Curso: REACT

Docente: David Alcolea

DETALLE Y DESCRIPCIÓN (introducción)

Nombre de la Actividad **Aplicación con REACT contratación excursiones por Argentina**

Objetivos de la Actividad:

- Construir una aplicación SPA (Single Page Application) con utilización de componentes
- Uso del framework React de javascript
- Utilizar rutas
- Uso del contexto de React

Modalidad Actividad: Ejercicio por etapas (1/2)

DESCRIPCIÓN DEL EJERCICIO

Se necesita crear una plataforma para que los usuarios puedan contratar excursiones o rutas por varias zonas de Argentina.

Toda la operativa se realizará en una única pantalla de forma que las opciones de menú cargarán los distintos componentes necesarios para la consulta de excursiones, contratación de las mismas y visualización del resumen de excursiones contratadas

Entrada a la plataforma

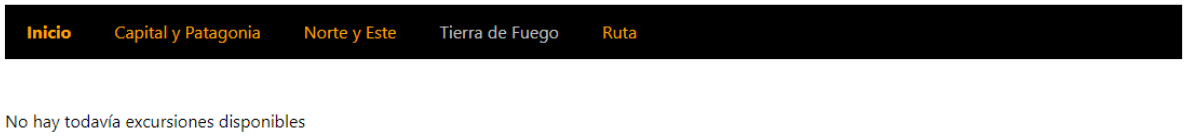
Al entrar a la plataforma se mostrará una sección superior con las opciones de la plataforma y una sección inferior con una imagen

- La opción inicio nos devolverá a la pantalla que vemos en la imagen superior
- Las opciones de '*Capital y Patagonia*', '*Norte y Este*' y '*Tierra de Fuego*' permitirán acceder a la lista de rutas disponibles para cada una de las tres zonas
- La opción 'Ruta' permitirá acceder al resumen de rutas contratadas con el total del precio a pagar

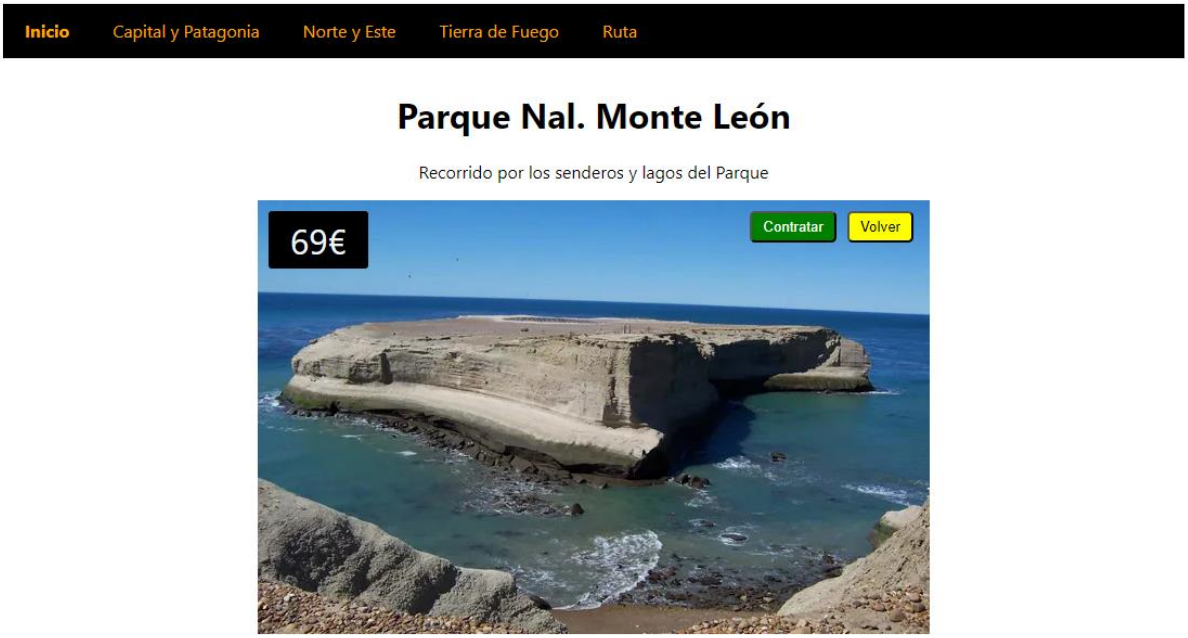
Operativa de consulta de rutas

Al pulsar sobre cualquiera de las opciones de menú '*Capital y Patagonia*', '*Norte y Este*' o '*Tierra de Fuego*' se mostrará el componente con la lista de rutas disponibles para cada una de las zonas

Si la opción seleccionada no tiene ninguna ruta asociada, se mostrará el siguiente mensaje. Ejemplo para 'Tierra de Fuego'



Al pulsar, en cada ficha de la ruta, el botón 'Más Info' accederemos a una nueva pantalla en donde se mostrará el detalle de la ruta seleccionada

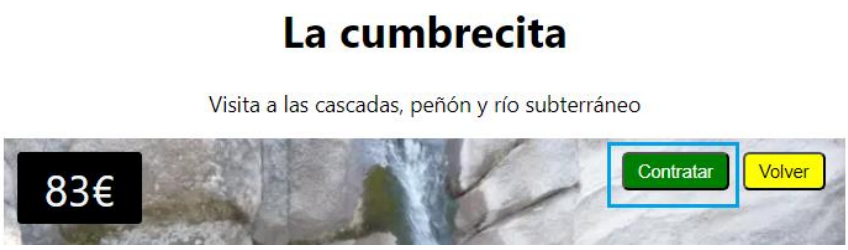


Mostraremos nombre, descripción y precio de la ruta y los botones:

- *Contratar* → Para contratar la ruta seleccionada
- *Volver* → Para volver a la pantalla anterior

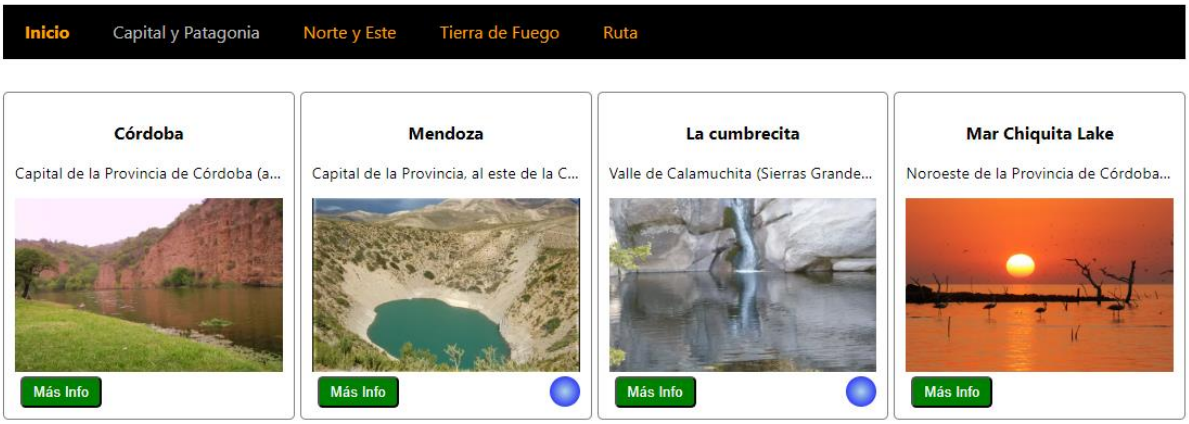
Operativa de contratación de rutas

Si se ha seleccionado una ruta específica desde cualquiera de las tres opciones que corresponden a cada una de las tres zonas para acceder al detalle de la misma, al pulsar el botón de *Contratar* se añadirá la ruta a una lista de rutas contratadas



Y, una vez contratada, cambiaremos el botón por otro de 'Anular'

Todas las rutas contratadas aparecerán con un circulo azul en la pantalla de consulta de todas las rutas por zona. Ejemplo para la ruta anterior contratada:



Operativa de consulta de rutas contratadas

Podremos acceder a la opción de menú ‘Rutas’ en donde consultaremos todas las rutas que el usuario a contratado

Para cada ruta informaremos:

- Foto en formato pequeño para poder identificar mejor la ruta contratada
- Nombre y precio de la ruta
- Botón de anular la contratación de la ruta
- Informaremos también el total a pagar para todas las rutas seleccionadas

Si no hubiera ninguna ruta contratada mostraremos el siguiente mensaje:

ESPECIFICACIONES TÉCNICAS

Especificaciones técnicas a considerar para el desarrollo de la plataforma:

- Se utilizarán componentes para cada una de las secciones de la plataforma:
  - inicio
  - lista de rutas
  - detalle de ruta
  - resumen de rutas contratadas
- Se utilizará un enrutador en un fichero externo para la navegación:
- Las rutas contratadas las guardaremos en el localStorage para no perderlas al refrescar la página y las recuperaremos cuando se cargue de nuevo la página
- Toda la construcción de la plataforma se debe realizar de forma dinámica y robusta, es decir, debe ser independiente del número de excursiones y zonas que tengamos en el array de objetos del que extraeremos la información y que se adjunta como recurso del ejercicio

  Ejemplo:

  Si en vez rutas de Argentina queremos mostrar rutas de la bella Región de Murcia solo tendríamos que cambiar los datos del array, las fotos de la carpeta de imágenes donde las guardemos y, únicamente, tendríamos que modificar el título que aparece en la página de Inicio:

  Argentina y Patagonia por **Región de Murcia**

- En toda la descripción técnica de la implementación de la plataforma se dará por entendido y conocido que tendremos que importar dentro del fichero de cada uno de los componentes el resto de componentes, directivas o hooks de React que se necesiten

Partes del desarrollo de la plataforma

Parte 0:

Descripción de la plataforma (el punto anterior)

Parte 1:

Creación del proyecto y componentes de la aplicación

Parte 2:

Creación de las rutas

Parte 3:

Creación de los contenidos dinámicos de todos los componentes

Parte 4:

Operativa de contratación y anulación de excursiones o rutas

Parte 5:

Mostrar el resumen de rutas contratadas y exportarlo a pdf o imprimirlo

Parte 6:

Guardar las rutas contratadas en el storage

EJERCICIO 1: CREACIÓN DEL PROYECTO Y COMPONENTES

Crearemos un proyecto React llamado **Argentina** desde el cmd del sistema o el terminal de VSC

create-react-app argentina

Editamos el fichero **src/App.css** y borraremos el contenido

Editamos el fichero **src/App.js** y vaciamos el contenido del return de la función **App()**

1.1 Creación de todos los componentes de la aplicación

Vamos a crear todos los componentes de la aplicación dentro de la carpeta **src/paginas:**

- **Inicio.js** → componente de inicio de la aplicación
- **Excursiones.js** → componente para mostrar todas las excursiones de cada ruta
- **Excursion.js** → componente para cada una de las fichas para cada excursión
- **Detalle.js** → componente de detalle de la excursión seleccionada
- **Ruta.js** → componente con el resumen de las excursiones contratadas

Por temas de organización de código, cada uno de los componentes anteriores irá acompañado de su correspondiente archivo **NombreComponente.css** de forma que evitaremos incorporar todo el css dentro de **App.css** (esto es opcional y no afecta al desarrollo de la actividad)

En cada uno de los componentes incorporaremos el fichero css con un import. Ejemplo para **Inicio.js**:

```
import './Inicio.css'
```

1.2 Preparación del entorno

Incorporamos las imágenes adjuntas en los recursos del ejercicio dentro de la carpeta **public/images**

Incorporamos los dos arrays de objetos **zonas.js** y **excursiones.js** con los datos de la aplicación (y que utilizaremos en los componentes de la misma) en la carpeta **src/context**

1.3 Componente principal App

En el componente principal de momento no incorporaremos nada, pero si añadiremos el css común para el resto de componentes de la aplicación dentro de **App.css**:

```
body {margin:0}
*{box-sizing: border-box;}
.App {width: 60%;margin:auto;border: 1px solid grey;padding: 30px;}
```

1.4 Componente Inicio

Editaremos el componente **Inicio.js** para añadir el contenido con el título y la imagen de entrada a la aplicación

```
<div className='inicio'>
    <h1>Argentina y Patagonia</h1>
    <hr/>
    <img src='../images/argentina.webp' alt='argentina'/>
</div>
```

Y, en el archivo **Inicio.css** añadiremos el css necesario:

```
.inicio {text-align: center;}
```

Para probarlo, incorporamos provisionalmente el componente dentro de **App.js**:

```
<div className="App">
    <Inicio/>
</div>
```

1.5 Componente Excursiones

Editaremos el componente **Excursiones.js** para añadir la pantalla de la operativa de consulta de todas las rutas o excursiones de cada una de las zonas

```
<div className='excursiones'>
    <Excursion/>
 </div>
```

NOTA: Podríamos crear un componente excursiones para cada una de las zonas (por ejemplo: **excursiones_ca.js**, **excursiones_ne.js**, etc.) pero esto no sería buena idea ya que si, en algún momento, añadimos, modificamos o suprimimos una zona, habría que modificar el código de la aplicación para añadir, modificar o eliminar el componente afectado.

Lo que vamos a hacer es crear un único componente y, posteriormente, utilizando la función map de javascript, leeremos el array de excursiones para mostrar una ficha (un componente **Excursion.js** que crearemos en el siguiente paso) para cada una de las excursiones

En el archivo **Excursiones.css** añadiremos el css necesario:

```
.excursiones {display: grid; grid-template-columns: repeat(auto-fit,minmax(250px, 1fr)); gap: 5px;}
```

Para probarlo, incorporamos provisionalmente el componente dentro de **App.js.** De momento no veremos nada ya que el componente **Excursion** todavía no existe (lo crearemos en el siguiente paso)

1.6 Componente Excursion

Editaremos el componente **Excursion.js** para añadir la pantalla con los datos de cada una de las fichas mostradas en el componente **Excursiones**

```
<div className='excursion'>
    <p className='titulo'>Nombre excursión</p>
    <p className='descripcion'>Situación de la excursión</p>
    <img src={`../images/buenosaires.jpg`} alt=''></img>
    <div className='flex'>
     <button>Más Info</button>
    </div>
</div>
```

NOTA: De momento, para probar el componente, utilizaremos literales fijos en el nombre y descripción de la excursión y cualquiera de las imágenes de la carpeta **public/images**

En el archivo **Excursion.css** añadiremos el css necesario:

```
.excursion { border: 1px solid grey; border-radius: 5px;padding: 10px;}
.titulo {text-align: center; font-weight: bold;}
img {width: 100%;}
button {background-color: green;color: white;padding: 5px 10px;border-radius: 3px;}
.flex {display: flex;justify-content: space-between;align-items: center;margin-top: 5px;}
```

Para probarlo, incorporamos provisionalmente dos o tres componentes **Excursion** dentro del componente **Excursiones.js.**

```
<div className='excursiones'>
    <Excursion/>
    <Excursion/>
    <Excursion/>
</div>
```

1.7 Componente Detalle

Editaremos el componente **Detalle.js** para añadir la pantalla con los datos de detalle de la excursión seleccionada en el componente **Excursiones**

```
<div className='detalle'>
    <h1>Nombre excursión</h1>
    <p>Servicio contratado</p>
    <div className='imagenruta'>
     <div className='precioruta'>999€</div>
     <div className='botonesruta'>
        <button className='rojo'>Anular</button>
        <button className='verde'>Contratar</button>
        <button className='yellow'>Volver</button>
     </div>
     <img src={`../images/buenosaires.jpg`} alt=''/>
    </div>
</div>
```

