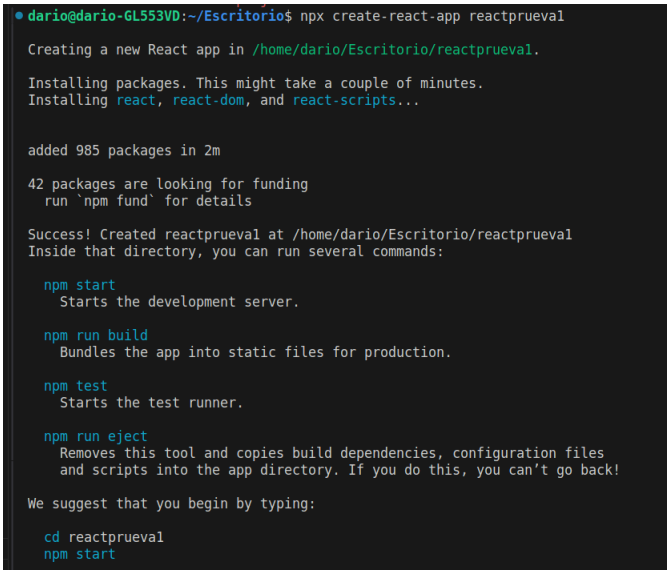
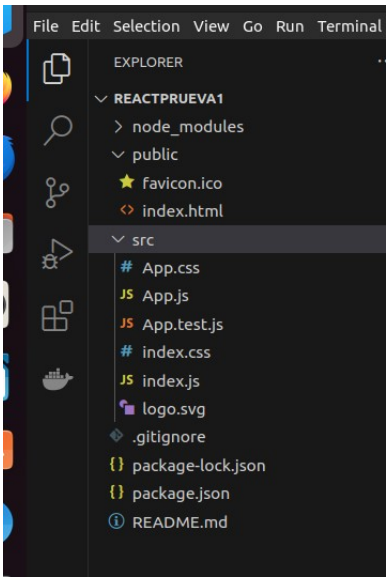
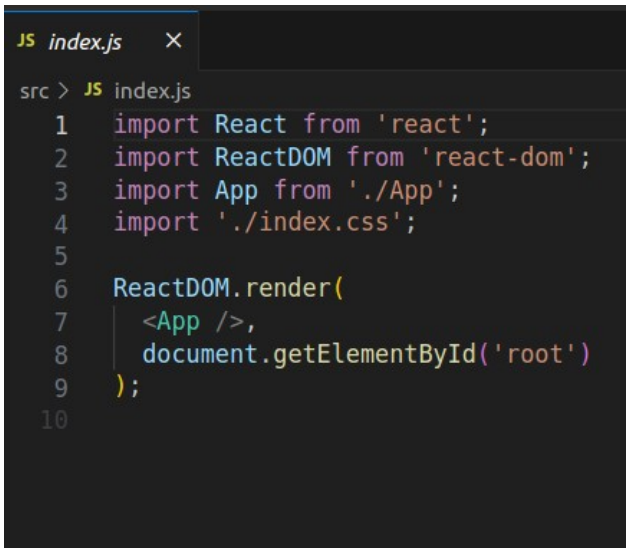
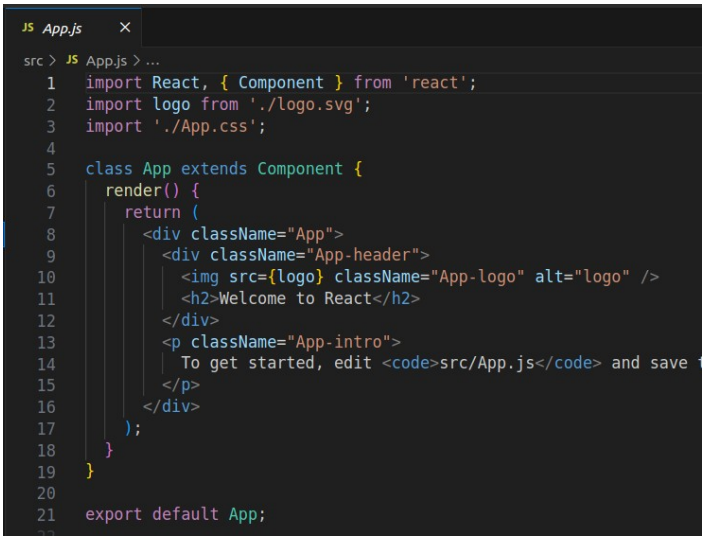
1-Creación del proyecto: abrimos la terminal o consola y tecleamos el siguiente comando:
npx create-react-app <nombre del proyecto>.





