En la siguiente sucesión de prácticas se abordará la creación de una pequeña aplicación que manejará una colección de libros. A medida que se avance con los contenidos del curso se irán implementando nuevas funcionalidades al proyecto. En esta ocasión: su creación y construcción de los primeros componentes.

1. Creación del proyecto

Sigue estos pasos:

- se instalan los programas ***nodeJS***, ***npm*** y ***Visual Studio Code*** en el equipo con el vayas a trabajar (modo de hacerlo está en la teoría de esta sesión),
- antes de la instalación es necesario limpiar la caché de `npm` para asegurar que se obtiene la versión más reciente,

```
npm cache clean –force
```

- se instalará la herramienta ***Vite*** que permitirá la creación de un proyecto `React` a través de un asistente,

```
npm install -g vite
```

- situándose en la carpeta en donde se va a localizar el proyecto, se lanza el siguiente comando para crearlo,

```
npm create vite@latest tus-iniciales-como-nombre
```

  como nombre del proyecto utiliza las iniciales de tu nombre. Hay que evitar el uso de caracteres especiales en el nombre de la aplicación,

- es posible que al lanzar este comando por primera vez solicite la instalación de un paquete denominado `create-vite@X.X.X`. Se trata de un paquete necesario para mostrar el asistente que ayudará a la creación del proyecto. Se acepta. Una vez lanzado el asistente sigue estos pasos:
  - se selecciona (pulsando la tecla Enter) el nombre del paquete que el asitente ofrece por defecto (que debe ser las siglas de tu nombre),
  - se selecciona el *framework* a emplear en el proyecto: `React`,
  - se elige el lenguaje de programación a utilizar en el proyecto (aparecerá como *variant* en la pantalla): `JavaScript`,
- tras la creación del proyecto hay que instalar todas sus dependencias. Todas ellas están enumeradas en el fichero `package.json`. Para ello:
  - se ubica en la carpeta del proyecto a través del comando,

```
cd tus-iniciales-como-nombre
```

  - se instalan todas las dependencias del proyecto,

```
npm install
```

- Cuando termine la instalación, se lanza el proyecto en un servidor local a través del siguiente comando,

```
npm run dev
```

Si todo ha ido bien debe aparecer una pantalla parecida a esta:

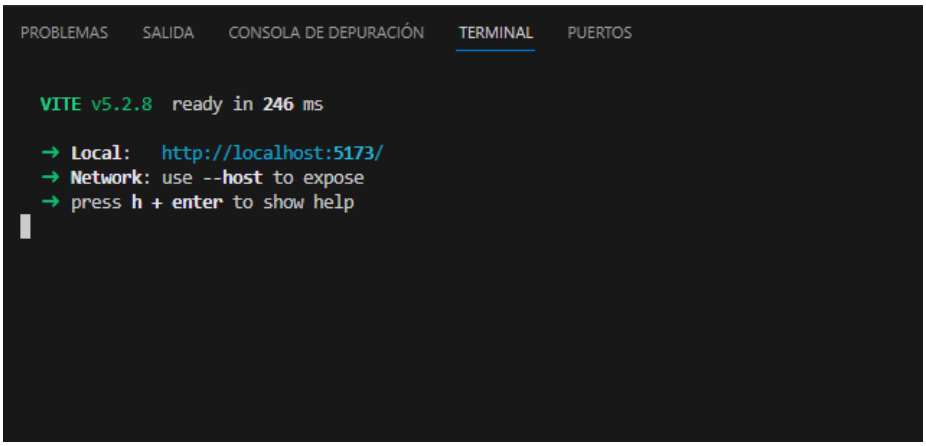

Figura 1: Resultado de la ejecución del comando `npm run dev`.

El último comando creará un servidor web con las bibliotecas necesarias para empezar a trabajar en el proyecto. La aplicación es accesible desde la red local (la URL está especificada en el terminal), lo que permite acceder al servidor desde un teléfono móvil o tableta para ir comprobando en tiempo real el desarrollo en este tipo de dispositivos.

