1. Creación de componentes reutilizables

Para la descomposición del código en componentes se tendrá en cuenta la estructura de la aplicación, por lo que comprobaremos en el código la funcionalidad de los mismos. En nuestro caso, todos tienen la misma función: crear estructura de la aplicación, por lo que dentro de la carpeta de `componentes` se colocarán dentro de una misma carpeta, a la que se le dará el nombre de `estructura`. No dejan de ser componentes normales, pero con una función muy específica, por eso su separación del resto.

Inicialmente, el código en el fichero `App.jsx` es el siguiente:

```
const App = () => {
  return (
    <>
      <header>
        <div>Mi biblioteca</div>
      </header>
      <nav>
        <div>Esta es la barra de navegación.</div>
      </nav>
      <main>
        <section>
          <div>Contenido de la web.</div>
          <div>
            <ListadoLibros />
          </div>
        </section>
      </main>
      <footer>
        <div>
          Curso 24AI32CF029 - Programación reactiva para aplicaciones Web  (REACT)
        </div>
      </footer>
    </>
  );
}
```

Partiendo, por tanto, de esta estructura y teniendo en cuenta las implicaciones a nivel semántico de las distintas etiquetas `HTML` que hemos utilizado, se descompone el código de la siguiente forma colocando estos nuevos componentes en la nueva carpeta estructura que hemos creado:

- `Cabecera.jsx`. Se corresponde con la cabecera de la web y el elemento `header`. Su código será el siguiente:

```
import React, { Fragment } from "react";

const Cabecera = () => {
  return (
    <Fragment>
      <header>
        <h1>Mi biblioteca</h1>
      </header>
    </Fragment>
  );
};

export default Cabecera;
```

  No debemos olvidar a la hora de crear el componente incluir las declaraciones de `import` en la primera línea y de `export` en la última de ellas.

- `Navegacion.jsx`. Tomaremos la parte correspondiente al menú de navegación, o lo que es lo mismo, la etiqueta `nav`. A nivel semántico, el menú de navegación podría haberse incluido dentro de la cabecera (es decir, la etiqueta `nav` en el interior de la etiqueta `header`), pero como se trata de un componente con una funcionalidad diferente, lo situamos como uno nuevo. Procedemos igual que en el apartado anterior, creando dentro de la carpeta estructura este nuevo componente:

```
import React, { Fragment } from "react";

const Navegacion = () => {
  return (
    <Fragment>
      <nav>
        <ul>
          <li>Inicio</li>
          <li>Crear libro</li>
          <li>Buscar libros</li>
        </ul>
      </nav>
    </Fragment>
  );
};

export default Navegacion;
```

- `Contenido.jsx`. En este componente, que se corresponde con la etiqueta `main` de nuestro código, podemos observar que ya tenemos un componente creado: `<ListadoLibros>`, por lo que tenemos que ser más cuidadosos al separarlo del resto, no olvidando importar este componente en nuestro nuevo documento.

```
import React, { Fragment } from "react";
import ListadoLibros from "../ListadoLibros.jsx";
```

```
const Contenido = () => {
  return (
    <Fragment>
      <main>
        <ListadoLibros />
      </main>
    </Fragment>
  );
};

export default Contenido;
```

- `PiePagina.jsx`. Finalmente, el código de la etiqueta `footer` lo añadiremos al nuevo componente

```
import React, { Fragment } from "react";

const PiePagina = () => {
  return (
    <Fragment>
      <footer>
        <small>
          Curso 24AI32CF029 - Programación reactiva para aplicaciones Web (React).
        </small>
      </footer>
    </Fragment>
  );
};

export default PiePagina;
```

Ahora, nuestro árbol de directorios debería parecerse a éste:

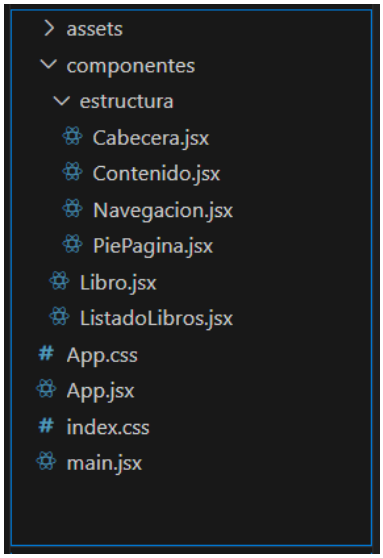

Figura 1: Árbol de directorios tras la descomposición por componentes.