Este comando creará un directorio con la estructura necesaria para comenzar a trabajar. Todavía sin hacer modificaciones y solo mirando la estructura y el código por defecto que nos proporciona React, podemos observar que además de node_modules, los archivos package.json y los relacionados con Git, React estructura sus archivos en una carpeta llamada src, donde estarán los componentes de la aplicación, cuyo punto de entrada es el archivo index.js.
Además, disponemos del primer componente (también llamado raíz), App.js. este componente es una función con el mismo nombre del archivo que devuelve un elemento "div", en el cual disponemos de un encabezado con un texto y una imagen. Esta función, exportada con el mismo nombre, es importada en el archivo index.js , es decir, el componente exportado se ubica en una etiqueta "<App />", que será renderizado por index.js.



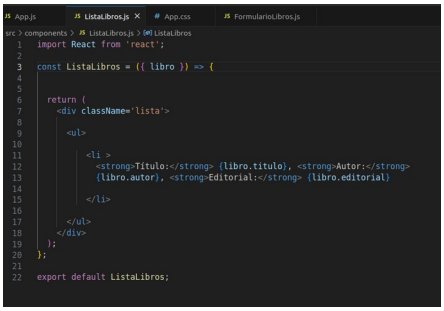


Entonces, cuando se ejecuta ReactDOM.render(), el componente <App /> se renderiza dentro del documento index.html (que se encuentra dentro del directorio 'public' con ID "root")mediante document.getElementById('root').Para compilar y lanzar la aplicación utilizaremos el comando npm start, y asi podremos comprobar el funcionamiento de la aplicación en el navegador.

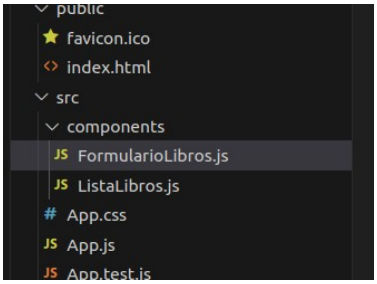
*Resultado del código por defecto de Ract

2.Una vez creado el proyecto, lo abrimos en Visual Studio Code y creamos en el directorio src un nuevo directorio llamado component; y,dentro del mismo, un archivo FormularioLibros.js , un formulario que recoge datos sobre un libro y otro llamado ListaLibros.js, que devolverá una lista con los libros existentes y los añadidos a través del formulario cada vez que se actualice componente , la actualización del componente se realiza cuando se envía el formulario. Utilizaremos componente de tipo función, es decir, los componentes serán devueltos por una función que es llamada por la etiqueta con el mismo nombre de la función que devuelve el componente, en esta etiqueta el componente raíz pasará a los componentes hijos la información necesaria para ejecutar su lógica, (props). También eliminaremos aquellos archivos que no utilizaremos, como logo.svg e index.css.

*FormularioLista, sin la logica todavia

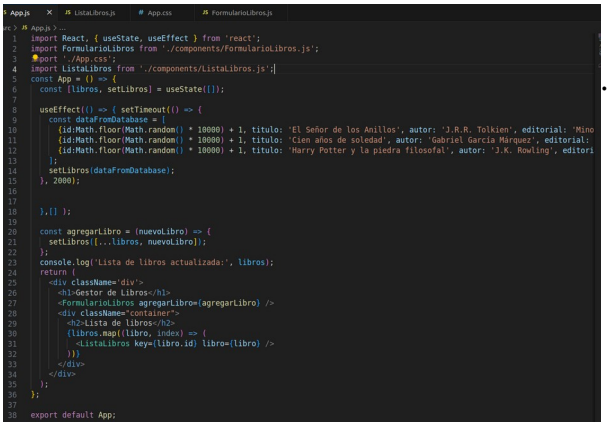


*ListaLibros, tabla con sintaxis JS entre{} se explicacion en apartado 6



*Estructura

3-Para poder renderizar nuestros componentes necesitaremos modificar el código que React nos proporciona por defecto



*app.js modificado,

Primero realizamos las importaciones de módulos y componentes: import React, { useState, useEffect } from 'react';import FormularioLibros from './components/FormularioLibros.js'; import ListaLibros from './components/ListaLibros.js';