Para lanzar la aplicación en un navegador hay que, en el terminal, escribir la letra O más la tecla Enter consecutivamente. Debe aparecer algo como esto en el navegador:

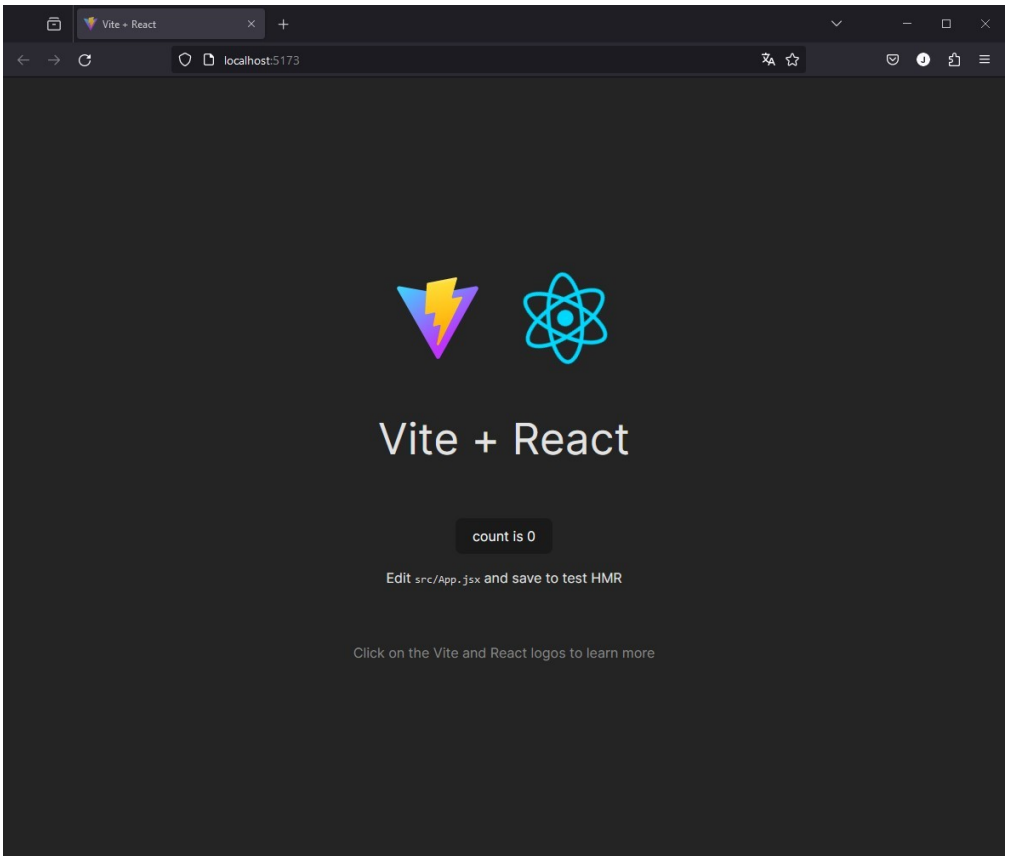

Figura 2: Apariencia del proyecto recién instalado.

Además de este comando es conveniente familiarizarse con los siguientes:

- `npm run build` , que genera el código final del proyecto. No es un compilador sino un transpilador que interpreta y optimiza el código del proyecto a código de producción,
- `npm run preview`, vista previa local del proyecto antes de su construcción,
- `npm run dev`, que inicia el servidor de desarrollo.

La instalación del proyecto trae consigo la construcción de una estructura de carpetas que es necesario conocer:

- `dist`, alberga los ficheros de producción de la aplicación. Estos ficheros se construyen interpretando el código de la carpeta `src` a través del comando `npm run build`,
- `node_modules`, contiene las dependencias del proyecto (bibliotecas asociadas),
- `public`, carpeta raíz del servidor donde se encuentra el `index.html` de la aplicación,
- `src`, contiene el código del proyecto `React`,
- `package.json`, que contiene información del proyecto, así como enumera sus dependencias tanto para desarrollo como para producción,
- `.gitignore`, que es el archivo para indicar a `Git` que ignore ciertos archivos durante la sincronización del proyecto,
- `package-lock.json`, archivo que no se debe tocar.