- para replicar la estructura inicial de la aplicación habrá que colocar los nuevos componentes (previa importación) en nuestro índice, es decir, en `<App>` del siguiente modo:

```
const App = () => {
  return (
    <>
      <Cabecera />
      <Navegacion />
      <Contenido />
      <PiePagina />
    </>
  );
};
```

Como cada componente devuelve una porción del código inicial, la estructura de la aplicación quedará intacta pero creada a partir de componentes. Esto permitirá dividir el contenido en varias partes y trabar con ellas de forma independiente.

2. Modificación de las hojas de estilo globales

A partir de este momento, ya hemos tenemos todas las herramientas para utilizar las clases creadas con el fin de asociar estilos y mostrar una web más agradable a nivel visual, modificando en los diferentes componentes características como el tipo de letra o el color en la cabecera y el menú, el color de fondo o la situación del pie de página y la colocación de los contenidos.

Hay estilos que afectan a todos los componentes y que es buena idea agrupar en un fichero de estilo global a la aplicación. Existen dos posibilidades:

- utilizar el estilo de `<App>`, ya que contiene todos los componentes de la aplicación y su estilo se propagará hacia sus descendientes,
- o el fichero `index.css` que incluye incluso al propio componente `<App>`.

Para evitar duplicidades, se va a optar por una de las dos opciones: la primera, de esta forma todo el `CSS` general estará contenido en `App.css`.

Comenzaremos con el código básico que afecta al conjunto de la página como los colores, y que incluiremos en el archivo `App.css`. Habitualmente se incluye un archivo también un conjunto de estilos conocido como `reset.css` para eliminar los estilos que los distintos navegadores proporcionan por defecto, pero por no complicar más la estructura, simplemente vamos a incluir algunos de estos estilos base.

También emplearemos variables en `CSS`, que es como se denomina habitualmente a las *custom properties* o propiedades personalizadas, con el fin de tener disponibles fácilmente aquellos valores que se repiten a lo largo del código como colores o ciertas medidas.

Para poder realizar estas modificaciones, sigue estos pasos:

- accede a `main.jsx` y elimina la carga del fichero `index.css`,
- elimina ese fichero del proyecto,
- abre el fichero `App.css` y elimina el estilo sugerido,
- aplica este estilo para `App.css` (se puede alcanzar un resultado similar con otros estilos u otro planteamiento a nivel de estilos).

```css
@import url('https://fonts.googleapis.com/css2?
family=MuseoModerno:ital,wght@0,100..900;1,100..900&display=swap');

:root {
  font-family: "MuseoModerno", sans-serif;
  font-optical-sizing: auto;
  font-weight: 400;
  font-style: normal;
  line-height: 1;

  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;

  --primary-color: #4e4e4e;
  --secondary-color: #fff;
  --menu-color: #f8f9fa;
```

```
  --success-color: #2e8b57;
  --danger-color: #d01e1e;

  --default-size: 80%;
  --current-font: "MuseoModerno", sans-serif;
}

/* Archivo de reset */

* {
  padding: 0;
  margin: 0;
}

body {
  color: var(--primary-color);
  background-color: var(--secondary-color);
}

a { text-decoration: none; }
a:hover { text-decoration: none; }

#root {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}
```

En el código adjunto se pueden ver algunas pequeñas anomalías con respecto a la metodología **BEM**, como el uso de un identificador como es `#root` o de etiquetas para designar a elementos. Por un lado, las etiquetas se utilizan como si de una hoja de estilo de `reset` de tratara y, únicamente, en este documento. Con respecto al uso del identificador (`#root`), prohibido también en esta metodología, al igual que las etiquetas, es debido a que es un elemento ya proporcionado por **React**, por lo que resulta conveniente su uso en este documento, aunque también podría asignarse una clase al elemento y utilizarla. En el resto de documentos, sí se sigue la metodología.