const [libros, setLibros] = useState([]);Utiliza el hook useState para declarar el estado libros y la función setLibros que permite actualizar dicho estado. Inicialmente, libros se establece como un array vacío [].
useEffect ejecuta el código que le pasamos una vez cuando se inicia el componente y luego cada vez que se produce una modificación en uno de los valores del estado o de las props.
En este ejemplo se utiliza para simular una carga asíncrona desde una base de datos ficticia. Se emplea setTimeout() para esperar 2 segundos y luego se establece el estado de libros con los datos de la base de datos ficticia, setLibros(dataFromDatabase). En este caso la carga de 'libros' se realiza solo una vez cuando el componente se monta, ya que pasamos un array vacío como segundo argumento. Si pasamos variables dentro de este array, el efecto se ejecutará cada vez que una de esas variables cambie. Por ejemplo, si pasamos [libros], el efecto se ejecutaría cada vez que el estado libros cambie, esto seria deseable si tuviésemos un back-end que guarde los cambios en la base de datos, pero no la tenemos, entonces volverá a renderizar el array que simula la base de datos que en realidad permanece con los mismos datos del principio y cada vez sera entendido como un cambio por lo que caeríamos en bucle constante de renderizado del efecto. Por eso en este caso, queremos que la carga de datos ocurra solo una vez, por lo que usamos un array vacío para indicar que no hay lo que se llaman dependencias, las dependencias son los datos que pasamos dentro de dicho array.
Para generar ID únicos utilizaremos el método Math.floor(Math.random() * 10000) + 1; tanto aquí como en la lógica para el envío de formulario.
const agregarLibro = (nuevoLibro) => {….} , luego de useEfect encontramos esta función que recibe un nuevo libro como argumento, . Lo que hace es utilizar el estado de 'libros' y agregar el nuevo libro al final de la lista utilizando el operador spread (...libros, nuevoLibro) y activando setNuevoLibro() con la data del nuevo libro recibida desde LibroFormulario cuya lógica se explicará mas adelante
La parte de retorno return {..} del componente JSX contiene la estructura del componente en sí. El componente tiene una , que encapsula todo el contenido, hay un encabezado <h1> y se devuelven los componentes FormularioLibros (al que se le pasa la función agregarLibro como una prop llamada agregarLibro) y otra que contiene un encabezado <h2> y realiza un mapeo (map()) sobre la lista de libros (libros) , y para cada libro se renderiza un componente ListaLibros. Se le pasa como prop el 'libro' actual y el atributo key que se utiliza para ayudar a React a identificar elementos , cada iteración de la función map()=>{..} nos devuelve un componente ListaLibros con la información del objeto que se encuentre en el array en cada momento.

*Resultado preliminar

4-Añadimos CSS:
body: Esta regla se aplica al elemento <body> de tu documento HTML. Aquí se establecemos varias propiedades:

margin: 0px;: Establece el margen exterior del cuerpo del documento en 0 píxeles, eliminando así cualquier margen predeterminado que podría aplicarse por defecto.

font-family: sans-serif;: Define la fuente que se usará para el texto del cuerpo del documento. .

background-color:rgb(255, 69, 0);: Establece el color de fondo del cuerpo del documento en un tono de rojo anaranjado.

border: 4px solid;: Agrega un borde sólido alrededor del cuerpo del documento, con un ancho de 4 píxeles.

min-height: 100vh;: Establece la altura mínima del cuerpo del documento en el 100% de la altura visible del viewport.

padding-top: 20px;: Establece un relleno superior de 20 píxeles en el cuerpo del documento.

span: Esta regla se aplica a todos los elementos del documento y establece el color del texto en un tono de rojo oscuro (crimson).

.lista: Esta regla se aplica a elementos con la clase "lista" y establece el color de fondo en un tono de verde claro con un cierto grado de transparencia.

#root: Esta regla se aplica al elemento con el ID "root" y establece varias propiedades:

display: flex;: Hace que el elemento "root" utilice el modelo de diseño flexible de CSS.

flex-direction: row;: Establece la dirección principal del contenedor flex como fila, lo que significa que sus elementos secundarios se colocarán uno al lado del otro horizontalmente.

justify-content: center;: Centra los elementos secundarios horizontalmente dentro del contenedor flexible.

h1, h2: Esta regla se aplica a todos los elementos <h1> y <h2> del documento y los centra horizontalmente mediante la propiedad text-align.

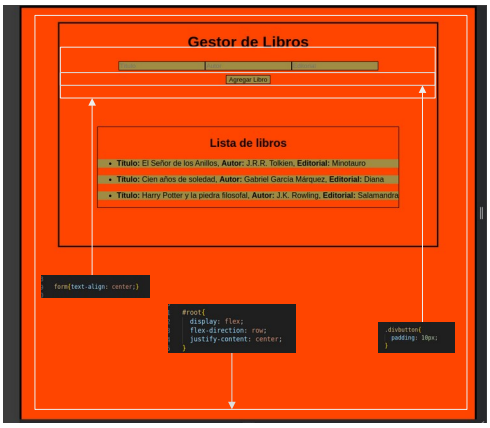
form: Esta regla se aplica a todos los elementos <form> del documento y los centra horizontalmente mediante la propiedad text-align.

.divbutton: Esta regla se aplica a elementos con la clase "divbutton" y establece un margen de 10 píxeles alrededor del contenido, no es visible.

.div: Esta regla se aplica a elementos con la clase "div" y establece un borde sólido de 3 píxeles y un margen exterior de 10 píxeles.

.container: Esta regla se aplica a elementos con la clase "container" y establece un margen exterior de 80 píxeles y un borde sólido de 1 píxel alrededor del contenido.

