado en la aplicación.

Aquellas rutas que no están protegidas se la denomina rutas públicas (*public routes*)

Haremos entonces que la ruta */login* sea una ruta pública y */library* una ruta protegida, que no puede ser accedida al menos que el usuario esté logueado en la app.

Primero, crearemos el componente que le va a agregar la protección a la ruta, llamado "Protected":

```
1  import { Navigate } from "react-router";
2
3  const Protected = ({ isSignedIn, children }) => {
4    if (!isSignedIn) {
5      return <Navigate to="/login" replace />;
6    }
7
8    return children;
9  }
10 }
```

Protected recibe por *props* *isSignedIn* y *children*, allí si *isSignedIn* es verdadero devuelve *children* y sino, lo redirige a */login* mediante el componente *Navigate* del react-router.

Luego agregaremos Protected a la ruta deseada:

```
<Route
  path='library'
  element={
    <Protected>
      <Dashboard />
    </Protected>
  } />
```

¿De donde obtenemos el valor *isSignedIn*? Sencillamente agregamos un estado (*loggedIn*) en App, que luego que el usuario hizo log in se convierta en verdadero, de la siguiente manera:

```
13  const [loggedIn, setLoggedIn] = useState(false);
14
15  const handleLogIn = () =>{
16    setLoggedIn(true)
17  }
```

Enviamos esa prop a Login, y luego hacemos que suba el estado si cumple las verificaciones:

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Navigate to='login' />} />
    <Route path='/login' element={<Login onLogin={handleLogin} />} />
    <Route
      path='/library'
      element={
        <Protected isSignedIn={loggedIn}>
          <Dashboard />
        </Protected>
      } />
  </Routes>
</BrowserRouter>
```

Luego, en Login:

```
42      setErrors({ email: false, password: false })
43      alert(`El email ingresado es: ${email} y el password es ${password}`);
44      onLogin()
```

Finalmente, queremos que cuando hace log in la app navegue automáticamente al usuario a `/library`. Para ello, podemos utilizar el *custom hook* de react-router llamado `useNavigate`:

```
16      const navigate = useNavigate();
```

Y utilizarlo dentro del `signInClicked`:

```
46      alert(`El email ingresado es: ${email} y el password es ${password}`);
47      onLogin();
48      navigate("/library");
```

Ahora, comprobaremos el funcionamiento correcto de nuestro código.

## Ejercicio

Necesitamos crear un botón “Cerrar sesión” dentro del Dashboard para que el usuario pueda hacer un sign Out de la aplicación:

1. El botón debe estar posicionado en la esquina superior derecha de la página.
2. Una vez clickeado, el estado de `isLoggedIn` debe volverse falso y la aplicación debe redirigir al usuario a `/login`.

[Como usar](#)

**Book champions app**  
(Quiero ver más)

**Título**


**Autor**

**Puntuación**

**Cantidad de páginas**


**URL de imagen**

☐ "Disponibilidad"
   
[Agregar libro](#)



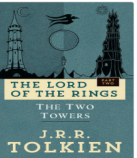
**100 años de soledad**  
Gabriel García Márquez  
★★★★★  
412 páginas

[Eliminar libro](#)
[Seleccionar libro](#)




**Murder on the Orient Express**  
Agatha Christie  
★★★★★  
128 páginas

[Eliminar libro](#)
[Seleccionar libro](#)



**The Lord of the Rings: The Two Towers**  
J.R.R. Tolkien  
★★★★★  
352 páginas

[Eliminar libro](#)
[Seleccionar libro](#)



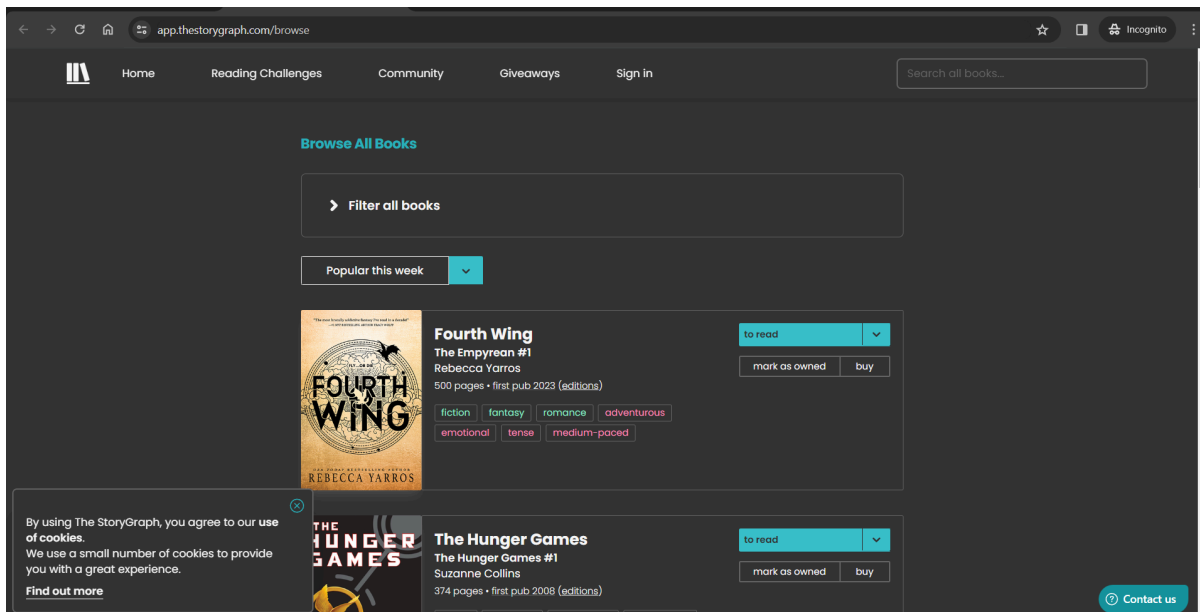
**50 sombras de Grey**  
E.L. James  
★★★★★  
514 páginas

[Eliminar libro](#)
[Seleccionar libro](#)

## Rutas anidadas

Reflexionando sobre las rutas que poseemos, ¿No deberíamos agregarle el componente Protected a la última que creamos? Y ese componente Protected, ¿No debería honestamente estar en todas las rutas por fuera de las páginas públicas?

Esto es un problema similar al caso en que tuviéramos un Layout (pensemos un componente que posea solo una barra superior de navegación y queremos que se repita en todas las páginas de la App)



Independientemente de lo que seleccione el usuario, yo quiero que mi barra de navegación (y la lógica que la acompaña) se mantenga en todas las rutas.



react-router entonces nos proporciona el concepto de **rutas anidadas**, donde a partir de un path (por ejemplo, `/shop`), puedo persistir una lógica y diseño a través de otras rutas hijas de esta (por ejemplo, `/shop/add-product`, `/shop/client`, etc).

Para poder anidar todas las rutas que necesiten protección en nuestra app, lo haremos de la siguiente manera:

```

28 <Route path="/login" element={<Login onLogin={handleLogin} />} />
29 <Route element={<Protected isLoggedIn={loggedIn} />}>
30   <Route
31     path="/library"
32     element={
33       <Dashboard onLogout={handleLogout} />
34     } />
35 </Route>

```

Ahora, todas las rutas dentro de ese componente Route sin path, tendrán los beneficios de ser protegidas.

A su vez, debemos modificar Protected ya que no es *children* la prop que logrará la composición, sino un componente específico de react-router denominada Outlet (sencillamente, reemplazamos *children* por Outlet):

