1. Crear rutas en la aplicación

Ahora que se tiene claro qué es una ruta y cómo se trabaja con ellas, se van a incluir en la aplicación que nos ocupa, para ello sigue estos pasos:

- instala la biblioteca *React Router* en el proyecto (descrito en la parte teórica),
- importa el componente `<BrowserRouter>` desde la nueva biblioteca:

```
import { BrowserRouter } from "react-router-dom";
```

- para que las rutas funcionen todos los componentes de esta biblioteca deben estar contenidos dentro de su contexto, es decir, deben ser `children` de `<BrowserRouter>`. Es probable que se requiera poner enlaces en cualquier componente, por lo que será recomendable que todos los componentes de estructura (creados en la sesión anterior) se conviertan en descendientes del contexto de esta forma:

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

  Desde este momento `<Cabecera>`, `<Navegacion>`, `<Contenido>` y `<PiePagina>` pueden utilizar los componentes ofrecidos por `<BrowserRouter>`.

- hay que diseñar las secciones de la aplicación que actuarán como páginas. Son componentes que tendrán esa función, por lo que será recomendable mantenerlos separados del resto:
  - se crea una carpeta denominada paginas dentro de componentes,
  - se escribe el componente funcional `<Inicio>` que contendrá el listado de libros (será la página de inicio de la aplicación:

```
import React, { Fragment } from "react";
import ListadoLibros from "../ListadoLibros.jsx";
const Inicio = () => {
  return (
    <Fragment>
      <ListadoLibros />
    </Fragment>
  );
};
```

```
export default Inicio;
```

- se crea, en la carpeta `páginas` también, el componente `<Buscar>` con el siguiente contenido `JSX`:

```
return (
  <Fragment>
    <section className='buscar'>
      <h2 className='buscar__titulo'>Buscar libro</h2>
    </section>
  </Fragment>
);
```

- escribe los componentes funcionales `<Crear>` y `<Error>` del mismo modo, que muestren una etiqueta `<h2>` con el nombre del componente dentro de un `<section>` (recuerda poner las clases según el método `BEM` ajustada a cada componente),

- se descarga el fichero `sesion3_estilos.zip` desde `Aules` e importa los estilos a estos componentes (con la excepción de `<Inicio>` que no tiene),
- hora de definir las rutas. La mejor manera de hacerlo es ubicarlas dentro de un componente. Se tratará, por tanto, de un componente de estructura, por lo que estará en la carpeta `estructura` (aunque en realidad da igual donde se encuentre). Crea el componente funcional `<Rutas>` en dicha carpeta,
- se define el contenedor de las rutas a través del componente `<Routes>` de la biblioteca de *React Router* (no hay que olvidar su importación),

```
return (
  <Fragment>
    <Routes> </Routes>
  </Fragment>
);
```

- se importa el componente `<Route>` añadiéndolo a la importación anterior (ya que comparten origen),

```
import { Routes, Route } from "react-router-dom";
```

- se crea la primera ruta que dirigirá hacia en componente `<Inicio>` cuando se cargue la página, es decir, cuando la ruta en la barra de navegación sea `/`,

```
return (
  <Fragment>
    <Routes>
      <Route path='/' element={<Inicio />} />
    </Routes>
  </Fragment>
);
```

- se añaden el resto de rutas de la aplicación (recuerda importar los componentes),

```
return (
  <Fragment>
    <Routes>
      <Route path='/' element={<Inicio />} />
      <Route path='/crear' element={<Crear />} />
      <Route path='/buscar' element={<Buscar />} />
    </Routes>
  </Fragment>
);
```

de este modo, cuando la barra de direcciones contenga alguna de las opciones especificadas en `path`, cargará el componente especificado en `element`,

- se incluye una ruta por defecto por si no hay correspondencia con ninguna de las anteriores, en cuyo caso se cargará el componente `<Error>`,

```
return (
  <Fragment>
    <Routes>
      <Route path='/' element={<Inicio />} />
      <Route path='/crear' element={<Crear />} />
      <Route path='/buscar' element={<Buscar />} />
      <Route path='*' element={<Error />} />
    </Routes>
  </Fragment>
);
```

- todas las rutas definidas se cargarán dentro del componente `<Routes>` en función del valor de la barra de navegación, por tanto, allá donde se sitúe `<Rutas>` se cargarán estos componentes. Parece que el más indicado para asumir esta tarea es `<Contenido>` por lo que se elimina `<ListadoLibros>` (y su importación) y se coloca en su lugar `<Rutas>` (tras su importación),

```
return (
  <Fragment>
    <main className="main">
      <Rutas />
    </main>
  </Fragment>
);
```