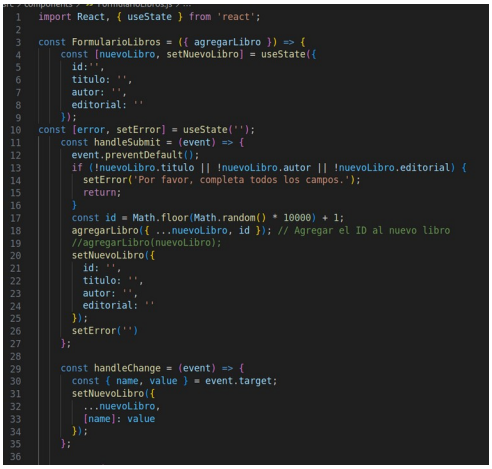
input, button: Esta regla se aplica a todos los elementos <input> y <button> del documento y establece el color de fondo y los estilos del borde para estos elementos. El color de fondo es un tono de verde claro con una cierta transparencia, y los estilos del borde están configurados para eliminar los estilos predeterminados y establecer un borde sólido de 1 píxel.



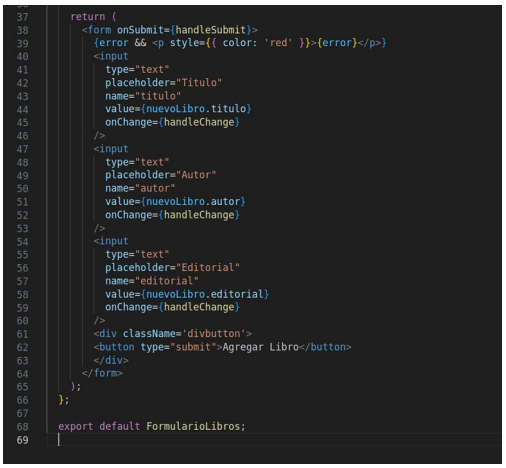
*Estilos no visibles pero necesarios para el centrado y encajes de elementos

*Css

5-Lo siguiente es añadir la lógica en LibroFormulario para que este componente pueda pasar la información al componente raiz.



```
import React, { useState } from 'react';

const FormularioLibros = ({ agregarLibro }) => {
    const [nuevoLibro, setNuevoLibro] = useState({
      id: '',
      titulo: '',
      autor: '',
      editorial: ''
    });
const [error, setError] = useState('');
    const handleSubmit = (event) => {
      event.preventDefault();
      if (!nuevoLibro.titulo || !nuevoLibro.autor || !nuevoLibro.editorial) {
        setError('Por favor, completa todos los campos.');
        return;
      }
      const id = Math.floor(Math.random() * 10000) + 1;
      agregarLibro({ ...nuevoLibro, id }); // Agregar el ID al nuevo libro
      //agregarLibro(nuevoLibro);
      setNuevoLibro({
        id: '',
        titulo: '',
        autor: '',
        editorial: ''
      });
      setError('')
    };

    const handleChange = (event) => {
      const { name, value } = event.target;
      setNuevoLibro({
        ...nuevoLibro,
        [name]: value
      });
    };

```



```
  return (
    <form onSubmit={handleSubmit}>
      {error && <p style={{ color: 'red' }}>{error}</p>}
      <input
        type="text"
        placeholder="Titulo"
        name="titulo"
        value={nuevoLibro.titulo}
        onChange={handleChange}
      />
      <input
        type="text"
        placeholder="Autor"
        name="autor"
        value={nuevoLibro.autor}
        onChange={handleChange}
      />
      <input
        type="text"
        placeholder="Editorial"
        name="editorial"
        value={nuevoLibro.editorial}
        onChange={handleChange}
      />
      <div className='divbutton'>
      <button type="submit">Agregar Libro</button>
      </div>
    </form>
  );
};

export default FormularioLibros;

```

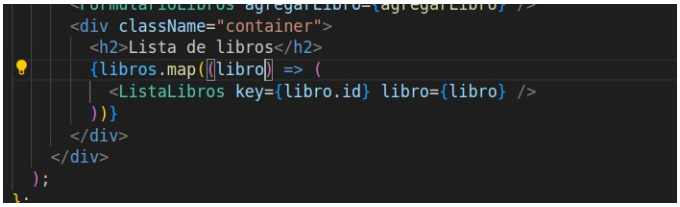
Const FormularioLibros = ({ agregarLibro }) => {: Aquí se define un componente funcional llamado FormularioLibros. Este componente recibe una única prop llamada agregarLibro que es la función que se utilizará para agregar un nuevo libro a la lista, como explicamos en el apartado 4.
const [nuevoLibro, setNuevoLibro] = useState({ ... });: Aquí se utiliza el hook useState para inicializar el estado del componente. nuevoLibro es el estado actual que contiene los detalles del nuevo libro y setNuevoLibro es una función que se utiliza para actualizar el estado nuevoLibro. El estado inicial se establece como un objeto con las propiedades id, titulo, autor y editorial, todas ellas inicializadas como cadenas vacías.
const [error, setError] = useState("");: Se declara otro estado utilizando el hook useState. ‘error’ se utiliza para almacenar mensajes de error relacionados con el formulario de entrada. Su estado original sera de cadena vacía , pero si las condiciones if (!nuevoLibro.titulo || !nuevoLibro.autor || ! nuevoLibro.editorial) no se cumplen, setError modificará el estado pro el mensaje 'Por favor, completa todos los campos.' Este bloque se ejecutará dentro de handleSubmit.
const handleSubmit = (event) => { ... }: Se define la función handleSubmit que se ejecuta cuando el formulario se envía. Primero, se llama al método preventDefault() del evento para evitar que el formulario se envíe automáticamente. Luego, se verifica si los campos titulo, autor y editorial del nuevo libro están vacíos. Si alguno de ellos está vacío, se establece un mensaje de error, como explicamos antes y se detiene la ejecución. Si los campos están completos, se genera un ID aleatorio para el nuevo libro, se llama a la función agregarLibro pasando el nuevo libro con el ID generado y se reinicia el estado del nuevoLibro y de ‘error’

