PRÁCTICA 5.06 Formularios persistentes en `React`

Normas de entrega

- En cuanto al **código**:
  - en la **presentación interna**, importan los **comentarios**, la claridad del código, la significación de los nombres elegidos; todo esto debe permitir considerar al programa como **autodocumentado**. No será necesario explicar que es un `if` un `for` pero sí su funcionalidad. Hay que comentar las cosas destacadas y, sobre todo, las **funciones** y **clases** empleadas. La ausencia de comentarios será penalizada,
  - en la **presentación externa**, importan las leyendas aclaratorias, información en pantalla y avisos de ejecución que favorezcan el uso de la aplicación,
  - si no se especifica lo contrario, la información resultante del programa debe aparecer en la consola del navegador `console.log(información)`,
  - los ejercicios deben realizarse usando ***JavaScript* ES6.** No se podrá utilizar ninguna biblioteca (si no se especifica lo contrario en el enunciado),
  - para el nombre de **variables**, **constantes** y **funciones** se utilizará *lowerCamelCase,*
  - el nombre de los componentes debe comenzar con letra **mayúscula**.
  - todos los formularios en `React` deben ser del tipo **controlados** (a través del **estado**). En caso contrario debe ser debidamente justificado.
- En cuanto a la **entrega** de los archivos que componen los ejercicios:
  - entrega **la práctica** en un sólo proyecto (el nombre a tu discreción),
  - los componentes creados deben estar separados en carpetas (los creados en el Ejercicio1 dentro de una carpeta denominada `Ejercicio1`),
  - el código contendrá ejemplos de ejecución, si procede,
  - comprime la carpeta `src` junto con el fichero `package.json` en un fichero `ZIP` , y
  - sube a **Aules** el fichero comprimido.

Ejercicio 1 - Mi colección de discos (pero esta vez bien)

Parte I. Construye la aplicación (crea un proyecto de `React` nuevo) para gestionar tu colección de discos. Su estructura debe contener, como mínimo, una **cabecera**, un **menú** (con inicio, insertar disco y listar disco), un **contenido** (parte principal) y un **pie** de página (estático). Por supuesto, puedes/debes reutilizar componentes de los que tienes disponibles.

El contenido de cada sección será:

- **inicio**, una página con la presentación de la aplicación,
- **insertar disco**, con un formulario para insertar discos en la aplicación (tras cada inserción debe mostrar un mensaje de confirmación o error),
- **listar disco**, con un listado simple (imagen de portada en pequeño, nombre del disco, nombre del grupo y género) que al pulsar sobre cada uno de ellos aparezca la información completa del disco.

Parte II. En la página de **insertar disco**, prepara un **formulario controlado** diseñado para almacenar los discos con los datos listados a continuación (puedes/debes reutilizar código):

- nombre del disco,
- carátula del disco (su `URL`),
- grupo de música o intérprete,
- año de publicación,
- género de música mínimo cuatro),
- localización que guardará un código (inventado pero que contiene números y letras),
- prestado que almacenará un valor booleano (por defecto será `false`).

Tendrá un botón **Guardar** que añadirá el disco a un listado almacenado en un objeto `JSON` (que puedes/debes reutilizar).

Parte III. Antes de añadir el disco al objeto `JSON` debe ser comprobado (reutiliza código). Crea las **funciones necesarias para validarlo** teniendo en cuenta:

- nombre del disco tiene, al menos, cinco caracteres y es obligatorio,
- grupo de música o intérprete posee, al menos, cinco caracteres y es obligatorio,
- año de publicación dispone de cuatro caracteres numéricos,
- tipo de música comprobará si se ha seleccionado alguno,
- localización tiene el formato **ES-001AA** donde **001** es el número de la estantería y **AA** la balda (combinación de dos letras mayúsculas),
- prestado y carátula no tienen comprobación.

En caso de que se produzca un error en la validación, el campo del formulario implicado será destacado con un estilo `CSS` adecuado. En cuanto ese campo contenga un valor válido, volverá a su estilo original.

Además, existe un `componente` que actúa como **contenedor de información** que, si se ha producido un error, mostrará un mensaje informando de qué campo (o campos) es el incorrecto y cómo solucionarlo (los insultos son opcionales). Ubícalo donde estimes oportuno.

Parte IV. En la sección **listar disco**, añade algún sistema que permita **filtrar** los discos según el texto introducido por el usuario en un `input`. Junto a él existirá otro denominado **Limpiar** que volverá el listado a su contenido original (reutiliza código).

Parte V. Crea un sistema que permita **eliminar un disco** de la colección. Para ello añade un botón para eliminar (un icono puede ser una buen idea) en cada uno de ellos, que permita quitar ese disco del listado. Se debe confirmar la acción de borrado.

Parte VI. Otorga **persistencia** a los datos del listado utilizando `localStorage`. Los datos se cargarán en un objeto `JSON` en la carga de la página y se guardarán cada vez que se modifique el listado (se añada o se elimine un disco). Para ello se deberá:

- cargar los datos desde `localStorage` al inicio de la aplicación y
- actualizar los datos en `localStorage` cada vez que sean modificados (efectos secundarios).