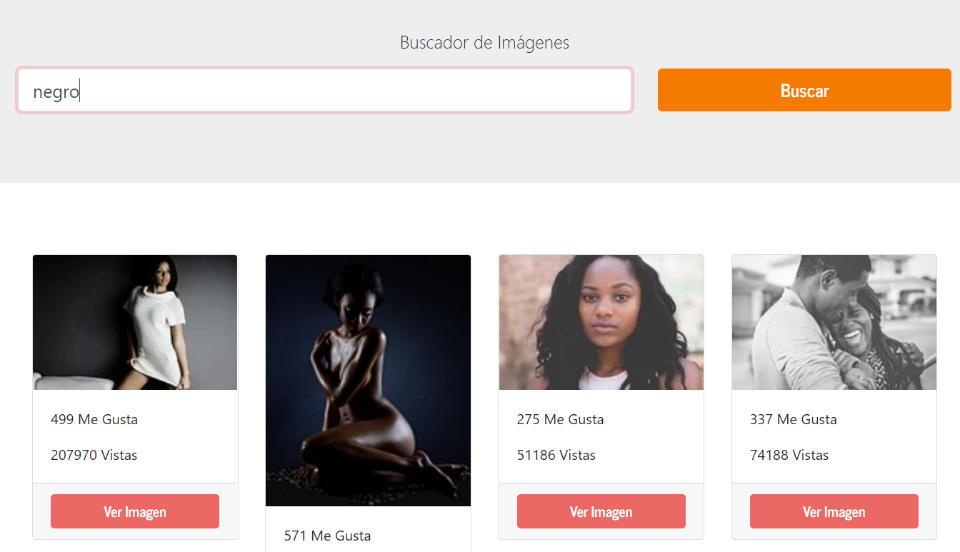
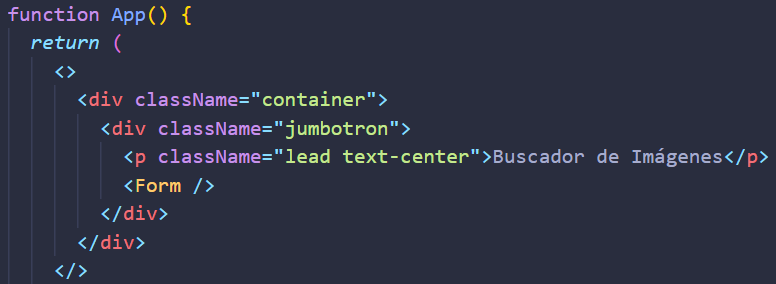
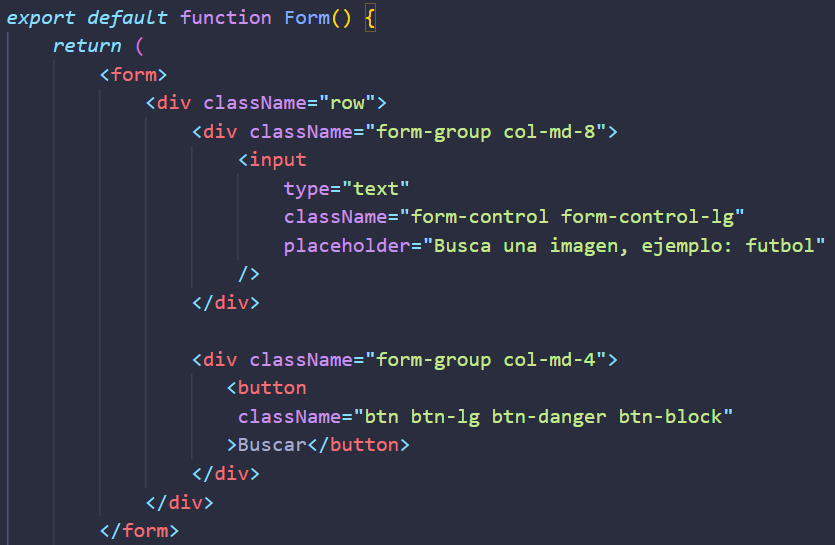
**Proyecto 9**

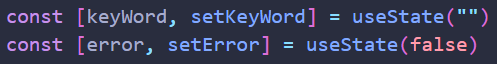
**Resultado final:**

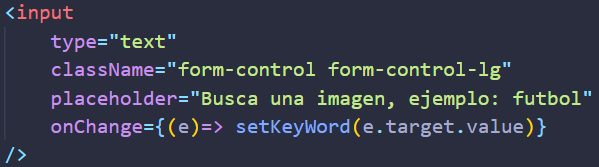
**1**. **Eliminamos** los **archivos** que **no usamos** e **incluimos** en el <**head**> **Bootstrap**.