const handleChange = (event) => { ... }: Se define una función llamada handleChange que toma un objeto de evento como parámetro. Esta función se utilizará como un controlador de eventos para los cambios en los campos de entrada del formulario. Es decir, cuando el usuario escriba en los campos a rellenar,(inputs), estos son eventos que suponen cambios cuyas propiedades y valores son registrados en el objeto event.target

const { name, value } = event.target;: Dentro de la función handleChange, se utiliza la desestructuración para extraer dos propiedades del objeto target del evento: name y value. El target del evento representa el elemento DOM en el que se originó el evento, que en este caso sería el campo de entrada que ha cambiado, (titulo, autor, editorial).
name: Contiene el atributo name del campo de entrada que cambió. Este atributo se utiliza para identificar qué propiedad del estado nuevoLibro debe actualizarse,, (titulo, autor, editorial).

value: Contiene el nuevo valor del campo de entrada,(lo que el usuario ingrese en el input).
setNuevoLibro({ ...nuevoLibro, [name]: value });: Se utiliza la función setNuevoLibro para actualizar el estado nuevoLibro del componente. Se utiliza el operador spread (...) para copiar todas las propiedades existentes de nuevoLibro en un nuevo objeto. Luego, se utiliza la sintaxis de los corchetes ([name]) para establecer a propiedad del objeto nuevoLibro que debe actualizarse, donde name es el nombre del campo de entrada que ha cambiado y value es el nuevo valor del campo. Finalmente, el componente retorna el JSX que representa el formulario. Se establece el onSubmit del formulario para que llame a la función handleSubmit cuando se envíe el formulario. Los tres campos de entrada (Título, Autor y Editorial) están vinculados al estado nuevoLibro y llaman a la función handleChange cuando cambian. El botón "Agregar Libro" envía el formulario y desencadena la función handleSubmit.
{error && <p style={{ color: 'red' }}>{error} }: Esta linea de código dentro del retorno del JSX evalúa mediante el operador && si la variable 'error' es verdadera, osea si contiene algo o esta vacía, en caso de que sea verdadera sera porque contiene el mensaje de error por lo que se renderizará el con el mensaje de error, contiene un estilo 'in-line' que hará que se muestre en color rojo.

6- En el componente ListaLibros, la función recogerá una prop (propiedad) que será un objeto llamado 'libro', esta será enviada por el componente raíz mediante el map(), y utilizando {} y dentro de ellas sintaxis JS, podremos mostrar los valores del objeto 'libro'. Todo esto se mostrará dentro de las etiquetas html de listado de elementos, <ul> .
Una vez con la lógica del formulario completa, cada vez que se envíe un formulario la función const agregarLibro = (nuevoLibro) => { setLibros([...libros, nuevoLibro]) actualizará la array 'libros' en el componente raiz por lo que React hara un nuevo render pero con la nueva data en 'libros' y como explicamos en el apartado 3, es pasada como prop al componente ListaLibros, por lo que como resultado veremos un nuevo libro en nuestra lista.



*Mapeo de 'libro' , pasado como 'prop'

