React intermedio

Rutas, navegación y portales

Navegación por la aplicación

ÍNDICE

- La navegación en una aplicación web
- Usando el historial de navegación
- React router y el historial
- Navegación programática en react router

Navegación en una aplicación web

01	**Enrutamiento** (Switch)	• Mostrar/ocultar rutas • Coincidir con expresión regex
02	**Navegación**	• Mandar parámetros, search y hash • Permitir recuperarlos
03	**Gestión del historial**	• Volver y registro de navegación • Acceso programático

Usando el historial de navegación

- El historial de navegación es accesible mediante la **API HTML5 History** en aquellos navegadores que la implementen
- La API de History proporciona una interfaz para **manipular el historial** de sesión del navegador
- El objeto History **no permite acceder** a las URLs de otras páginas en el historial de la sesión
- React Router **implementa una conexión de su componente con esta interfaz** y actúa como gestor de la misma

Usando el historial de navegación

- Estos son los métodos básicos que implementa la interfaz History:

```
const history = window.history;

// Esto funciona como hacer "atrás" en el navegador
history.back()
// Esto funciona como hacer "avanzar" en el navegador
history.forward();

// Este método avanza o retrocede el número de paginas especificado. En el primer caso
avanza 2, y en el segundo retrocede 2.
history.go(2);
history.go(-2);

// Es posible saber el tamaño de la pila de navegación haciendo un length
const numDeNavegaciones = history.length;
```

Usando el historial de navegación

- Usamos **history.pushState** y **history.replaceState** para generar o modificar entradas en el historial sin provocar cargas o comprobaciones

```
// Cambia la URL y genera nueva entrada en el historial
const stateObj = { miClave: "un valor" };
history.pushState(stateObj, "Un titulo", "pagina2.html");

// Cambia la URL y no genera nueva entrada en el historial
const stateObj = { miClave: "otro valor" };
history.replaceState(stateObj, "Otro titulo", "pagina1.html");
```

Usando el historial de navegación

```
history.pushState(
  // El estado es cualquier objeto JS
  // que pueda pasarse por JSON.stringify
  {
    page: 10,
    size: 50,
    search: [
      'un termino',
      'otro termino'
    ]
  },
  // El título actualmente no se implementa
  "Un titulo",
  //Esta es la ruta a la que navegas
  "pagina2.html"
);
```

- El estado puede ser cualquier objeto **útil en el contexto** de nuestra APP
- **pushState** y **replaceState** son intercambiables pero **pushState** produce nueva entrada
- Cada cambio de estado produce un evento **popstate**
- **window.onpopstate** es un listener global de **popstate**

React router y el historial

- El **BrowserRouter** de react-router implementa la gestión de la API History previamente vista
- La acción de pulsar sobre un componente **Link** o **NavLink** provoca un **history.pushState**
- La acción de usar un componente **Redirect** de react-router provoca un **history.replaceState**
- Estos cambios son gestionados por una librería propia llamada también **history**, que añade la ventaja de ser compatible en navegadores que no implementen la API History de Javascript

React router y el historial

```
history.length // Para hallar el número de entradas en el historial
history.action // La acción que se está llevando a cabo
history.location = { // Un objeto localización compuesto por varias propiedades
  pathname, // El path de la URL, ej: /pagina1.html
  search, // Los parámetros de búsqueda de la URL, ej: ?page=2&size=50
  hash, // Los hashes de la pagina: #ejemplo
  state // Un estado, similar al objeto estado de la API History
}
history.push(path, [state]) // Método similar a pushState, con un path y un nuevo estado
history.replace(path, [state]) // Método similar a replaceState, con un path y un nuevo
estado
history.go(n) // Similar al método de la API
history.goBack() // Equivalente a .go(-1)
history.goForward() // Equivalente a .go(1)
history.block(prompt) // Previene la navegación
```

Navegación programática en react router

- En ocasiones nos interesa navegar sin input del usuario, es decir, sin usar **Link** o **NavLink**
- Esto es posible accediendo programáticamente al **history** generado por react-router
- El método difiere sin embargo dependiendo de si utilizamos **componentes de clase** o **componentes funcionales con hooks**

Navegación programática en react router

- En componentes de clase o funcionales podemos acceder al objeto history desde las propiedades **sólo si este se encuentra renderizado por un BrowserRouter**

```
const Pagina1 = props => {
 const handleAction = () => {
  miServicio().then(response => {
    props.history.push("/pagina2");
  });
 };
 return (
  <>
   <h1>Pagina1</h1>
   <button onClick={handleAction} />
  </>
 );
};
```

```
ReactDOM.render(
  <BrowserRouter>
    <Switch>
      <Route path="/pagina1">
          <Pagina1 />
      </Route>
      <Route path="/pagina2">
          <Pagina2 />
      </Route>
    </Switch>
  </BrowserRouter>,
  document.getElementById("root")
);
```

Navegación programática en react router

- El hook **useHistory** nos permite acceder al objeto history del **BrowserRouter** independientemente de que este renderize directamente al componente, sólo debe de ser un descendiente cualquiera

```
function ComponenteBoton() {
  let history = useHistory();

  function handleClick() {
    history.push('/pagina2');
  }

  return (
    <button type="button" onClick={handleClick}>
      Ir a pagina 2
    </button>
  );
}
```

PARA RESUMIR

- El routing en una aplicación también implica una **gestión de la navegación** y una forma de **recuperar y manipular el historial**
- La API de HTML5 **History** nos permite manipular el historial del navegador sin tener que realizar cargas o recargas reales del contenido
- React router implementa **una versión propia** de la API de History del navegador que además es compatible con otros navegadores desactualizados
- La navegación programática es posible en React router gracias a la **inyección de propiedades** del router y el **uso de hooks**