**2**. Dentro de **App.js** imprimimos el **título de la página** y el **formulario de búsqueda**.

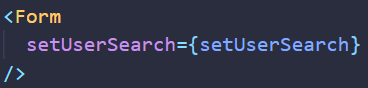
**3**. **Creamos** el **Form.js** que va consistir de un input de texto y un botón.

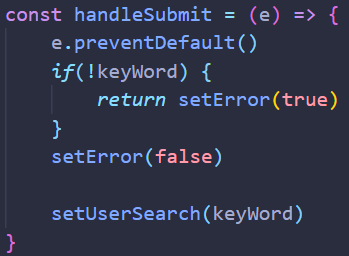
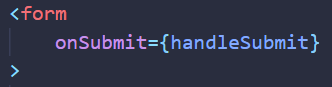
**4**. **Validando la búsqueda**: dentro de **Form.js** **creamos dos estados**, uno para **almacenar** lo que el **usuario** vaya **escribiendo** y **otro** para **detectar** si hay un **error**. En el evento **onChange** del **input** le **cambiamos** el **valor** al **estado**.

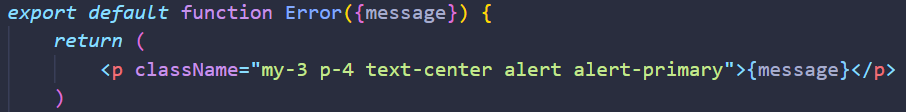




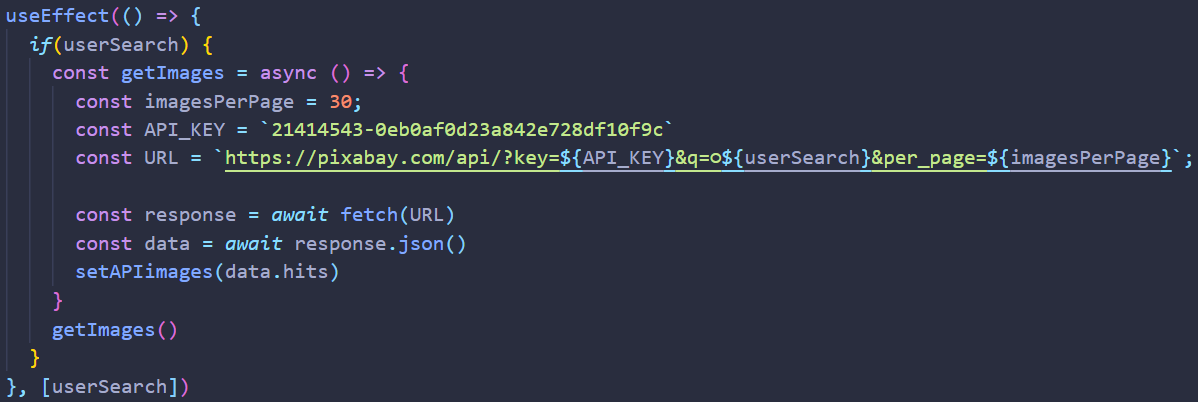
Dentro de **App.js** **creamos** un **estado** para **guardar** los **datos validados** y se lo **pasamos** a **Form.js.**

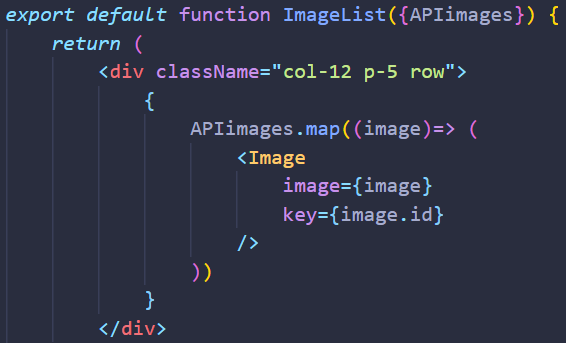


En **Form.js** **creamos** la **función** que va a **validar** los **datos** **en el submit** y se la **pasamos** al <**form**> bajo el evento **onSubmit**. Allí nos aseguramos de que el campo no este vacío y en caso de **no** **haber errores** **guardamos** los **valores validados** en el **estado** que nos llega del App.js.