*Sintaxis JS entre{}

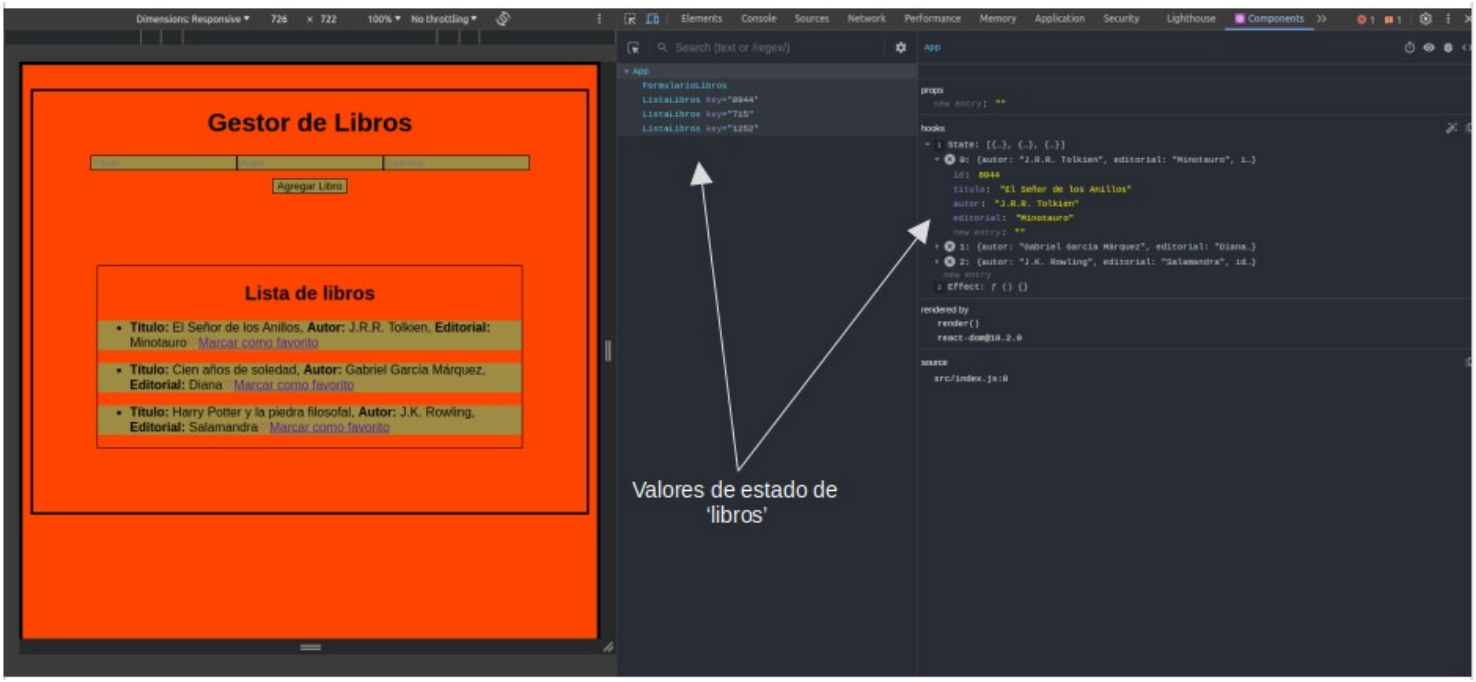
Añadiremos una funcionalidad mas a 'ListaLibros' , para que el usuario pueda marcar libros como favorito.
Se inicializa el estado isFavorite como 'false' usando el hook useState. isFavorite es un booleano que representa si el libro ha sido marcado como favorito o no. El valor de isFavorite se invierte con cada click en el 'anchor' tag, (enlace) que, en su estado inicial, ('false'), se representa con el texto 'Marcar como favorito', y entonces con un click la función handleIsFavorite que esta asociada a él cambia el estado de isFavorite a 'true',entonce se mostrará un corazón relleno junto al texto "Desmarcar como favorito" y de nuevo al hacer clic en el enlace, se invoca nuevamente la función handleIsFavorite y volveremos a la situación inicial.

```jsx
import React, { useState } from 'react';

const ListaLibros = ({ libro }) => {
  const [isFavorite, setIsFavorite] = useState(false);

  const handleIsFavorite = () => {
    setIsFavorite(!isFavorite);
  };

  return (
    <div className='lista'>

      <ul>

        <li >
          <strong>Título:</strong> {libro.titulo}, <strong>Autor:</strong>{' '}
          {libro.autor}, <strong>Editorial:</strong> {libro.editorial}
          {isFavorite ? <span>&#9829;</span> : <span>&#9825;</span>}
          {isFavorite ?
            <a onClick={handleIsFavorite} href='#'>Desmarcar como favorito</a>
          :
            <a onClick={handleIsFavorite} href='#'>Marcar como favorito</a>
          }
        </li>

      </ul>
    </div>
  );
};

export default ListaLibros;
```

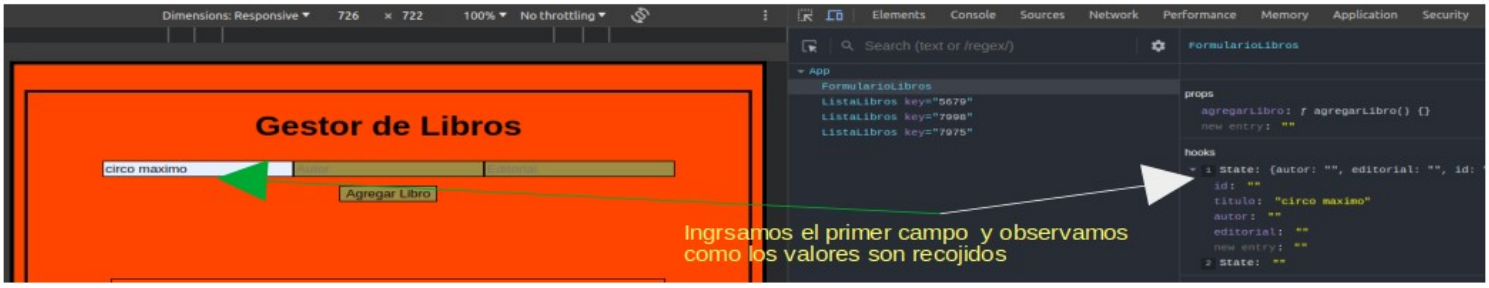
*Nueva funcionalidad añadida en ‘ListaLibros’