Para empezar a trabajar con `React` no se necesita más que entrar en la carpeta `src` y editar su código a través del fichero `App.jsx` , que será el componente índice de la aplicación que se va a construir:

- se abre el fichero `App.jsx` con *Visual Studio Code*,
- se selecciona todo el código existente entre los símbolos `<>` y `</>` (deben ser las líneas desde la once a la treinta),
- se elimina esas líneas,
- se escribe el siguiente código entre `<>` y `</>` :

```
<h1>¡Hola Mundo!</h1>
<h2>Soy un discente del curso de React del CEFIRE</h2>
```

- se guarda el archivo. La web del navegador debe actualizarse con el nuevo contenido de forma automática,
- debe aparecer algo parecido a esto:

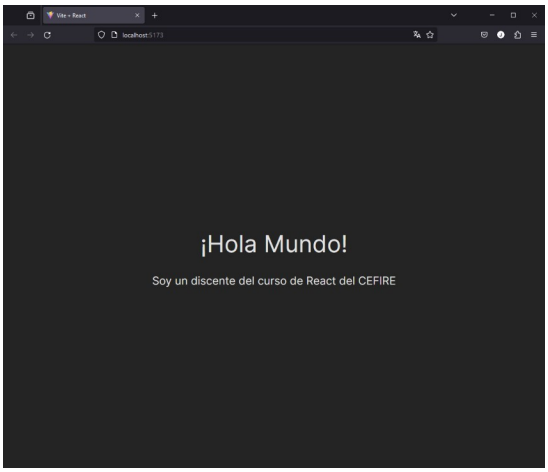

Figura 3: Apariencia del proyecto una vez modificado.

Bien, ya se dispone del servidor y se sabe cómo modificar el código. Ahora vamos a por cosas más serias.

2. Creación de componentes

Antes de empezar a crear componentes, es necesario entender cómo funciona `React`.

Fuera de la carpeta `src` se encuentra el fichero `index.html` con el siguiente `body`:

```
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
```

Existen dos etiquetas dentro de él: un `div` que contendrá todo el código de la aplicación (`root`) y otra `script` para cargar el fichero `main.jsx`. Este fichero (situado dentro de la carpeta `src`) contiene lo siguiente:

```
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Lo que hace este código, a través del método `createRoot` del objeto `ReactDOM`, es seleccionar el `div` con identificador `root` y dibujar el componente `<App>` dentro de él. Además, este componente está situado dentro de `<React.StrictMode>` que obliga a que el código *JavaScript* contenido en él esté escrito en modo estricto (más información sobre este modo aquí).

Lo que se intuye a partir de este código es que el resto de componentes que se creen para la aplicación deben estar contenidos en `<App>` para que sean dibujados en la página inicial. Esto se puede modificar y dibujar aquí cualquier componente creado, pero es buena idea seguir esta estructura ya que es la que cualquier programador espera encontrar.

El primer componente a crear será el encargado de mostrar la información de un libro de la biblioteca.

Para ello sigue estos pasos:

- el orden en un proyecto de `React` es fundamental, y para ello es buena idea crear una estructura de carpetas que mantenga todo en su sitio. Se crea una carpeta denominada `componentes`,
- se crea un fichero con el nombre `Libro.jsx` (recuerda que debe comenzar con mayúscula para que el transpilador de `React` no lo confunda con una etiqueta de `HTML`),
- se escribe la estructura de un componente funcional en `React` (aunque puede ser de otra forma, es conveniente que tanto el fichero, como la función, como el módulo de exportación del componente tengan el mismo nombre siempre, `Libro` en este caso):

```
// Se importan los recursos necesarios para este componente.
import React, { Fragment } from "react";
// Se crea la función (flecha) que construirá el componente.
const Libro = () => {

// Área para el código JavaScript (vacía en este ejemplo).

  return (
    /* Siempre es recomendable envolver el código en fragmentos. */
    <Fragment>
      {/* Zona para el código JSX. */}
      <article>
        <div>Título del libro</div>
          <div>Autor del libro</div>
      </article>
    </Fragment>
  );
};
// Se exporta el componente (la función que lo contiene).
export default Libro;
```

- se abre el fichero `App.jsx` y se borra el contenido situado entre los fragmentos `<>` y `</>`,
- se importa el nuevo componente para que pueda ser utilizado aquí:

```
import Libro from "./conponentes/Libro.jsx";
```

- se coloca el componente `<Libro>` dentro de los fragmentos tantas veces como libros se necesiten:

```
return (
  <>
    <Libro />
    <Libro />
  </>
);
```

- se debe obtener el código (del `return`) del componente `<Libro>` una vez por cada componente instanciado:

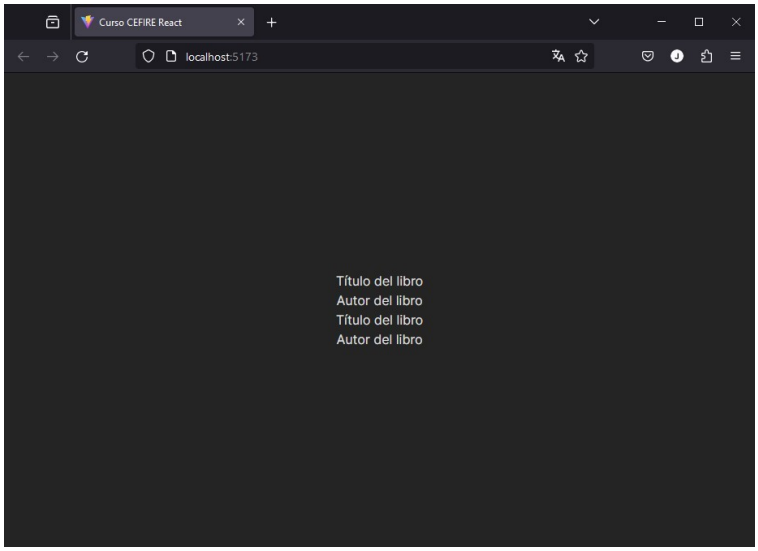

Figura 4: Resultado de la ejecución del código anterior.

Esta será la forma de trabajar de ahora en adelante: primero importar componente/recurso y luego utilizarlo.

Hay que parametrizar `<Libro>` (no en vano es una función) para poder imprimir los datos de cada elemento de la biblioteca. Eso se va a conseguir utilizando el objeto `props`.

Sigue estos pasos:

- se cambia el código de `<Libro>` para que acepte parámetros,

```
import React, { Fragment } from "react";
const Libro = (props) => {
  // Se usa la desestructuración de objetos para recoger los datos que interesan.
  const { portada, titulo, autor, id } = props;
  return (
    <Fragment>
      <article id={id}>
        <img width='150px' height='225px' src={portada}></img>
        <div>{titulo}</div>
        <div>{autor}</div>
      </article>
    </Fragment>
  );
};
export default Libro;
```

- en el componente `<App>`, se pasan parámetros a `<Libro>` para crear varios libros,

```
return (
  <>
    <Libro
      id='2ad6b5e2-9c2b-4959-b740-9335c85eed74'
      titulo='1984'
      autor='George Orwell'
      portada='https://imagessl4.casadellibro.com/a/l/t7/44/9788499890944.jpg'
    />
    <Libro
      id='9bbf84ae-20bc-4477-89c9-eab36ac417dd'
      titulo='¿Sueñan los androides con ovejas eléctricas?'
      autor='Phillip K. Dick'
      portada='https://imagessl7.casadellibro.com/a/l/t7/57/9788445006757.jpg'
    />
  </>
);
```

El resultado debe ser parecido a este:

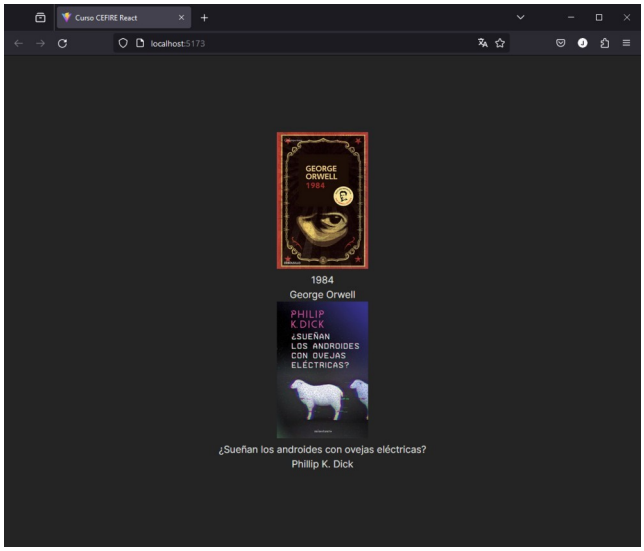

Figura 5: Componente `<Libro>` con propiedades.

React es muy quisquilloso con los errores, ¿qué pasa si algunas de sus propiedades no tienen valor? Es buena práctica ofrecer siempre una alternativa a los datos que se esperan por si estos no aparecen.

Para ello:

- en `<Libro>`, se carga una imagen alternativa por si la esperada no existe (se deberá descargar desde **Aules** y copiarla en la carpeta `assets/img/sin_portada.png` del proyecto),