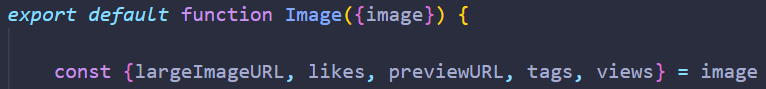
**5**. **Mostrando los errores**: **creamos** un **componente** de **error** y lo **imprimimos** en el **Form.js** en **caso** de que su **estado** sea **true** (es decir, en caso de que haya un error).

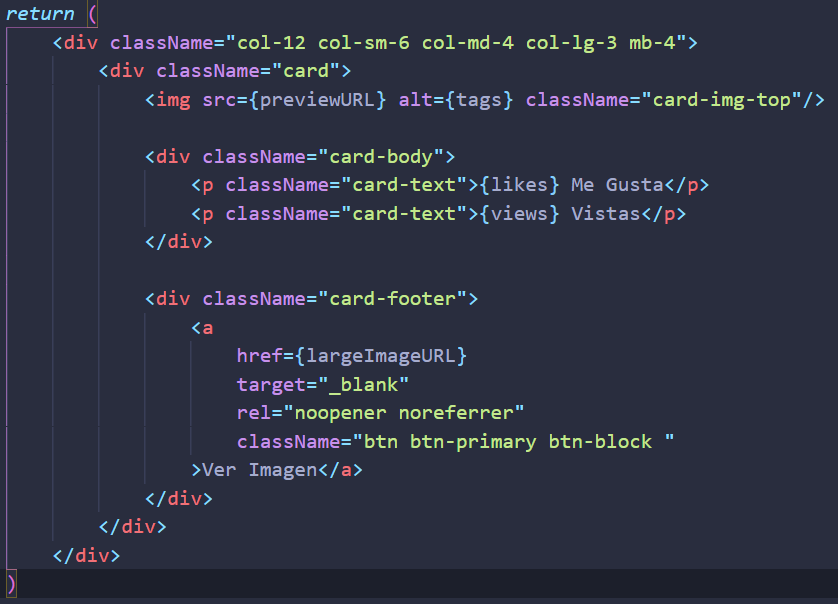
**6**. **Obteniendo imágenes de la API**: **creamos** un nuevo **estado** para **guardar** la **información** que nos llega desde la **API**. Hacemos un **useEffect** que toma como **dependencia** la frase de **búsqueda validada**. **Si hay frase de búsqueda** le **pedimos la información a la API** y la **guardamos en el estado**. Nótese que le estamos pidiendo que nos traiga únicamente 30 imágenes.



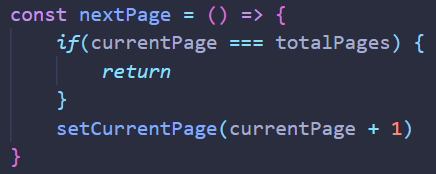
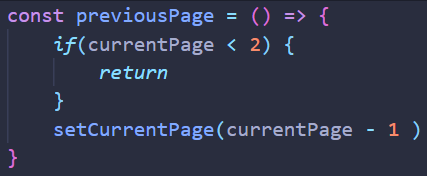
**7**. **Mostrando las imágenes**: **creamos** el componente **ImageList.js** que va a **recibir** un **array** con las **30 imágenes** de la **API**. Allí **hacemos** un **map** del array e **renderizamos** en **cada iteración** el **componente** **Image.js**.

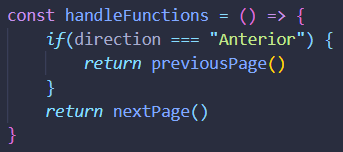
**Creamos** el componente **Image.js** y dentro del mismo **descomprimimos** **ciertas propiedades** **del objeto** y mediante clases de Bootstrap creamos una tarjeta con la información.

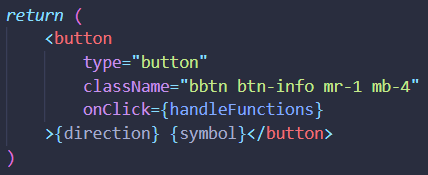




**8**. **Creando la paginación**: **creamos** un nuevo componente llamado **Pagination.js**, que va a **recibir** como **props** una **dirección** (siguiente/anterior), un **símbolo**, una **página actual**, el **actualizador** de estado de la **página actual** y el **total** de **páginas**.