- ejecuta el proyecto y el resultado debe ser el siguiente:

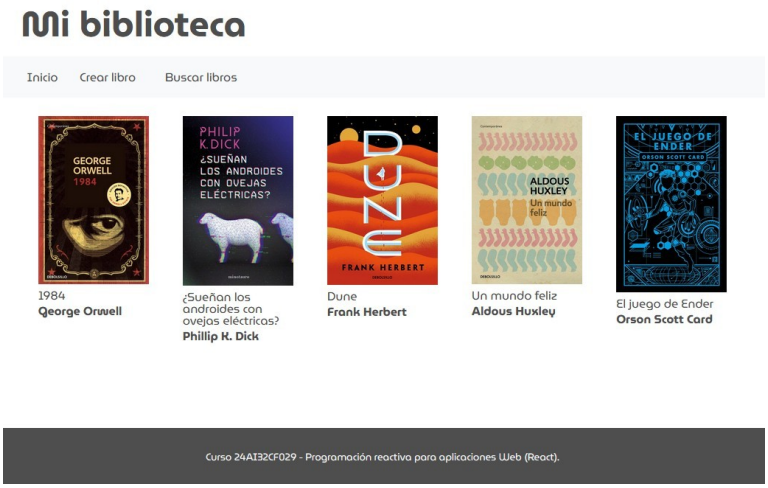

Figura 1: Resultado de la aplicación con rutas.

En efecto, no debe haber cambiado nada ya que, aunque ahora se están aplicando las rutas en la aplicación, se está cargando el componente `<Inicio>` (la ruta actual es `/`) que tiene los mismos elementos que tenía el componente `<Contenido>`,

- es hora de modificar `<Navegacion>` para navegar entre las rutas. Se comprueba que su contenido es el siguiente:

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

- se importa el componente `<NavLink>` desde la biblioteca de *React Router*:

```
import { NavLink } from "react-router-dom";
```

- se añade el primer enlace de la aplicación que conducirá hasta la ruta `crear`:

```
<li className='menu__item'>
  <NavLink to='/crear'> Crear libro </NavLink>
</li>
```

- de igual modo, se añaden los enlaces para las rutas `inicio` y `buscar` quedando del siguiente modo:

```
return (
  <Fragment>
    <nav className='menu'>
      <ul className='menu__list'>
```

```
        <li className='menu__item'>
          <NavLink to='/'>Inicio</NavLink>
        </li>
        <li className='menu__item'>
          <NavLink to='/crear'>Crear libro</NavLink>
        </li>
        <li className='menu__item'>
          <NavLink to='/buscar'>Buscar libros</NavLink>
        </li>
      </ul>
    </nav>
  </Fragment>
);
```

desde este momento ya se podrá navegar por la aplicación cambiando de “página” sin la necesidad de recargar el navegador.

La razón por la que se ha elegido `<NavLink>` y no `<Link>` para la construcción de los enlaces es que el primero permite cambiar la clase cuando detecta que el elemento está activo de forma muy sencilla. Su atributo `className` permite utilizar una función *callback* que recibe un parámetro boleano denominado `isActive` (entre otros) que informa de si es el elemento activo.

Para hacer esto, sigue estos pasos:

- añade una función flecha al atributo `className` de `<NavLink>`,

```
className={ () => {}  }
```

Las llaves exteriores son para indicar que se está utilizando código *JavaScript* en zona **JSX**, como ya se sabe. Dentro de ellas se encuentra la función flecha que actuará de *callback* (de momento vacía). Esta función recibe un objeto con las claves boleanas `isActive`, `isPending` e `isTransitioning`, pero sólo interesa uno en esta ocasión: `isActive`, por lo que se procederá a la desetructuración de ese objeto para obtenerlo,

```
className={({ isActive }) => {} }
```

para simplificar el código se ha desestructurado el objeto en la definición de la función, pero también se podría haber hecho del modo tradicional,

```
className={(objeto) => {
  const { isActive } = objeto; // Recomendable no utilizarlo en esta situación.
}}
```

la idea es mantener sencillo el código por lo que se utilizará la primera,

- si `isActive` es `true`, hay que añadir un estilo activo; si no lo es, hay que quitarlo,

```
className={({ isActive }) => {
  return isActive ? `menu__link menu__link--activo` : `menu__link`;
}}
```

como sólo hay una instrucción en la función flecha se puede utilizar su versión reducida (muy recomendable acostumbrarse a esta versión):

```
className={({ isActive }) => isActive ? `menu__link menu__link--activo` : `menu__link`}
```

en ella se eliminan la llaves de la función flecha y la palabra `return`. La versión larga de este código totalmente desaconsejada en esta situación (aunque equivalente) es la siguiente:

```
className={(objeto) => {
  const { isActive } = objeto;
  return isActive ? `menu__link menu__link--activo` : `menu__link`;
}}
```