NOTA: De momento, para probar el componente, utilizaremos literales fijos en el nombre, servicio y precio de la excursión y cualquiera de las imágenes de la carpeta **public/images**

En el archivo **Detalle.css** añadiremos el css necesario:

```
.detalle {text-align: center;}
.imagenruta {position: relative;width: fit-content;margin: auto;}
.precioruta {border-radius: 3px;background-color: black;color: aliceblue;position: absolute; top:10px;
left: 10px;font-size: 2em;padding: 5px 20px;}
.botonesruta {position: absolute; top:10px; right: 10px;}
button {color: white;padding: 5px 10px;border-radius: 5px;margin: 0px 5px}
.verde {background-color: green;}
.yellow {background-color: yellow;color:black}
.rojo {background-color: red;}
```

Para probarlo, incorporamos provisionalmente el componente dentro del componente **App.js.**

1.8 Componente Ruta

Editaremos el componente **Ruta.js** para añadir la pantalla con la lista de las excursiones contratadas y el precio total a pagar

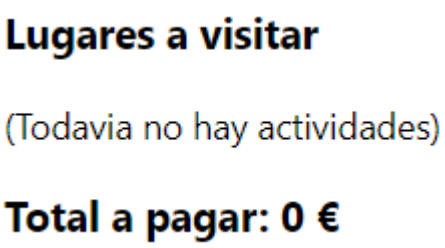
```
<div className='ruta'>
    <h3>Lugares a visitar</h3>
    <p>(Todavia no hay actividades)</p>
    <h3>Total a pagar:0 €</h3>
</div>
```

NOTA: De momento, para probar el componente, utilizaremos literales fijos en la lista de rutas y precio total

En el archivo **Ruta.css** añadiremos el css necesario:

```
..ruta button {background-color: red;color: white;padding: 0px 10px;border-radius: 3px;}
.ruta img {width: 80px; margin-right: 10px;}
.ruta p {display: flex; align-items: center;}
```

Para probarlo, incorporamos provisionalmente el componente dentro del componente **App.js.**



EJERCICIO 2: CREACIÓN DE LAS RUTAS

Vamos a crear ahora todas las rutas de la aplicación y la barra de navegación para acceder a ellas. Lo haremos en tres pasos:

1. Incorporar la librería de enrutado de React
2. Creación de la barra de navegación
3. Creación de un único enrutador con todas las rutas para poder probar la carga de todos los componentes

2.0 Importar librería de enrutado de React

El primer paso es incorporar la librería de enrutado de React desde el cmd o el terminal de VSC:

```
npm add react-router-dom
```

Y añadir la etiqueta <BrowserRouter> englobando todos los componentes de la aplicación que vayan a utilizar el enrutado. Puesto que lo van a utilizar todos los componentes lo que vamos a hacer es incorporarla dentro del fichero **index.js** de la siguiente forma:

```
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

2.1 Creación de la barra de navegación

Crearemos un nuevo componente para la barra de navegación dentro de una carpeta **navbar/NavBar.js**

```
<nav>
    <NavLink to='/'><b>Inicio</b></NavLink>
    <NavLink to='/excursiones/cp'>Capital y Patagonia</NavLink>
    <NavLink to='/excursiones/ne'>Norte y Este</NavLink>
    <NavLink to='/excursiones/tf'>Tierra de Fuego</NavLink>
    <NavLink to='/ruta'>Ruta</NavLink>
</nav>
```

NOTA 1: Hemos creado tres enlaces estáticos para cada una de las zonas que tenemos en el array zonas. Más adelante los sustituiremos por enlaces que se construirán dinámicamente utilizando el método map de javascript sobre el array zonas que encontraréis en recursos

NOTA 2: No hemos incorporado en la barra el acceso al componente **detalle.js** ya que a este componente solo podremos acceder mediante el botón 'Ver más' que se encuentra en cada ficha de las rutas

Añadimos también el css necesario dentro de **NavBar.css**

```
nav {background-color: black; margin-bottom: 30px;height: max-content; position: sticky;
top:0px}
nav a {text-decoration: none; line-height: 50px; color:orange; padding: 10px 20px;}
nav a:hover {color:white; }
.active {color:silver}
.loginlogout {float: right; }
nav button {padding: 5px 10px;margin: 10px; background-color: red;}
```

De momento no inyectaremos todavía el componente en ningún sitio. Vamos a crear primero el enrutador y luego lo inyectaremos en él para poder probarlo

2.2 Creación del enrutador

Vamos a confeccionar un fichero de rutas que utilizaremos para inyectar la barra de navegación y definir todas las rutas de la aplicación. Lo crearemos en la carpeta **router/Router.js**

```
<div>
    <NavBar/>
    <Routes>
       <Route path='/' element={<Inicio/>}/>
       <Route path='/excursiones/:zona' element={<Excursiones/>}/>
       <Route path='/ruta' element={<Ruta/>}/>
       <Route path='/detalle/:id' element={<Detalle/>}/>
       <Route path='/*' element={<Navigate to='/'/>}/>
    </Routes>
</div>
```

Vamos a ver en detalle cada una de los componentes del fichero:

- Incorporamos en este fichero la barra de navegación **NavBar** que hemos creado antes
- Hemos definido un grupo de rutas (con el componente <Routes> de react):
    - La ruta **'/'** que corresponderá al componente **Inicio.js**
    - Las rutas '**/excursiones**', '**/detalle'** y **'/ruta'** que corresponderán con los componentes **Excursiones.js, Detalle.js** y **Ruta.js**
    - Si en la url se introduce una ruta que no corresponde con las anteriores (ruta '**/***') redirigimos con Navigate a la ruta de inicio

Para poder probar la carga de componentes con la barra de navegación nos falta inyectar el enrutador **Router.js** dentro del componente principal **app.js**:

```
<div className="App">
     <Router/>
</div>
```

SI accedemos a la aplicación deberíamos ver la barra de navegación y, si pulsamos sobre cada uno de los enlaces, deberíamos ver cada uno de los componentes asociados a cada uno de ellos

2.3. Confección dinámica de la barra de navegación

Una de las especificaciones técnicas de la plataforma es que ésta sea lo más robusta posible de forma que, si añadimos, modificamos o eliminamos alguna zona para la que tengamos excursiones, no tengamos que modificar el código de la aplicación

Si nos fijamos en el array adjunto en recursos **zonas.js** veremos que hay tres objetos que corresponden a cada una de las tres zonas que mostramos en la barra de navegación

```
const zonas = [
  {
    zona: 'cp',    descripcion: 'Capital y Patagonia'
  },
  {
    zona: 'ne',   descripcion: 'Norte y Este'
  },
  {
    zona: 'tf',  descripcion: 'Tierra de Fuego'
  }
];
```

```
<NavLink to='/excursiones/cp'>Capital y Patagonia</NavLink>
<NavLink to='/excursiones/ne'>Norte y Este</NavLink>
<NavLink to='/excursiones/tf'>Tierra de Fuego</NavLink>
```

Vamos a optimizar la creación de los enlaces de la barra de navegación que tenemos en **navbar/NavBar.js** para confeccionarlos de forma dinámica utilizando el método map de javascript:

```
import zonas from '../context/zonas';