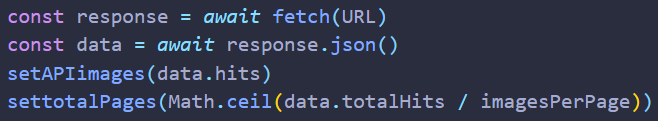
**Creamos** las **funciones** para **aumentar** **y** **disminuir** **el número de página actual**. En caso de que la página actual sea igual al total de páginas evitamos que avance y en caso de que la página actual sea 1 evitamos que siga descendiendo.

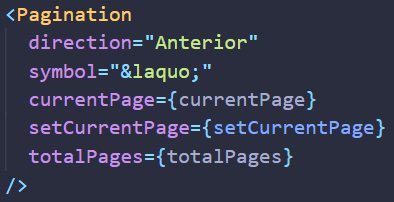
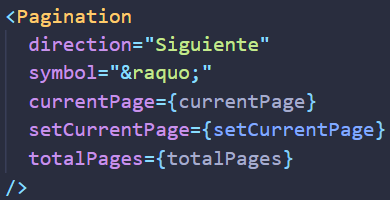
Luego **creamos** una **función** que va a **terminar** **cuál de las dos funciones se va a ejecutar**.

El **componente** en si simplemente se va a **formar** por un **<button>** que en el evento **onClick** va a ejecutar la función que decide cuál de las dos funciones debe ejecutarse.

Dentro de **App.js** **creamos** **dos nuevos estados**, **uno** para **almacenar** el **total de páginas** que **vamos a tener** (**teniendo** en **cuenta** la **cantidad** de **imágenes** que **nos trae la API y** la **cantidad** de **imágenes** que **mostramos** **por página)**, y **otro** para **guardar** la **página** **actual** del usuario. Dentro del **UseEffect**, luego de recibir la información, **definimos** el **total de páginas** **dividiendo** **los resultados por el total de imágenes que queremos mostrar** (haciendo redondeo para arriba.)



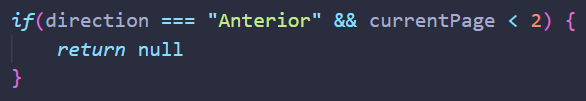


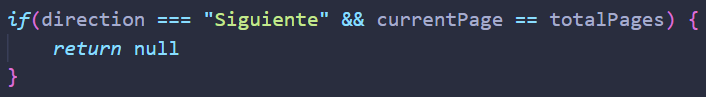
**Finalmente** **renderizamos** **el componente dos veces dentro de App.js**

**9**. **Evitar botón de siguiente y anterior**: para evitar que el botón de anterior aparezca en la página 1 o el botón de siguiente en la última página o que ambos botones se renderizen al cargar la página debemos: **dentro** del componente **Pagination.js hacer un if**.

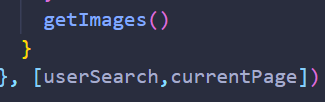
- **Botón de anterior**: si es la página actual es menor a 2 que no se renderize.

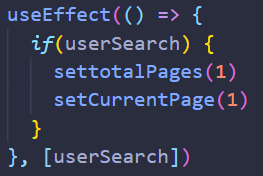
- **Botón de siguiente**: si la página actual es igual al número total de páginas que no se renderize.

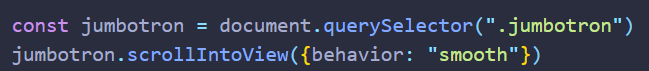




**10**. **Haciendo funcionar la paginación**: a la **URL** de la **API** debemos **pasarle** el **parámetro** “**page**” con el **valor** de la **página actual**, de esa forma **nos traerá** las **próximas** **30 imágenes** en caso de que las tenga. **No** hay que **olvidar** **agregar** la **página actual** como **dependencia** del **useEffect** para que haga el pedido cada vez que cambiamos de página.



**11**. **Limpiando para nueva búsqueda**: debemos **limpiar** la **página actual** y el **total de páginas** para que cada vez que hagamos una **nueva búsqueda se reinicie**, para esto **dentro** de **App.js** dentro de un **nuevo useEffect** **cada vez que el usuario busque algo nuevo volvemos a 1.**

**12**. **Efecto de Scroll cada vez que cambiamos de página**: para esto dentro del useEffect de App.js seleccionamos el <div> con la clase “jumbotron” y hacemos un scroll up hasta él.