- al final el código debe quedar del siguiente modo:

```
<li className='menu__item'>
  <NavLink to='/'
    className={
      ({ isActive }) => isActive ? `menu__link menu__link--activo` : `menu__link`
    }
  >
    Inicio
  </NavLink>
</li>
```

- añade, del mismo modo, el atributo `className` al resto de `<NavLink>`.

Con este cambio el menú activo debe mostrar una clase diferente:

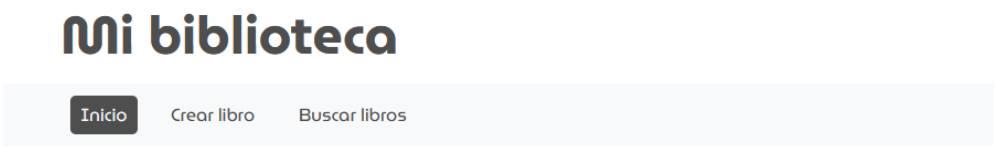

Figura 2: Elemento de menú activo.

Se han añadido algunos estilos nuevos a `<Navegación>`, puedes consultarlos en `Navegacion.css`.

2. Detalles de un libro

Para esta práctica, junto con el enunciado, se dispone de un nuevo componente (junto con su estilo) llamado `<LibroDetalles>` que se encargará de mostrar los detalles de un libro pasado como parámetro. La idea es utilizar este componente para que cada vez que se pinche sobre un libro se muestren sus detalles.

Para conseguirlo hay que seguir estos pasos:

- este componente estará contenido en una página de la aplicación, por lo que habrá que crear un nuevo componente funcional `<Mostrar>` en la carpeta `paginas`. Su contenido será el siguiente:

```
return (
  <Fragment>
    <section className='mostrar'>
      <h2 className='mostrar__titulo'>Mostrar</h2>
    </section>
  </Fragment>
);
```

- se crea un fichero para el estilo de este componente denominado `Mostrar.css` (en la misma carpeta) con el siguiente contenido:

```
.mostrar {
  padding: 30px 0;
  margin: 0 auto;
}

.mostrar__titulo {
  font-size: 2em;
}
```

- se añade la nueva ruta hacia esa página dentro de `<Routes>` en `<Rutas>`,

```
<Route path='/mostrar' element={<Mostrar />} />
```

No importa el orden en el que se escriban las rutas para su correcto funcionamiento, aunque sí es cierto que la ruta por defecto (`<Error>`) suele situarse al final de todas ellas,

- para acceder a esta ruta es necesario pinchar en cada `<Libro>`, por lo que será necesario que cada uno de estos componentes se convierta en `children` de `<Link>` y no hay mejor manera de hacerlo que dentro del `map` que los itera en `<ListadoLibros>`, de modo que se pasa de este código (en el interior del `map`):

```
return ( <Libro key={datos_libro.id} datos={datos_libro} /> );
```

a este otro (recuerda, importa el nuevo componente):

```
return (
  <Link key={datos_libro.id} to='/mostrar' className='listado__libro' >
    <Libro datos={datos_libro} />
  </Link>
);
```

cada vez que se pinche en un `<Libro>` se estará, en realidad, activando una ruta a través de `<Link>` que conducirá hasta `<Mostrar>`. Hay que tener en cuenta que tras este cambio, `<Libro>` ya no es la etiqueta más externa dentro del `map`, sino `<Link>`, por eso contiene el atributo `key`. Esta vez se ha optado por utilizar `<Link>` en lugar de `<NavLink>` puesto que no será necesario el cambio de estilo en función del estado del botón, aún así, si se decide utilizar `<NavLink>` también es una buena opción a pesar de que está diseñado para menús,

- se comprueba que todo funciona de forma correcta y al pulsar sobre cada `<Libro>` de aparece en pantalla `<Mostrar>`,

Figura 3: Aspecto del componente `<Mostrar>`.