<nav>
    <NavLink to='/'><b>Inicio</b></NavLink>
    {
     zonas.map((item) =>
       <NavLink key={item.zona} to={`/excursiones/${item.zona}`}>{item.descripcion}</NavLink>
     )
    <NavLink to='/excursiones/cp'>Capital y Patagonia</NavLink>
    <NavLink to='/excursiones/ne'>Norte y Este</NavLink>
    <NavLink to='/excursiones/ne'>Norte y Este</NavLink>
    }
   <NavLink to='/ruta'>Ruta</NavLink>
</nav>
```

NOTA: Recordad que React exige que cada elemento creado desde una iteración tenga una clave única (key), para ello utilizamos el propio código de zona

Si nos fijamos en la definición de la ruta del componente excursiones que tenemos en el fichero **router/Router.js** vemos tiene un parámetro que tenemos que enviar obligatoriamente como parte de la misma:

<Route path='/excursiones/**:zona**' element={<Excursiones/>}/>

Podemos comprobar como, efectivamente estamos enviando el código de zona como valor para este parámetro

<NavLink key={item.zona} to={`/excursiones**/${item.zona}**`}>{item.descripcion}</NavLink>

Más adelante veremos como recoger este parámetro en el componente <Excursiones/> para mostrar la lista de excursiones de la zona que corresponda al enlace pulsado

EJERCICIO 3: CREACIÓN DE LOS CONTENIDOS DINÁMICOS

Aun tenemos los contenidos estáticos que hemos utilizado al crear los componentes pero ahora toca modificarlos para añadir los contenidos reales que tenemos en el array suministrado en recursos **excursiones.js**

```
const excursiones = [
  {
    'id' : 1,
    'nombre':'Cafayate',
    'zona':'ne',
    'imagen':'cafayate.jpg',
    'situacion':'Localidad de los valles Calchaquíes de la provincia de Salta.',
    'servicio':'Visita a las formaciones de roca caliza y viñedos',
    'precio':90
  },
  … /…
}
```

3.1 Componente Excursiones.js

Si nos fijamos en la ruta asociada a la carga de este componente veremos que tenemos que enviar un parámetro que corresponda a la zona a consultar:

En **Router.js**: <Route path='/excursiones**/:zona**' element={<Excursiones/>}/>

En **NavBar.js** : <NavLink key={item.zona} to={`/excursiones/**${item.zona}**`}>{item.descripcion}</NavLink>

El primer paso será, por tanto, recuperar este parámetro desde el componente **Excursiones.js** para conocer de que zona tenemos que mostrar las excursiones. Para ello utilizaremos el hook useParams de react:

```
const {zona} = useParams()
```

Con el parámetro zona recuperado consultaremos en el array excursiones todas aquellas que pertenezcan a esta zona y, para ello, utilizaremos el método filter de JS que nos devolverá solo los objetos que cumplan la condición del filtro

```
const excursionesZona = excursiones.filter((item) => item.zona === zona)
```

Por último recorreremos con map el nuevo array excursionesZona con los resultados del filtro para mostrar un componente <Excursion> por cada uno de los objetos que contenga:

```
<div className='excursiones'>
    {excursionesZona.length === 0
    ? <p>No hay todavía excursiones disponibles</p>
    : excursionesZona.map((excursion) =><Excursion key={excursion.id} excursion={excursion}/>
    )}
 </div>
```

Fijémonos que hemos añadido una comprobación para que, si el array está vacío porque no hay excursiones para esta zona, aparezca un mensaje '*No hay todavía excursiones disponibles*' y que era otro de los requisitos de la plataforma

Por el contrario, si el array tiene excursiones, mostramos un componente <Excursion> para cada una de ellas.

A este componente le pasamos la propiedad con el objeto que corresponda a cada una de las excursiones.

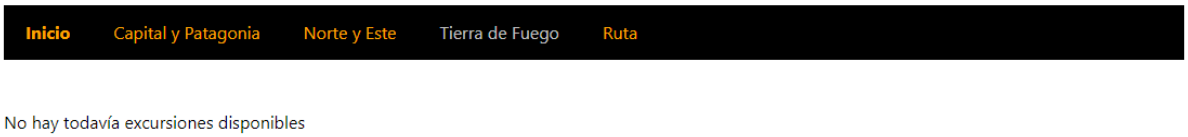
Para que cada enlace sea único hemos utilizado el atributo id del array excursiones ya que es un valor único que no se repite en ninguna de ellas (correspondería a una clave primaria de una tabla de base de datos si la obtención de los mismos fuera desde un petición a un backend, que no es nuestro caso)

Prueba de la operativa

SI entramos a las opciones de menú '*Ciudad y Patagonia*' y '*Norte y Este*', que si tienen excursiones asociadas, veríamos

NOTA: Obviamente vemos todas las fichas iguales porque todavía no hemos añadido contenido dinámico al componente Excursion

SI entramos a la opción '*Tierra de Fuego*', que no tiene ninguna excursión asociada, veríamos:



No hay todavía excursiones disponibles

Pero ¿Y si el usuario manipula la url y introduce una zona que no exista, por ejemplo: localhost:3000/excursiones/**zz**?

Pues tal como tenemos la aplicación veríamos el mismo mensaje que tenemos cuando una zona válida no tiene excursiones:

Ya que el array que obtenemos con filter estaría vacío

No obstante, es mejor utilizar una operativa diferente para estos casos y lo que vamos a hacer es, sencillamente, si la zona que nos llega en el parámetro de entrada al componente no existe, redireccionaremos a inicio o a cualquiera de las zonas que tengamos como válidas:

```
const {zona} = useParams()

if (!zonas.find(item => item.zona === zona)) {
   return <Navigate to='/excursiones/cp'/>
}
```

Buscamos con find el parámetro zona en el array de objetos zonas y, si no existe, redirigimos con <Navigate> a la ruta */excursiones/cp*

NOTA: Es importante que la sintaxis del enlace sea '/excursiones/cp' y no 'excursiones/cp'. Si probamos con el segundo (sin el primer hash) veremos como se carga el componente de inicio. Esto es debido a que si no ponemos el hash inicial '/' la ruta que estamos confeccionando sería: localhost:3000/excursiones/zz/excursiones/cp que, por supuesto no existe y acaba cargándose en componente de inicio

3.2 Componente Excursion.js

Este es el componente que se inyecta en el componente anterior **Excursiones.js** de forma que se mostrará uno por cada excursión que obtengamos para cada zona consultada

Ahora mismo muestra una serie de literales fijos pero, en su lugar, tendríamos que mostrar los datos que correspondan a cada una de las excursiones que mostramos en el componente donde lo inyectamos:

- El primer paso es recoger el parámetro que enviamos cuando inyectamos el componente desde **Excursiones.js**:

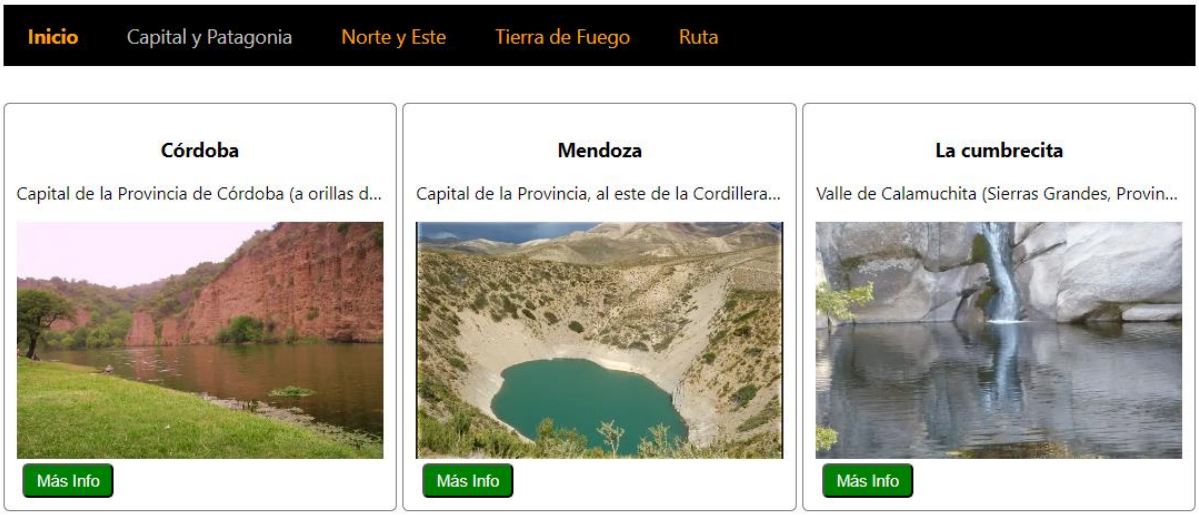
  ```
  const Excursion = ({excursion}) => { ... }
  ```

  NOTA: Tenemos que desestructurar con {} el parámetro de entrada ya que nos llega en formato objeto (recordad que con la desestructuración podemos obtener variables simples a partir de los atributos de un objeto)

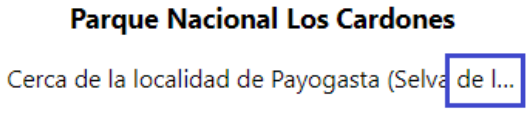
- Con los datos de cada una de las rutas ya podemos confeccionar dinámicamente el contenido de cada una de las fichas que mostramos en el componente <Excursiones>