```

1  import { Navigate, Outlet } from "react-router";
2
3  const Protected = ({ isLoggedIn }) => {
4    if (!isLoggedIn) {
5      return <Navigate to="/login" replace />;
6    }
7
8    return <Outlet />;
9  };
10
11
12  export default Protected;

```

## Rutas relativas

Llamaremos rutas relativas (*relative paths*) a aquellas rutas que se conformen de rutas anteriores. ¿A qué nos referimos con esto?

Imaginemos que quisiéramos mejorar nuestras prácticas de composición en el *book-champions* y busquemos no renderizar la lista de libros junto al formulario, sino en rutas separadas: `/library`

para la lista de libros y `/library/add-book` para el formulario.

Para ello, lo primero que vamos a hacer es modificar la ruta original de `library` de la siguiente forma:

```
28 // ...
29
30 <Route element={<Protected isLoggedIn={loggedIn} />}>
31   <Route
32     path="/library/*"
33     element={
34       <Dashboard onLogout={handleLogout} />
35     }>
36   </Route>
37 </Route>
```

De esa forma, el asterisco indica que la totalidad de rutas relativas que empiecen con `/library` van a renderizar primero al componente `Dashboard`. Nuestro siguiente paso es armar un enrutado en el mismo componente `Dashboard`:

```
78 <h2>Book champions app</h2>
79 <p>¡Quiero leer libros!</p>
80 <Routes>
81   <Route index element={<Books books={bookList} onDeleteBook={handleDeleteBook} />} />
82   <Route path="add-book" element={<NewBook onBookAdded={handleBookAdded} />} />
83 </Routes>
```

Allí el `h2` y el `p` **siempre se van a renderizar en las rutas de `/library`** (como si fuera un `Layout`), y las dos rutas relativas quedan conformadas. Esto lo hacemos dentro de `Dashboard` y no en nuestro componente `App` de manera que le podamos pasar las props en este nivel y no tengamos que subir los *handlers* y los *states* a `App`.

Ahora el único problema es que ¡No tenemos forma de acceder al formulario mediante interfaz! Para ello, agregamos al lado de cerrar sesión un botón de Agregar libro que navegue al *path* deseado.

```
50 const handleNavigateAddBook = () => {
51   navigate("/library/add-book", { replace: true });
52 }
53
```

El `replace` lo agregamos ya que sino, cada vez que hagamos click en el botón realizará un agregado al `path`, quedando algo del estilo:

`/library/add-book/library/add-book/library/add-book...`

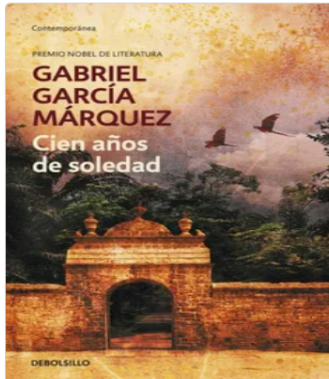
Agregar libro

Cerrar sesión

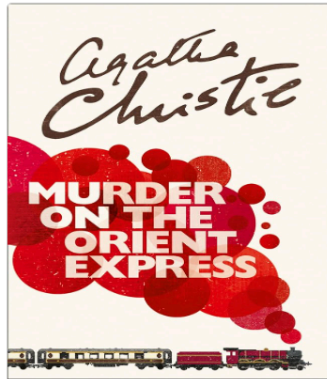
## Book champions app

¡Quiero leer libros!

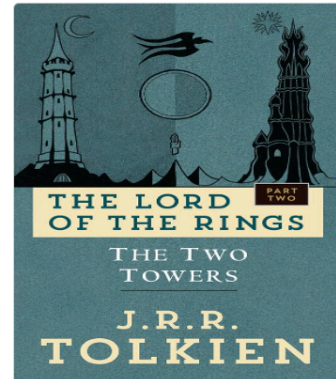
Buscar libro...



Disponible



Disponible



Disponible

Y debemos agregar un botón de volver al formulario para poder volver a `/library`

## Book champions app

¡Quiero leer libros!

Título

Ingresar título

Autor

Ingresar autor

Puntuación

Ingresar cantidad de estrellas

Cantidad de páginas

Ingresar cantidad de páginas

URL de imagen

Ingresar url de imagen

☐ ¿Disponible?

Volver

Agregar lectura

## Rutas dinámicas y navegación con parámetros opcionales

Deseamos ahora, mediante el botón agregado de “Seleccionar libro”, poder mostrar una tarjeta con información extra (la sinopsis por ejemplo) del libro seleccionado. Esta información la vamos a poder acceder navegando hacia el *path* `/:id` (id del libro), es decir, según el libro seleccionado el path será distinto. A las rutas que poseen *path* variables se las denomina **rutas dinámicas**.

Agregamos la ruta a nuestro enrutado dentro de Dashboard, de la siguiente manera:

```
<Routes>
  <Route index element={<Books books={bookList} onDeleteBook={handleDeleteBook} />} />
  <Route path="/:id" element={<BookDetails />} />
  <Route path="add-book" element={<NewBook onBookAdded={handleBookAdded} />} />
</Routes>
```

Donde `:id` representa el valor **dinámico** de la id del libro (el nombre del valor puede ser cualquiera, por ejemplo otra posibilidad podría haber sido `library/:bookId`).

En el antiguo `clickHandler` de `BookItem`, reemplazamos la lógica de libro seleccionado por un `navigate` que nos dirija hacia la nueva ruta:

```

9      const handleClick = () => {
10        navigate(`:${id}`, {
11          state: {
12            book: {
13              title,
14              author,
15              rating,
16              pageCount,
17              summary,
18              imageUrl,
19              available,
20            },
21          },
22        })
23      }
24    }
25  }
```

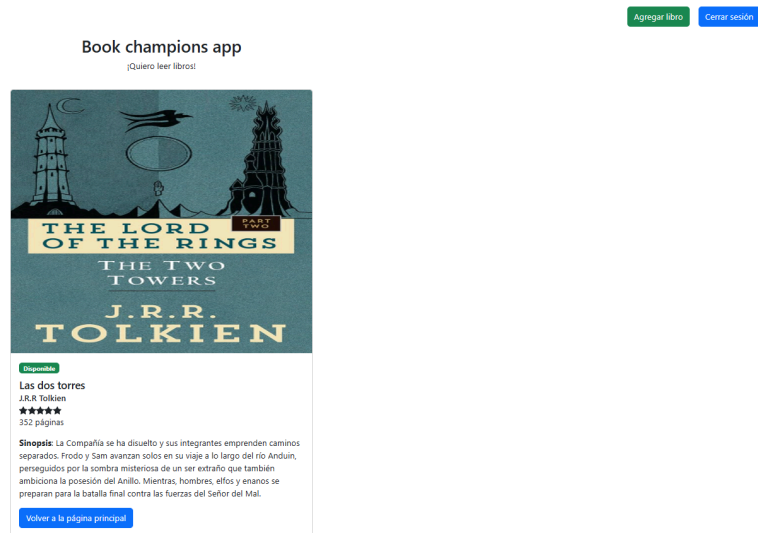
El segundo parámetro se denomina en react-router como un estado “opcional”, con información extra que podemos pasar a la navegación. El formato de estos parámetros tiene que ser

**obligatoriamente** un objeto con la propiedad *state*, que dentro si puede guardar cualquier tipo de valor que deseemos enviar en la navegación (en este caso, la información perteneciente al libro).

Podemos encontrar los *summary* para los libros en el siguiente [link](#). Debemos agregarlos al array original de *books*.

Creamos finalmente el componente [BookDetails](#).

Para poder acceder a ese estado “opcional” que enviamos, react-router nos provee un *custom hook* llamado *useLocation*. Mediante la metodología de *destructuring*, accedemos entonces a las propiedades del libro para poder renderizarlas en el JSX.



<b>Fecha</b>	<b>Versionado actual</b>	<b>Autor</b>	<b>Observaciones</b>
12/05/2023	1.0.0	Gabriel Golzman	Primera versión
12/01/2024	2.0.0	Gabriel Golzman	Segunda versión
16/01/2024	2.1.0	Gabriel Golzman	Agregado de la sección BookDetails
15/12/2024	3.0.0	Gabriel Golzman	Tercera Versión