3. La metodología BEM aplicada a los componentes

Ahora procederemos a aplicar estilo a los distintos componentes. El primer paso es crear las clases y para ello seguiremos la **metodología** BEM (Bloque - Elemento - Modificador) para mantener una coherencia a la hora de nombrarlas y mantener el código ordenado de forma sencilla:

- En el componente `<Cabecera>` podemos apreciar claramente que estamos ante un **bloque** como es el contenido dentro de la etiqueta `header`, ya que tiene sentido por sí mismo. Lo nombraremos como `.cabecera`. Siguiendo con la estructura propia de esta metodología, el título contenido en la etiqueta `h1` es un elemento propio dentro de la cabecera, por lo que manteniendo la nomenclatura su clase sería `.cabecera__titulo`.

```
<Fragment>
  <header className="cabecera">
    <h1 className="cabecera__titulo">Mi biblioteca</h1>
  </header>
</Fragment>
```

- Continuamos con el componente `<Navegacion>` donde realizaremos la misma tarea, identificando los bloques y los elementos que lo forman.

  En este caso, a la clase se le ha dado el nombre de `.menu`, pero podría ser cualquier otro que hiciera referencia a la estructura. Una vez dentro del bloque podemos observar que tenemos dos elementos, una lista e ítems en su interior.

```
<nav className='menu'>
  <ul className='menu__lista'>
    <li className='menu__item'>Inicio</li>
    <li className='menu__item'>Crear libro</li>
    <li className='menu__item'>Buscar libros</li>
  </ul>
</nav>
```

- En `<PiePagina>` ocurre algo similar a la cabecera pero con la etiqueta `footer`.

```
<footer className='pie'>
  <small className='pie__texto'>
    Curso 24AI32CF029 - Programación reactiva para aplicaciones Web (React).
  </small>
</footer>
```

- Finalmente, en `<Contenido>` incluimos la sección principal de la web con su correspondiente clase `.main`.

```
<main className='main'>
  <ListadoLibros />
</main>
```

Una vez diseñadas las clases, se debe crear un fichero de estilos por cada uno de los componentes. Cada uno de ellos contendrá las clases y el código CSS que afectará sólo al contenido de ese componente, de ese modo se separa el estilo en pequeños ficheros que son fáciles de manejar y no queda concentrado en un interminable fichero. Además, si es necesario reutilizar el componente en otro proyecto, tan sólo será necesario copiar tanto el fichero con extensión `jsx` como el `css` que lo acompaña.

Además, para poder asociar las hojas de estilo a cada uno de los componentes no debemos olvidarnos de incluir la cláusula de `import`. Por ejemplo, en el fichero de `Cabecera.jsx` se haría de la siguiente forma:

```
import "./Cabecera.css";
```

Ahora, añade el estilo a los componentes de estructura siguiendo estos pasos:

- descarga el fichero `sesion2_estilos.zip` desde **Aules**,
- sitúa estos ficheros junto a los componentes que dotan de estructura (en su misma carpeta),
- importa los estilos a cada uno de los componentes importando los ficheros,
- comprueba que el estilo se ha aplicado de forma correcta.

Por supuesto, también es necesario modificar el estilo de `<Libro>` y `<ListadoLibros>` con fin de poder proporcionarles una visualización adecuada.

Para ello, sigue estos pasos:

- añade las siguientes clases a `<Libro>`,

```
<Fragment>
  <article id={id ? id : crypto.randomUUID()} className='libro'>
    <img className='libro__portada'
      src={portada ? portada : sin_portada}
      width="150"
    ></img>
    <div className='libro__titulo'>
      {titulo ? titulo : "No se ha especificado título."}
    </div>
    <div className='libro__autor'>
      {autor ? autor : "No se ha especificado autor."}
    </div>
  </article>
</Fragment>
```

  En este código se aprecia cómo disponemos de un bloque, que se corresponde con cada uno de los libros, y de un conjunto de elementos: aquellos que proporcionan información sobre el libro, como título, portada o autor,

- crea el fichero `Libros.css` (junto a su homónimo `jsx`) y escribe el siguiente estilo en él:

```
.libro__contenido {
  padding: 10px;
  width: 150px;
}

.libro__titulo {
  margin: 5px 0;
}

.libro__autor {
  font-weight: bold;
}
```

- crea las clases en el componente `<ListadoLibros>`. Para conseguirlo, sigue estos pasos:
  - se crea el fichero `ListadoLibros.css`,
  - se importa ese fichero en `<ListadoLibros>`,
  - se identifican los bloques y componentes y se les asigna clases
  - se escriben las clases y el código `CSS` en el fichero de estilos para que la aplicación tenga este aspecto:

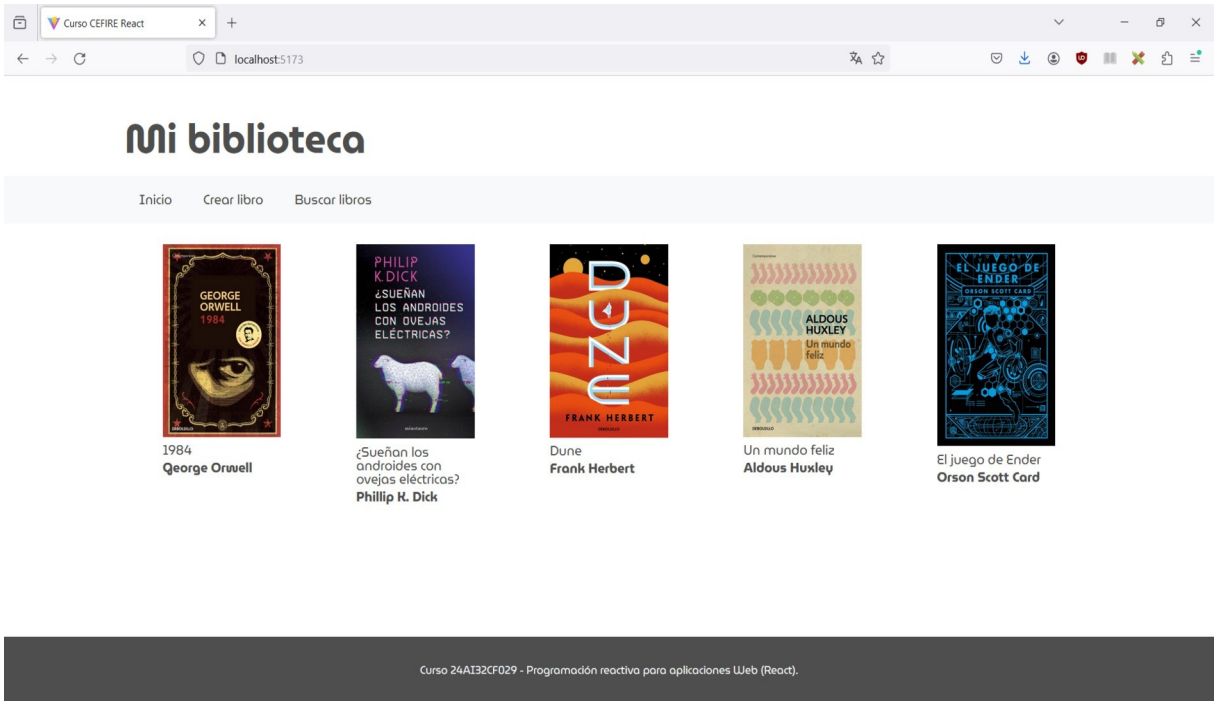

Figura 2: Modificaciones de nivel de estilo.

No se aprecia en la imagen anterior, pero cuando nos situamos con el cursor sobre un elemento del menú (primera parte de la imagen) o sobre un libro (imagen inferior), éstos modifican su aspecto para indicarlo:

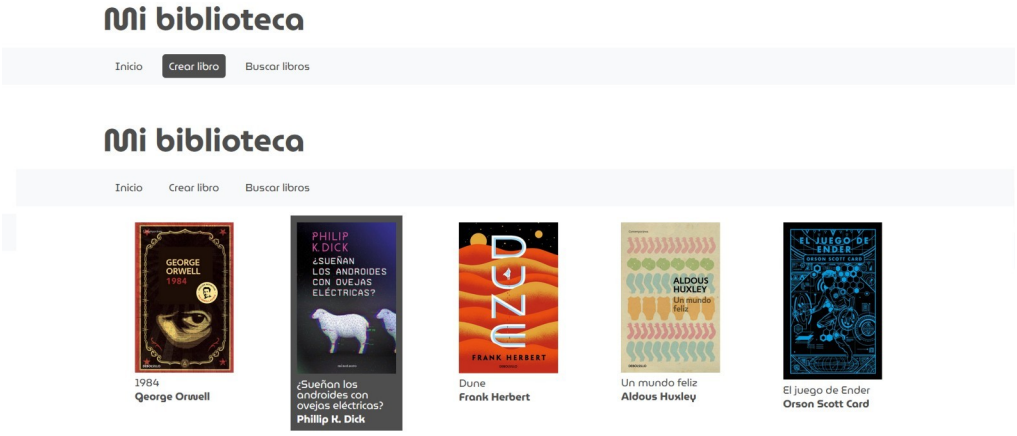

Figura 3: Elementos con una pseudoclase `:hover`.