```
<div className='excursion'>
    <p className='titulo'>{excursion.nombre}</p>
    <p className='descripcion'>{excursion.situacion}</p>
    <img src={`../images/${excursion.imagen}`} alt={excursion.nombre}></img>
    <div className='flex'>
     <button>Más Info</button>
    </div>
</div>
```

Si ahora probamos la aplicación, deberíamos ver como se muestran las excursiones que corresponden a cada una de las rutas seleccionadas en la barra de navegación



Si nos fijamos en la descripción de la ruta en cada ficha, vemos que solo ocupa una línea y que, si la descripción es más larga, aparece una elipsis con puntos suspensivos …



Esto se consigue editando el fichero **Excursion.css** para añadir estos estilos:

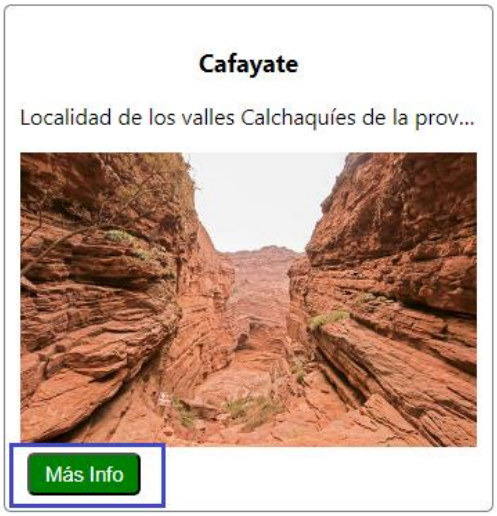
```
.descripcion {font-size: 0.9em; text-overflow: ellipsis;overflow: hidden; white-space: nowrap;
/*impide que la linea de texto continua en la linea siguiente*/}
```

3.3 Componente de detalle de la ruta seleccionada

Cuando en cada una de las rutas, que corresponden a la zona seleccionada en el menú, pulsemos sobre el botón 'Más info' tenemos que cargar el componente **Detalle.js** en donde mostraremos el reto de información sobre la ruta: precio, nombre y descripción

3.3.1. Modificación componente **Excursion.js**

Primero tendremos que volver a modificar el componente donde mostramos cada excursión para activar el enlace al componente de detalle cuando el usuario pulse sobre el botón 'Más info'



Para ello incorporamos el botón dentro de un componente <Link>:

```
<Link to={`/detalle/${excursion.id}`}><button>Más Info</button></Link>
```

Fijaos como pasamos como parámetro de la ruta el id de la excursión a mostrar tal como exige la definición de la ruta en el fichero **router/Router.js**:

```
<Route path='/detalle/:id' element={<Detalle/>}/>
```

3.3.2. Modificación componente **Detalle.js**

En este componente tendremos que consultar el detalle de la excursión que nos llega en el parámetro de la url después de seleccionarla en la pantalla de consulta de excursiones por zona.

- El primero paso es recuperar el parámetro con el id de la ruta a mostrar utilizando el hook useParams

  const {id} = useParams()

- Posteriormente buscamos la ruta dentro del array excursiones utilizando el método find:

  const excursion = excursiones.find((item) => item.id === Number(id))

- Editamos el código JSX del componente para sustituir los literales fijos que tenemos ahora por el contenido dinámico obtenido del array excursiones

```
<div className='detalle'>
    <h1>{excursion.nombre}</h1>
    <p>{excursion.servicio}</p>
    <div className='imagenruta'>
     <div className='precioruta'>{excursion.precio}€</div>
     <div className='botonesruta'>
         <button className='rojo'>Anular</button>
         <button className='verde'>Contratar</button>
         <button className='yellow' onClick={volver}>Volver</button>
     </div>
     <img src={`../images/${excursion.imagen}`} alt={excursion.nombre}/>
    </div>
</div>
```

Si probamos la aplicación y pulsamos sobre el botón '*Más Info*' en el componente <Excursiones> deberíamos ver como se carga el componente de detalle con la info de la ruta

Vemos también en este componente los botones para contratar la ruta y anularla que veremos en detalle en el siguiente ejercicio. Pero si que vamos a dejar activo el botón '*Volver*' que nos permitirá regresar a la pantalla que habíamos utilizado para seleccionar esta ruta:

- Hemos añadido un evento onclick al botón:

  ```
  <button className='yellow' onClick={volver}>Volver</button>
  ```

- Necesitaremos usar el hook useNavigate de react para poder indicar que se cargue la ruta anterior cuando se puse sobre el botón de volver

  ```
  const navegacion = useNavigate()
  ```

- Y creamos la función volver con el siguiente contenido:

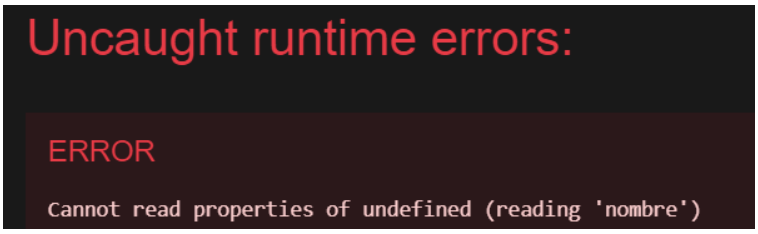
  ```
  const volver = () => {
     navegacion(-1) //retornar a la página anterior
  }
  ```

  NOTA: El parámetro -1 indica al hook useNavigate que queremos volver a la ruta inmediatamente anterior y que corresponderá a la ruta que nos cargará el componente <Excursiones> con la zona correspondiente a la excursión consultada

Si probamos el botón veremos como se carga el componente inmediatamente anterior: 'Capital y Patagonia' o 'Norte y Este' que son las dos únicas zonas para las que tenemos rutas

¿Y si enviamos por la url un parámetro que corresponda a una ruta o excursión que no exista, por ejemplo *localhost:3000/ruta/99*'?

Pues que veremos un error severo del tipo:



Ya que estamos intentado acceder en el documento JSX a los atributos de un objeto excursion que tiene el valor nulo (no tiene, por tanto, atributos) ya que no se ha encontrado ninguna ruta con el id que nos llega en el parámetro

¿Cómo lo solucionamos?

Lo más sencillo sería comprobar que si el objeto excursion, obtenido de la búsqueda de la ruta en el array excursiones, tiene valor null, redirigimos a inicio:

```
if (!excursion) return <Navigate to={`/`}/>
```

EJERCICIO 4: CONTRATACIÓN Y ANULACIÓN DE RUTAS

Tenemos la plataforma completamente desarrollada para mostrar todas las rutas y el detalle de las mismas.

Nos falta añadir la operativa de contratación y anulación de excursiones o rutas que es, realmente, el objetivo final de la aplicación.

Para ello necesitaremos un nuevo array en donde guardaremos el id de todas las rutas que vayamos contratando (será un array escalar sencillo con solo valores numéricos, no hará falta que sea un array de objetos).

En cualquier caso necesitaremos acceder a este nuevo array, que llamaremos contratacion, desde varios componentes y, además necesitaremos la correspondiente función asociada para poder añadir o eliminar rutas de él. Utilizaremos por tanto el Contexto

4.1 Creación del contexto

Vamos a necesitar para el procedimiento de contratación de rutas dos elementos:

- La variable contratacion que será un array con las rutas contratadas
- El correspondiente método setContratacion para poder añadir o eliminar rutas del array

Vamos a utilizar el hook useState para trabajar con esta variable pero, como tenemos que utilizarla en varias operativas (las operativas de contratar y eliminar rutas) en varios componentes, necesitaremos crear un contexto para poderla utilizar en estos componentes

Por lo tanto vamos a crear, para empezar, nuestro fichero de contexto en **context/Contexto.js**