```
import sin_portada from "../assets/img/sin_portada.png";
```

- se usa como alternativa (hay que recordar que los valores falsos en *JavaScript* son **`undefined`**, **`null`**, **`NaN`**, **`0`**, "" (cadena vacía) y **`false`**, por supuesto, por lo que si la propiedad esperada no contiene valor, se sustituye por la ruta importada en **`sin_portada`** (hay que recordar que sólo es posible el uso del operador ternario como estructura alternativa en **JSX**),

```
<img width='150px' height='225px' src={portada ? portada : sin_portada}></img>
```

- del mismo modo se cambia el resto de propiedades,

```
<article id={id ? id : crypto.randomUUID()}>
[...]
<div>{titulo ? titulo : "No se ha especificado título."}</div>
<div>{autor ? autor : "No se ha especificado autor."}</div>
```

- quedando el componente de la siguiente manera:

```
import React, { Fragment } from "react";
import sin_portada from "../assets/img/sin_portada.png";

const Libro = (props) => {
  const { portada, titulo, autor, id } = props;
```

```
  return (
    <Fragment>
      {/* Si no se especifica id, se crea uno nuevo con el método randomUUID(). */}
      <article id={id ? id : crypto.randomUUID()}>
        <img
          width='150px'
          height='225px'
          src={portada ? portada : sin_portada}
        ></img>
        <div>{titulo ? titulo : "No se ha especificado título."}</div>
        <div>{autor ? autor : "No se ha especificado autor."}</div>
      </article>
    </Fragment>
  );
};
export default Libro;
```

- de esta forma, si se utilizan los componentes de este modo en `<App>`,

```
return (
   <>
    <Libro
      id='2ad6b5e2-9c2b-4959-b740-9335c85eed74'
      titulo='1984'
      autor='George Orwell'
      portada='https://imagessl4.casadellibro.com/a/l/t7/44/9788499890944.jpg'
    />
    <Libro />
</>
);
```

el resultado debe ser parecido a este:

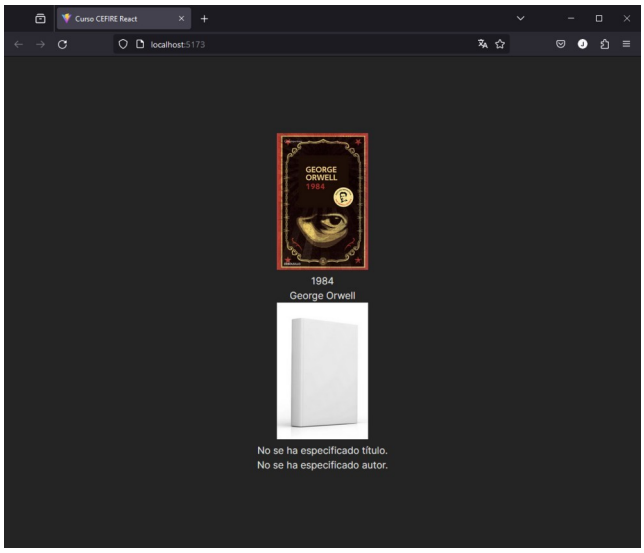

Figura 6: Resultado de `<Libro>` sin propiedades.

Ahora que ya se dispone de un componente robusto para mostrar libros, será conveniente extraer las propiedades de cada libro desde una fuente de datos y no colocar toda esta información a mano.

Para este proyecto se obtendrán los libros desde un fichero `JSON` externo (este fichero se descarga desde **Aules**) que se deberá copiar a la carpeta `assets/bbdd/biblioteca.json`. Habitualmente esta información se obtiene desde una `API REST`, como se estudiará en la última sesión, pero ahora este fichero `JSON` será suficiente para las necesidades actuales de la aplicación.

Además, se va a crear un nuevo componente encargado de mostrar un `<Libro>` por cada registro que exista en el origen de datos. La idea de la división en componentes de `React` es que cada uno de ellos se encargue de una tarea y sólo una. Así, `<Libro>` se encarga de mostrar la información de un libro y `<ListadoLibros>` (el nuevo componente) se encargará de mostrar los libros de una biblioteca.

Para ello:

- en la carpeta `componentes`, se crea un nuevo fichero denominado `ListadoLibros.jsx`,
- se crea la estructura de un componente funcional,