- en `<Mostrar>`, se añadirá un nuevo objeto libro para que sea detallado (en la parte de *JavaScript* del componente):

```
const libro = {
  id: "85f06643-f095-4a85-9d93-b9a78eb48r54",
  titulo: "Yo, robot",
  autor: "Isaac Asimov",
  portada: "https://imagessl0.casadellibro.com/a/l/t7/40/9788435021340.jpg",
  completado: false,
  sinopsis:
    "Esta obra visionaria tuvo una gran influencia en la ciencia ficción posterior, pero también en la propia ciencia de la robótica. Asimov formuló por primera vez las tres leyes fundamentales de la robótica, con claras implicaciones éticas y psicológicas. ¿Qué diferencia hay entre un robot inteligente y un ser humano? ¿Puede el creador de un robot predecir su comportamiento? ¿Debe la lógica determinar lo que es mejor para la humanidad? Yo, robot conecta entre sí una serie de historias de todo tipo de máquinas inteligentes a través de la robopsicóloga Susan Calvin. Estos robots son cada vez más perfectos y llegan a desafiar en ocasiones a sus creadores.",
};
```

- además, se elimina la etiqueta `<h2>` y se sustituye por el componente `<LibroDetalles>` que recibe como atributo el nuevo libro creado,

```
return (
  <Fragment>
    <section className='mostrar'>
      {libro
        ? <LibroDetalles libroBuscado={libro} />
        : "No se ha encontrado ningún libro."
      }
    </section>
  </Fragment>
);
```

Ahora, cada vez que se haga clic cobre un `<Libro>` mostrará el detalle del libro seleccionado. Resulta evidente que siempre se mostrará el mismo y que la mejor manera de hacer esto es pasando parámetros a través de las rutas, pero esa tarea se realizará en lo sucesivo ya que es recomendable utilizar contextos para evitar importar varias veces la fuente de datos, y ese contenido aún no se ha tratado. Así que, de momento, se queda como está.

Figura 4: Aspecto de la aplicación al pulsar sobre un libro.

Para finalizar con esta parte de la práctica, queda comprobar la ruta por defecto. Para conseguir esto, sigue estos pasos:

- en `<LibroDetalles>`, inmediatamente a continuación del `<div>` que contiene la sinopsis, se crean dos botones con el siguiente código:

```
<input type='button' value='Eliminar de la biblioteca' className='boton boton--cancelar'/>
<input type='button' value='&lt; Atrás' className='boton boton--volver' />
```

- ahora se añade el código necesario para realizar las siguientes acciones:
  - el primero botón dirigirá a `rutaInexistente` para comprobar que se carga `<Error>` de forma correcta,
  - el segundo dirigirá al inicio de la aplicación.

3. Usando iconos en los detalles

En `<LibroDetalles>` hay un detalle, valga la redundancia, que es muy mejorable: mostrar si se ha leído o no el libro. Tal y como está el componente, esta información de muestra a través de una cadena de texto:

```
{completado ? "Leído" : "No leído"}
```

Se va a sustituir estas palabras por iconos de la biblioteca *Font Awesome*. Para hacer esto, sigue estos pasos:

- se instalan las bibliotecas necesarias: `fontawesome-svg-core`, `free-regular-svg-icons` y `react-fontawesome` (revisa la teoría para ver cómo hacer esta acción),
- si se tenía el servidor en marcha durante el proceso de instalación (por algún motivo), se reinicia,
- se importa el componente que contendrá los iconos:

  ```
  import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
  ```

- también se importan los iconos a utilizar, dos en este caso:

  ```
  import {
    faCircleCheck,
    faCircleXmark,
  } from "@fortawesome/free-regular-svg-icons";
  ```

- para mostrar un icono en `React` se utiliza el componente recién importado de la siguiente manera:

  ```
  <FontAwesomeIcon title='Leído' icon={faCircleCheck} />
  ```

- además es posible añadirle clases de `CSS` y hasta modificar su tamaño:

  ```
  <FontAwesomeIcon
    title='Leído'
    icon={faCircleCheck}
    className='libro-detalle__completado libro-detalle__completado--true'
    size='2x'
  />
  ```

  Este componente dispone de muchas propiedades que será conveniente consultar en su documentación.

- cambia la ternaria para que, en lugar de las cadenas de texto, muestre el icono `faCircleCheck` si el libro ha sido leído y `faCircleXmark` si, por contra, no lo ha sido.

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**.