```
import { createContext } from "react";
const contexto = createContext();
export default contexto;
```

Este fichero únicamente contiene la variable de creación de Contexto pero necesitamos otro fichero para poder utilizarlo como proveedor de las variables y funciones que necesitemos compartir entre los componentes que las vayan a utilizar. Crearemos por tanto, un segundo fichero **context/Provider.js** con las siguientes operativas:

- Indicamos como parámetro de entrada en la función del componente, que vamos a utilizar el contexto para todos los componentes hijos que se encuentren dentro de las etiquetas <Provider></Provider> que colocaremos más adelante en el componente principal **App.js**

  ```
  const Provider = ({children}) => { ... }
  ```

- Creamos la variable contratacion con el hook useState dentro de la función Provider del componente y la inicializamos como un array vacío

  ```
  const [contratacion, setContratacion] = useState([])
  ```

- Una vez hemos creado la variable contratacion con su método setContratacion que son los dos elementos que necesitaremos en otros componentes, tenemos que indicar en este fichero de contexto lo que necesitaremos compartir. Y esto lo hacemos de la siguiente forma:

```
return (
  <Contexto.Provider value={{
    contratacion,
    setContratacion
   }}>
    {children}
  </Contexto.Provider>
)
```

Indicamos los dos elementos a compartir y el componente {children} hace referencia en dónde queremos compartirlo y que serán todos los componentes hijos que se encuentren dentro de las etiquetas <Provider></Provider> que colocaremos en el siguiente punto

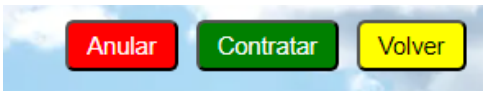
- Y, por fin, el último paso es indicar en qué componentes utilizaremos el contexto que serán todos aquellos que se encuentren dentro del componente principal. Para ello editaremos el fichero **App.js** y indicamos:

```
return (
  <div className="App">
   <Provider>
    <Router/>
   </Provider>
  </div>
);
```

4.2 Contratación/anulación de la ruta en Detalle.js

Vamos a modificar el componente de detalle para poder activar los botones de 'Contratar' y 'Anular' y, además, para que solo se muestre uno de ellos:

- El de contratar si la ruta todavía no ha sido contratada
- El de anular si la ruta ha sido contratada y queremos anular la contratación



- Añadimos los eventos para cada uno de los botones:

```
<button className='rojo' onClick={() => anular(id)}>Anular</button>
<button className='verde' onClick={() => contratar(id)}>Contratar</button>
```

NOTA: Pasamos como parámetro de la función el id de la ruta a contratar o anular

- Estas funciones se encargaran de añadir o eliminar la ruta contratada en el array contratacion y, para ello, necesitamos recuperar el método setContratacion del contexto que hemos definido previamente en el fichero **Provider.js**:

  ```
  const {contratacion, setContratacion} = useContext(Contexto)
  ```

  NOTA: recuperamos también el array contratacion porque lo necesitaremos en el siguiente paso

- Vamos a crear la función contratar que nos permitirá añadir rutas en el array contratacion:

  - Utilizaremos el método setContratacion para añadir el id de la ruta seleccionada al array, conservando el contenido previo y, para ello, utilizaremos el **spread operator**:

    ```
    const contratar = (id) => {
       setContratacion([...contratacion, id])
    }
    ```

    NOTA: No podemos utilizar en arrays, para los que REACT controla su estado, el método push ya que este método retorna la nueva longitud del array y, por tanto, nos cambiaría el valor del mismo por un nuevo valor no deseado. En su lugar utilizamos el operador spread **…** que lo que hace es añadir un nuevo elemento al final de los elementos que pudiera contener el array

- Y a crear la función anular que nos permitirá eliminar rutas en el array contratacion:

  - Obviamente también tendremos que utilizar el método setContratacion para eliminar el id de la ruta seleccionada al array

  - Para ello no podemos utilizar el método splice ya que este método nos retorna, al igual que el método push, la nueva longitud del array. En su lugar tendremos que confeccionar un nuevo array temporal a partir del array contratacion pero sin el id de la ruta a eliminar:

    ```
    const anular = (id) => {
       let idrutas = contratacion.filter((item) => item !== id)
       setContratacion(idrutas)
    }
    ```

    NOTA 1: Creamos un array idrutas utilizando el método filter con el array contratacion para que nos entregue todas las rutas que existan excepto aquella cuyo id coincida con la ruta a eliminar

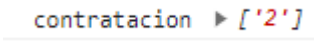
    NOTA 2: Posteriormente utilizamos este array para modificar el array contratacion utilizando su correspondiente método setter

- Ahora necesitamos que aparezca un botón u otro en función si la ruta se encuentra contratada o no. Para ello utilizaremos un IF ternario que nos compruebe si el id de la ruta que estamos mostrando en el componente de detalle se encuentra dentro del array contratación que hemos recuperado del contexto

```
<div className='botonesruta'>
        {contratacion.includes(id)
         ? <button className='rojo' onClick={() => anular(id)}>Anular</button>
         : <button className='verde' onClick={() => contratar(id)}>Contratar</button>
        }
        <button className='yellow' onClick={volver}>Volver</button>
</div>
```

Si ahora probamos la aplicación deberíamos ver como en el array se van guardando los id de las rutas para las que pulsamos '*Contratar*' y como, al pulsar sobre el botón, este cambia al botón '*Anular*'

Y en el array:

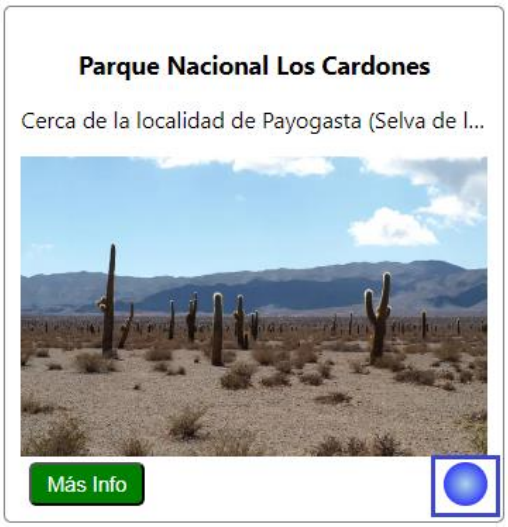


Y si, pulsamos sobre anular, volverá a aparecer el botón 'Contratar' y eliminaremos el id del array

Y en el array: `contratacion ▶ []`

4.3 Identificar rutas contratadas en la consulta de rutas por zona (Excursion.js)

Tenemos otra especificación consistente en que, cuando consultamos las rutas de cada zona, aquellas que tengamos contratadas se muestren con un circulo azul en la esquina inferior derecha la ficha que corresponda a esa ruta



Para añadir esta funcionalidad vamos a editar el fichero **Excursion.js** para comprobar si el id de la ruta al que corresponde el componente se encuentra dentro del array de rutas que obtendremos del contexto y, de ser así, añadiremos una caja redonda azul que conseguiremos con algunos estilos css:

- Recuperamos el array contratacion del contexto

  ```
  const {contratacion} = useContext(Contexto)
  ```

- Si el id de la ruta se encuentra en el array mostramos la caja azul:

  ```
  <div className='flex'>
      <Link to={`/detalle/${excursion.id}`}><button>Más Info</button></Link>
      {contratacion.includes(String(excursion.id)) &&
          <div className='circulo'></div>
      }
  </div>
  ```

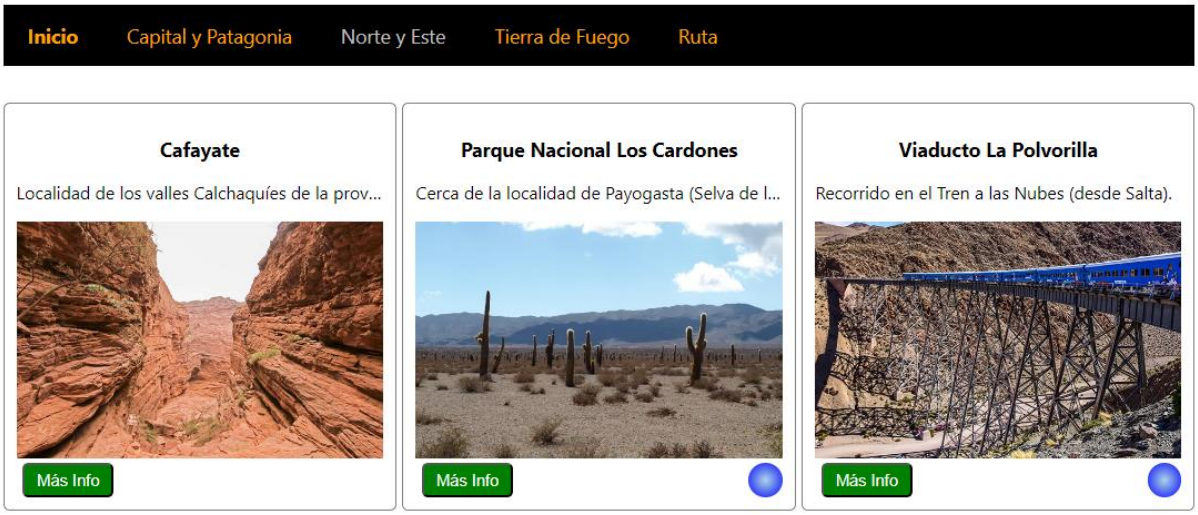
  NOTA 1: Puesto que el id que llega por el parámetro es de tipo string mientras que los id que tenemos en el array contratación son de tipo numérico, tenemos que garantizar que la comparación entre ambos se realice con el mismo tipo de dato. Para ello utilizamos la función JS String()

  NOTA 2: El operador && actúa como un if que evalúa que si la condición es cierta se ejecute el código situado a la derecha del operador

- Editamos el fichero **Excursion.css** para añadir el css necesario que nos permita visualizar una caja redonda

  .circulo {width: 28px; height: 28px; border-radius: 50%; background: radial-gradient(lightblue, blue)}

Si contratamos unas cuentas excursiones veremos como aparecen con el circulo azul en el componente de consulta



EJERCICIO 5: RESUMEN DE RUTAS CONTRATADAS

Nos falta una última e importante funcionalidad en la plataforma consistente en mostrar un resumen de las rutas contratadas y el precio total de todas ellas dentro del componente **Rutas.js**

5.1 Mostrar lista resumen de las rutas contratadas

Editaremos el componente **Rutas.js** para añadir una lista con las rutas que tengamos en el array contratación y, para cada una de ellas, informaremos:

- Una imagen de tamaño pequeño de la ruta
- Un botón para poder anular la ruta desde este componente y evitar al usuario desplazarse al componente de detalle
- La descripción de la ruta
- El precio individual de cada ruta

Las modificaciones que realizaremos serán las siguientes:

- Recuperar el array de rutas contratadas y la función de anular ruta (que asociaremos al botón anular de cada ruta) del contexto

  const {**contratacion, setContratacion**} = useContext(Contexto)

- Confeccionamos un array con todos los datos de las rutas contratadas.

  Para ello tendremos que utilizar el método filter sobre el array que contiene todas las rutas (**excursiones.js**) y, como filtro, indicaremos si el id de cada ruta de excursiones se encuentra dentro del array de rutas contratadas (contratacion) par devolver el objeto correspondiente a esa ruta:

  const destinos = excursiones.filter((item) => contratacion.includes(String(item.id)))

- Modificamos el código JSX del componente para recorrer con map el array de destinos y confeccionar una fila para cada uno de ellos y con la info solicitada

```
<h3>Lugares a visitar</h3>
{destinos.length === 0
      ? <p>(Todavia no hay actividades)</p>
      : destinos.map(item =>
       <p key={item.id}>
          <img src={`../images/${item.imagen}`} alt='imagen'/>
          <button onClick={() => anular(String(item.id))}>Anular</button>
          {item.nombre} ({item.precio}€)
       </p>
      )
}
```

NOTA : Comprobamos si el array destinos está vacio para mostrar el texto ‘ *Todavia no hay actividades*’

- Añadimos el evento onClick para cada botón de anular y que nos permitirá invocar a la función anular (observad como le pasamos el id de la ruta en formato string)
- Incorporamos la función anular utilizando el mismo procedimiento que el utilizado en el componente de Detalle

```
const anular = (id) => {
   setContratacion(contratacion.filter((item) => item !== id))
}
```

Si contratamos unas cuantas rutas y entramos en el componente de rutas deberíamos ver:

Y si pulsamos sobre los botones de anular de cada ruta deberíamos ver como desaparece de la lista y del array de rutas contratadas

5.2 Mostrar precio total de las rutas contratadas

Para mostrar el precio total de todas las rutas tenemos dos opciones: utilizar una tercera variable de contexto que mantendremos informada cada vez que añadimos u anulemos una ruta, o, para simplificar, calcular el precio a partir de las rutas contratadas que tenemos en el componente rutas

Por sencillez, y porque el documento ya tiene más de 30 páginas, vamos a utilizar la segunda opción.

- Utilizaremos una variable llamada totalViaje que mostraremos en el documento JSX:

  ```
  <h3>Total a pagar: {totalViaje} €</h3>
  ```

- Para informar la variable utilizaremos el array de destinos que ya hemos creado en el paso anterior y lo recorreremos con el método forEach de JS para ir acumulando en esta variable los precios de las rutas contratadas

  ```
  let totalViaje = 0
  destinos.forEach(item => totalViaje += item.precio)
  ```

Si probamos la aplicación de nuevo y contratamos unas cuantas rutas veremos como aparece el precio total de cada una de ellas

5.3 Imprimir o generar pdf con el resumen de rutas y el precio total

Vamos a complementar el ejercicio añadiendo una pequeña funcionalidad que nos permitirá enviar a la impresora o generar un pdf de una parte del documento.

Concretamente queremos imprimir de la pantalla de resumen la parte donde se muestra la lista de rutas contratadas, el precio y la referencia, pero no la barra de navegación, es decir, solo lo que tenemos dentro de la sección ... pero sin los botones

Lo primero que haremos será incorporar un nuevo botón dentro del componente **Ruta.js** con un evento asociado para poder imprimir la página:

```
<button className='imprimir' onClick={imprimir}>Imprimir</button>
```

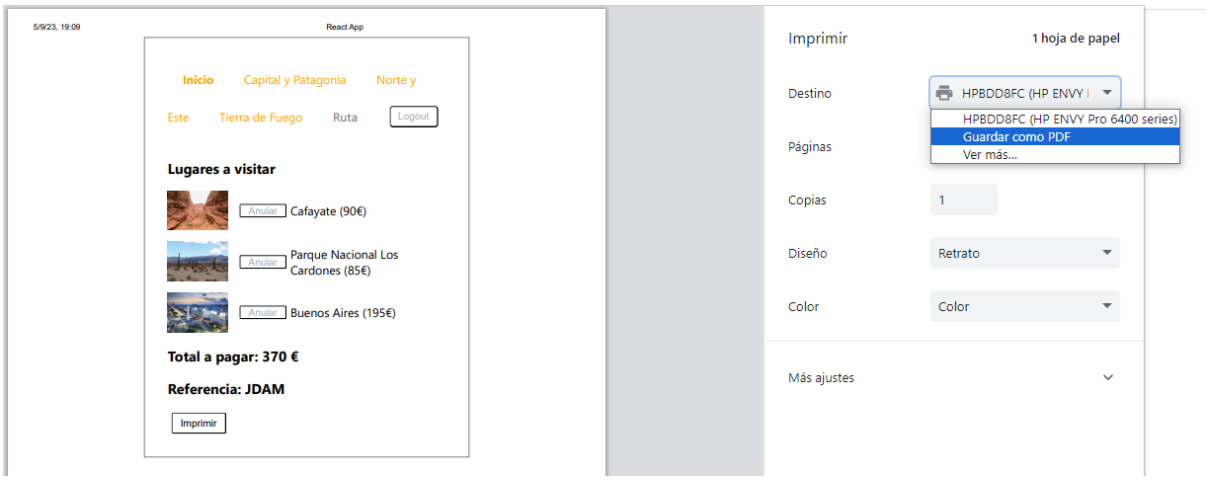
Y con un poco de css en **Ruta.css**

```
.ruta .imprimir {background-color: lightblue; color:black;padding: 5px 10px;border-radius: 3px;}
```

Creamos la función imprimir:

```
const imprimir = () => {  window.print() }
```

Si probamos a pulsar el botón veremos como Javascript nos abre el cuadro de diálogo del navegador en donde podemos escoger si imprimir o guardar en formato el pdf toda la página:



Vemos que aparecen todos los elementos de la página:

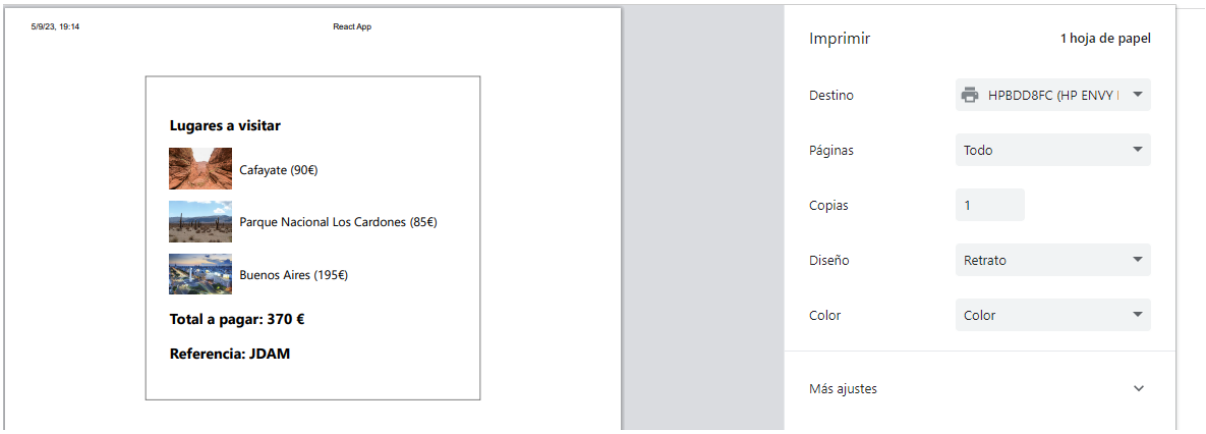
- barra de navegación
- botones de anular de las rutas
- el propio botón de imprimir

Pero nosotros solo queremos que aparezca estrictamente el resumen sin los botones ni la barra de navegación y, para ello, recurriremos al mágico css y las ***mediaqueries***:

Editamos de nuevo el fichero **Ruta.css** y añadimos la siguiente media que se activara sólo cuando intentemos imprimir la página

```
@media print {
   nav {display: none;}
   .ruta button {display: none;}
   .App {margin-top: 50px}
}
```

Y si ahora pulsamos el botón de imprimir veremos como solo aparece el resumen sin botones ni barra de navegación



EJERCICIO 6: GUARDAR LAS RUTAS CONTRATADAS EN EL STORAGE

Tal como tenemos la aplicación vemos como, mientras no refresquemos la pantalla, se van guardando las rutas en el array contratacion. Pero al volver a cargar la misma se pierden todas las rutas previamente seleccionadas

Vamos a introducir una mejora adicional consistente en guardar el array de rutas en el storage cada vez que éste se modifique (se añada o se elimine una ruta) y se informe con las rutas guardadas en el storage cuando se carga el componente por primera vez

6.0 Paso previo: eliminar el strict mode de react

Para evitar problemas con el storage tenemos que comprobar que hemos eliminado el componente **Strict mode** de react del fichero **index.js** de la siguiente forma:

```
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

6.1 Guardar rutas en el storage

Vamos a editar el fichero **context/Provider.js** que ya tenemos para añadir la operativa que necesitamos para guardar las rutas en el storage

- Necesitamos un sistema que nos asegure que el array se guarda en el storage una vez se ha modificado en las funciones contratar o anular, y ese sistema es el hook useEffect en donde nos subscribimos a los cambios del array contratacion

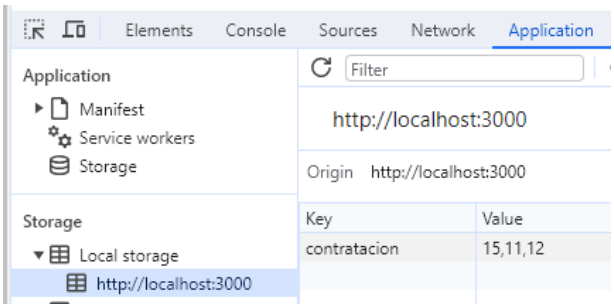
```
useEffect(() => {
    localStorage.setItem('contratacion', contratacion.join(','))
}, [contratacion])
return (
  <Contexto.Provider value={{ ... }}
)
```

- Con este hook garantizamos que se guarde el array en el storage cada vez que cambie el contenido de la variable contratacion que indicamos como segundo argumento de la función useEffect

- ¿Y por qué guardaremos el array en formato texto en el storage?. Pues sencillamente porque el storage solo admite texto plano y no podemos guardar en él ni objetos ni arrays tal como hemos visto antes para la operativa de login/logout

  NOTA:Utilizamos el método join(‘,’) para poder convertir el array a un texto con cada valor separado por una coma ya que, como ya sabemos, el storage solo admite formato texto

- Según vayamos contratando o eliminando rutas del array deberíamos ver en *application→localStorage* el fichero **contratacion** del storage con los id de las rutas contratadas:



6.2 Recuperar rutas del storage

Tamnién en el fichero **context/Provider.js** vamos a añadir la operativa para recuperar las rutas del storage e informar con ellas la variable contratacion al cargar la aplicación. Por tanto esta acción solo se tiene que realizar una vez en la carga del componente a diferencia de la operativa anterior que hay que realizarla cada vez que cambia el contenido del array

- Necesitamos un sistema que nos asegure que el contenido del storage se asigne al array contratacion al cargar la página y ese sistema vuelve a ser el hook useEffect pero, esta vez, no hay que subscribirse a ninguna variable ya que solo queremos que se ejecute una vez al cargar la aplicación

```
useEffect(() => {
    const storage = localStorage.getItem('contratacion')
    const idrutas = storage ? storage.split(',') : [];
    setContratacion(idrutas)
}, [])
```

  Recuperamos el contenido del storage y comprobamos que éste este informado (en caso de no estar informado contendrá el valor undefined que se evaluaria como un false). Si no está informado inicializamos el array contratacion como un array vacio y, en caso contrario, lo inicializamos con el contenido del storage que, previamente, tenemos que convertir a array utilizando split() (recordad que en el storage los id de las rutas se guardan en modo texto separados por comas)

Fijaos que estamos utilizando el IF ternario porque, si no existiera el fichero en el storage (por ser la primera vez que entramos a la plataforma por ejemplo) tendríamos que asignar un array vacio como valor inicial