```
import React, { Fragment } from "react";
const ListadoLibros = () => {
  return (
    <Fragment>
      <h2>Listado de libros</h2>
    </Fragment>
  );
};
export default ListadoLibros;
```

- se añade este componente en `<App>` (no hay que olvidar importarlo antes) y elimina los componentes `<Libro>` (así como su importación). Se comprueba que se muestra el texto del componente `<ListadoLibros>`,
- se vuelve al código del componente `<ListadoLibros>` y se importa el origen de los datos desde el archivo `biblioteca.json` a la variable `biblioteca`,

```
import biblioteca from "../assets/bbdd/biblioteca.json";
```

  este fichero contiene un objeto con un sólo par de clave-valor: la clave es `libros` y el valor es una array de objetos `JSON`. Es buena práctica utilizar `console.log(elemento_a_inspeccionar)` para comprobar que los datos con los que se trabaja son correctos. Existen herramientas de depuración muy sofisticadas que están indicadas para problemas mucho más avanzados que los que se van tener en este proyecto. Por lo que el tradicional `console.log` será suficiente,
- comprueba con un `console.log(biblioteca)`, en la parte de *JavaScript* del componente, que el su contenido es un objeto con una clave `libros` que contiene un `array` de objetos `JSON`. Debe aparecer algo así por la consola del navegador:

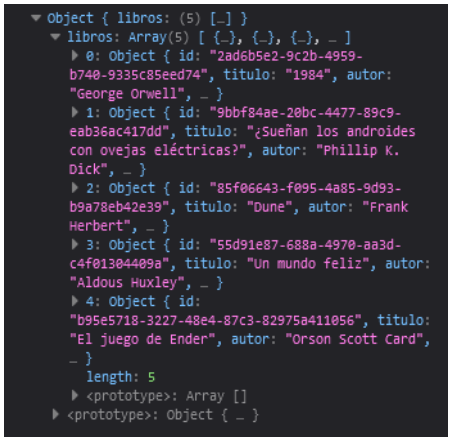

Figura 7: Contenido del fichero `biblioteca.json`.

Todo esto quiere decir que el `array` con los datos de los libros se encuentran en `biblioteca.libros`.

- el objetivo es mostrar un `<Libro>` por cada elemento de ese fichero, por lo que se hará uso de ese componente. Hay que importarlo en `<ListadoLibros>`,

```
import Libro from "./conponentes/Libro.jsx";
```

- para recorrer un objeto iterable en `JSX` sólo se dispone de su método `map`. En cada iteración se devolverá un `<Libro>` con la información pertinente:

```
return (
  <Fragment>
    <h2>Listado de libros</h2>
    {/* Se recorre el array libros dentro del objeto biblioteca. */}
    {biblioteca.libros.map((datos_libro) => {
    /* En cada iteración se devuelve un componente <Libro>. */
      return (
        <Libro
          id={datos_libro.id}
          titulo={datos_libro.titulo}
          autor={datos_libro.autor}
          portada={datos_libro.portada}
        />
      );
    })}
  </Fragment>
);
```

- se está suponiendo que todo irá bien y que siempre se obtendrán los datos de forma perfecta. Mala idea. Será necesario comprobar que los datos recibidos son un `array` y que, efectivamente, contiene datos,

```
return (
  <Fragment>
    <h2>Listado de libros</h2>
    {/* Si libros es un array y contiene algún elemento... */}
    {Array.isArray(biblioteca.libros) && biblioteca.libros.length
      ? /* ...se recorre el array mostrando los libros.  */
```