*Añadimos el código css correspondiente a

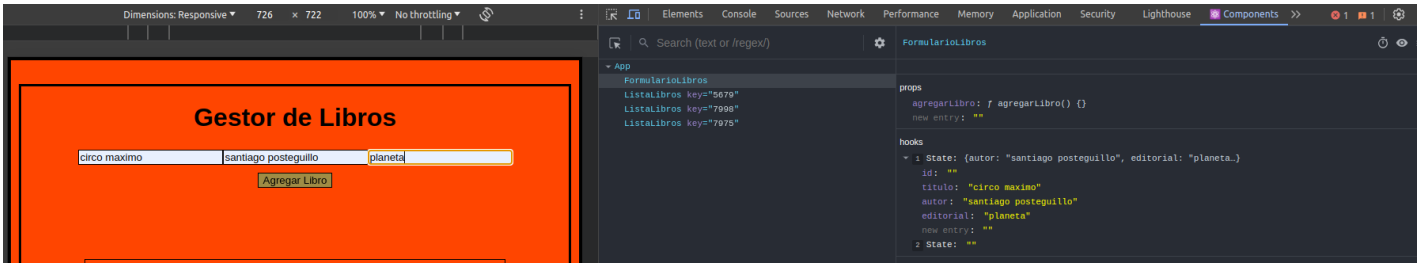
7 y 8- Compruebación de valores en las React Dev Tools y el funcionamiento en el navegador.



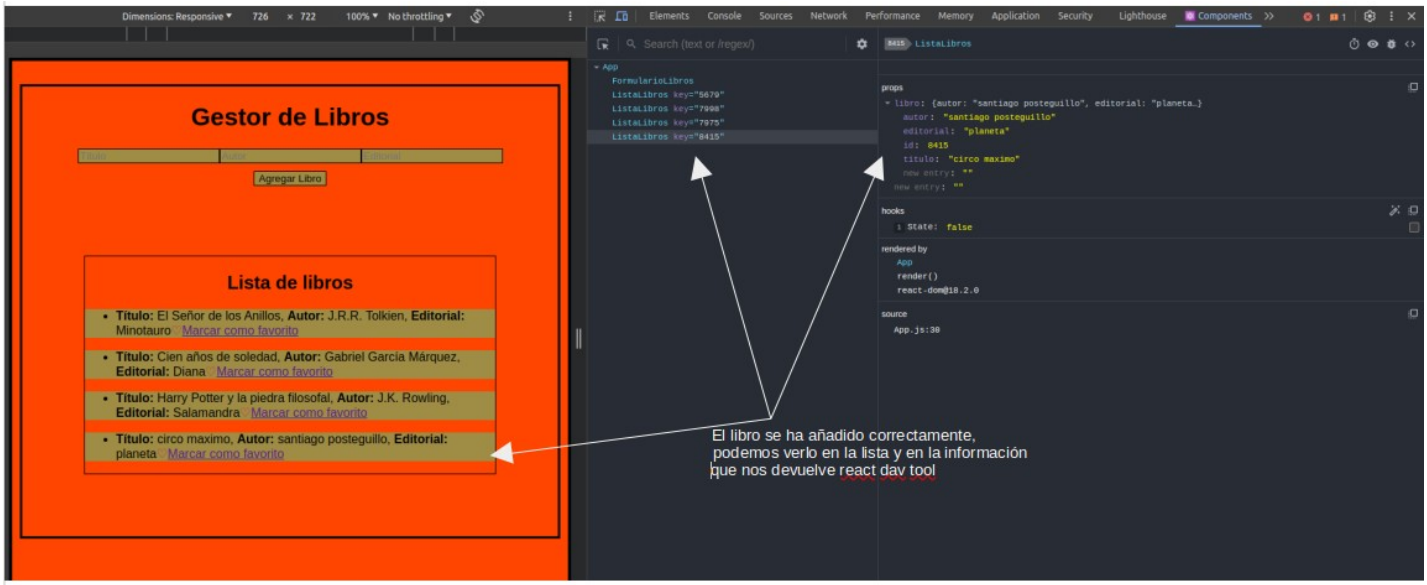
*Lanzamos la aplicación en el localhost, ingresamos en chrome devTools y hacemos click en la pestaña ‘components’ que nos proporciona React Developer Tool, observamos a la derecha la ventana que nos muestra los valores iniciales del estado de nuestra aplicación.