Entrega opcional:

Se ha valorado la posibilidad de que nuestra aplicación se utilice en dispositivos más pequeños como tabletas (parte izquierda de la imagen)o móviles (parte derecha), por lo que también deberá reflejarse en el `CSS` en forma de *media queries* o consultas de medios:

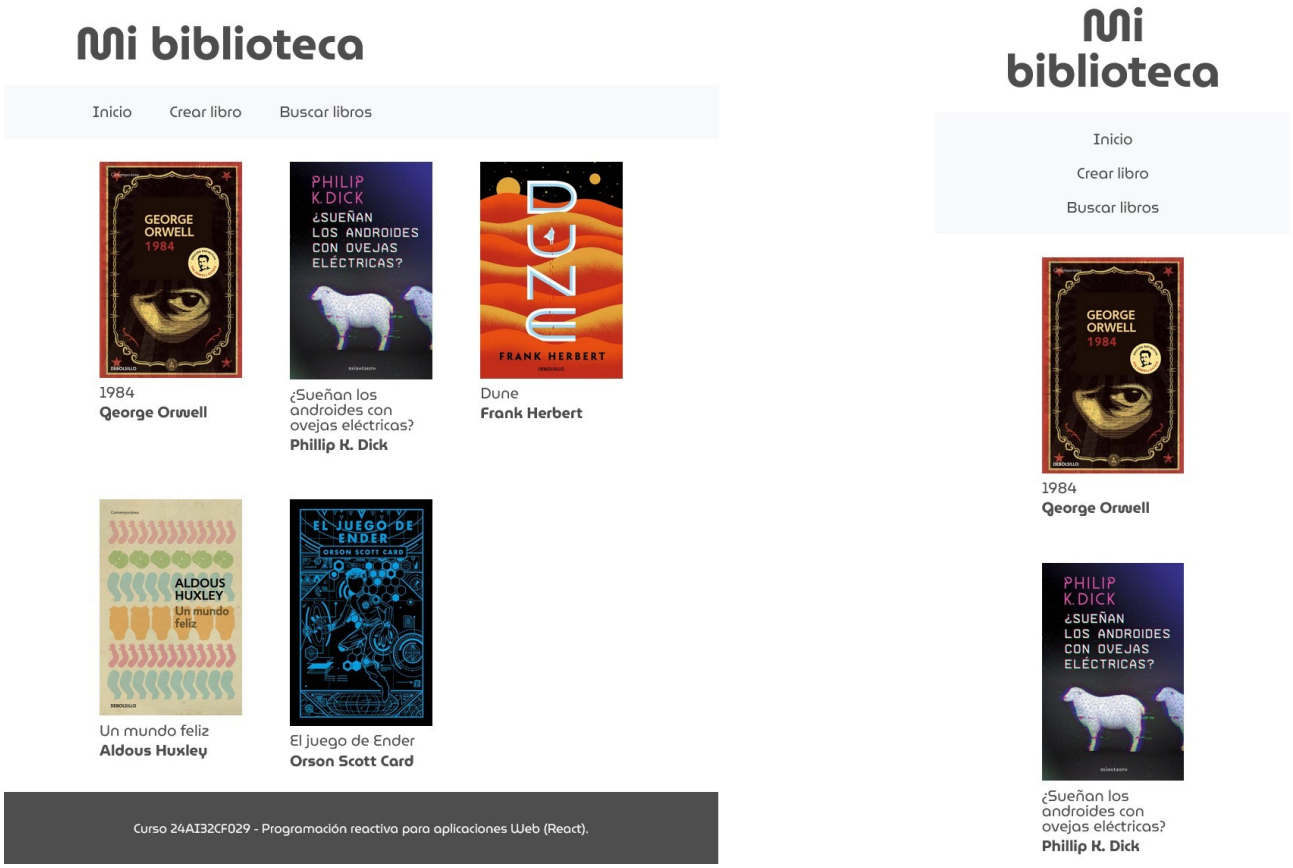

Figura 4: Vistas con distintos `viewport` o áreas de visualización.

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**.