```
      biblioteca.libros.map((datos_libro) => {
        return (
          <Libro
            id={datos_libro.id}
            titulo={datos_libro.titulo}
            autor={datos_libro.autor}
            portada={datos_libro.portada}
          />
        );
      })
      : /* ...o se muestra un mensaje en su lugar. */
      "No se han encontrado libros."}
    </Fragment>
  );
```

estas comprobaciones son obligatorias para evitar que un error dé al traste con la ejecución de la aplicación. Todo debe comprobarse y no dejar nada al azar.

Si todo ha ido bien, se debe obtener un resultado como este:

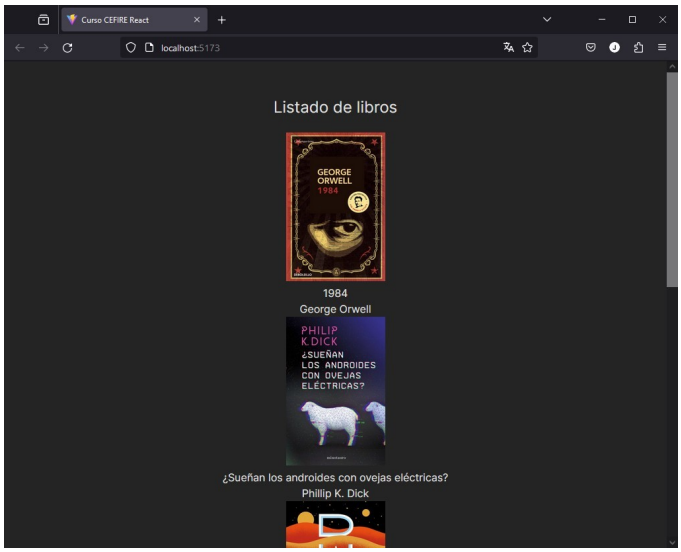

Figura 8: Resultado de `<ListadoLibros>`.

Pero no todo ha ido bien. Si se activa la consola del navegador (tecla F12) se observa que se ha producido una advertencia:

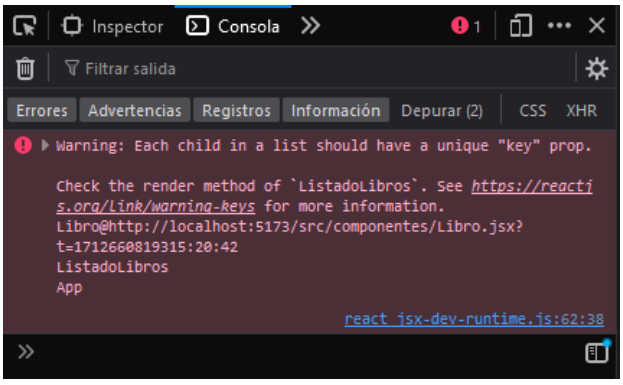

Figura 9: Advertencia de identificación en un `map`.

Para facilitar el trabajo al transpilador de `React`, será conveniente que cada etiqueta que se genere a través de una iteración posea un identificador. Al tratarse de un `map` y teniendo en cuenta que su función *callback* recibe tres parámetros (valor, índice de la iteración y el propio `array`) es posible utilizar el índice como identificador. Pero en esta ocasión se dispone de un identificador de cada libro, por lo que será éste el que se utilizará para esta acción (siempre es mejor idea que utilizar el índice de la iteración).

Para hacer esto, sigue estos pasos:

- la identificación se realiza con la propiedad `key` que sólo se usa en el proceso de transpilación del código y no tiene traducción a ninguna propiedad `HTML`. Ésta debe estar colocada en la etiqueta más externa, en el caso que nos ocupa se trata de `<Libro>`, por lo que el código final quedaría del siguiente modo:

```
biblioteca.libros.map((datos_libro) => {
  return (
    <Libro
      key={datos_libro.id}
      id={datos_libro.id}
      titulo={datos_libro.titulo}
      autor={datos_libro.autor}
      portada={datos_libro.portada}
    />
  );
});
```

  Esto evita la fastidiosa alerta y se facilita el proceso de creación del código.

La construcción de la aplicación será el resultado de repetir los pasos que se han realizado en esta sesión. En la siguiente se abordará la forma de añadir estilos y estructura.

Pero antes, hay que realizar unos cambios en estos componentes:

- `<Libro>` tiene demasiadas propiedades y es incómodo para trabajar. Sería más apropiado que recibiera toda esa información a través de un objeto `JSON` y que este objeto se desestructurara en su interior. Realiza los cambios necesarios para que `<Libro>` acepte un objeto (parámetro) como el siguiente y sustituya a las propiedades establecidas anteriormente:

```
const objetoSesion1 = {
   "id": "2ad6b5e2-9c2b-4959-b740-9335c85eed74",
   "titulo": "1984",
   "autor": "George Orwell",
   "portada": "https://imagessl4.casadellibro.com/a/l/t7/44/9788499890944.jpg",
   "completado": true,
   "sinopsis": "La historia comienza en el año 1984 en una sociedad inglesa dominada
por un sistema de colectivismo burocrático controlado por el Gran Hermano..."
}
```

- haz los cambios necesarios para que `<ListadoLibros>` pase los datos de forma correcta a `<Libro>` en cada iteración.

Entrega de la práctica.
Después de realizar todas las acciones que se solicitan, comprime en formato `ZIP` la carpeta `src` del proyecto y súbela a **Aules**.