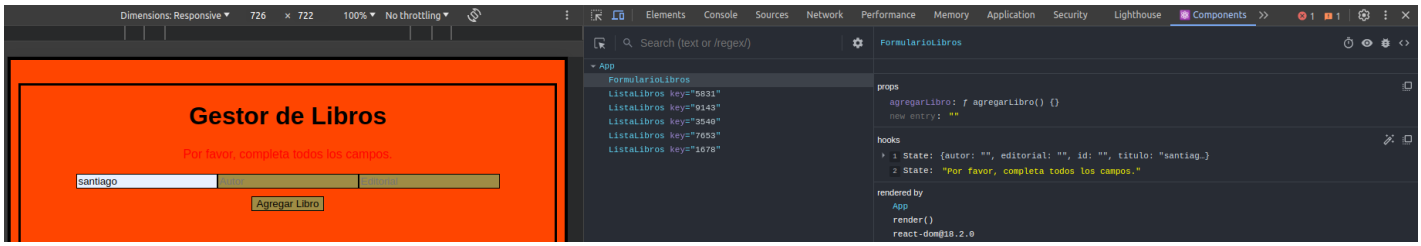
*Comenzamos a rellenar el formulario y podemos observar como la funcion ‘handleChange’ recoje lo que escribimos.



*Formulario al completo y enviamos

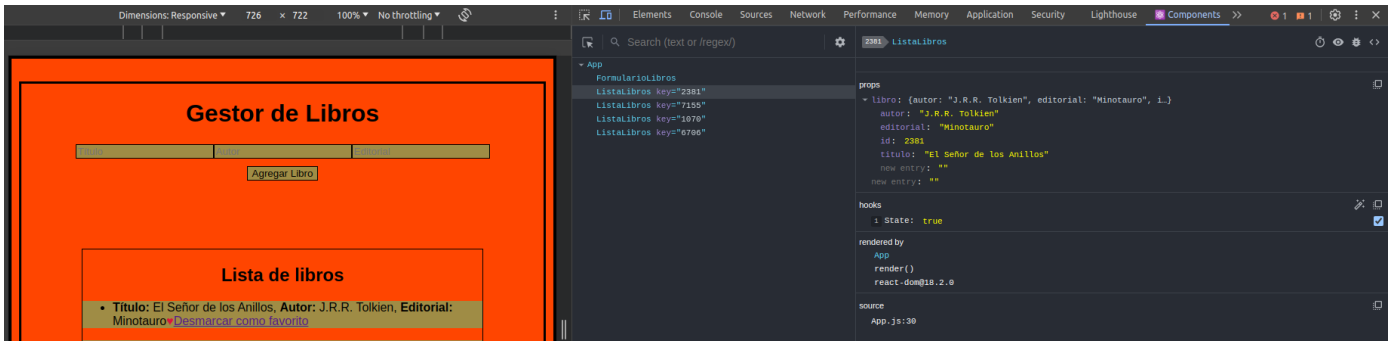


*Aquí se puede observar que el nuevo libro se ha añadido

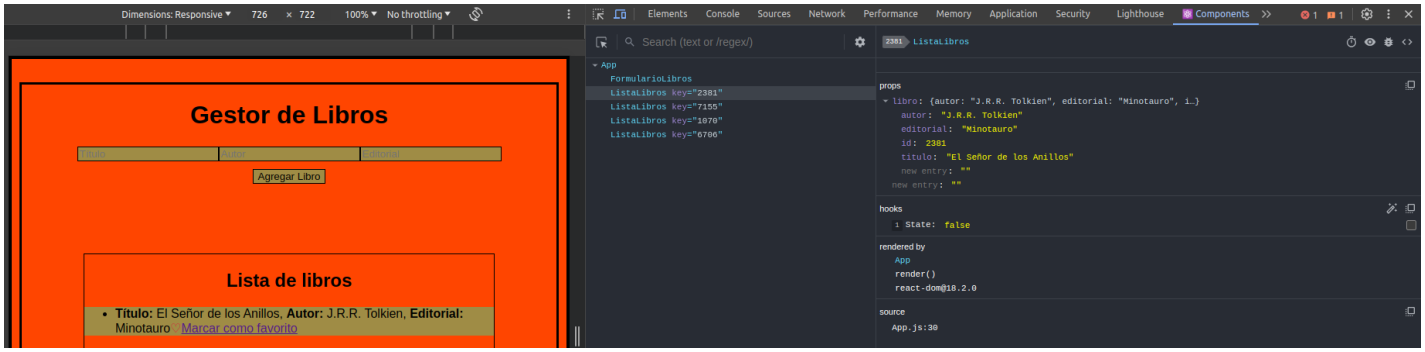


*Intentamos enviar sin tener los campos completos, el mensaje en rojo aparece, ‘por favor complete todos los campos’ y a la derecha el mismo mensaje en el aparatado ‘hook’ de Ract Dev Tools

*Detalle del mensaje que verá el usuario



*Marcaje como favorito, vemos el corazon en rojo al lado del texto ‘Desmarcar……’ y en la interfase de React Dev Tools podemos observar debajo de ‘hooks’ ‘State: true’



*Desmarcamos el favorito, el texto cambia y el estado también ‘State: false’

*Detalle de nuestra lista con libros marcados como favoritos

*Vista general