

ALCYON IT SERVICIOS

EJERCICIO REACT VINOTECA

David Alcolea

23-4-2024

Práctica Vinoteca

Introducción

En la siguiente práctica realizaremos una aplicación de consulta de referencias de vinos.

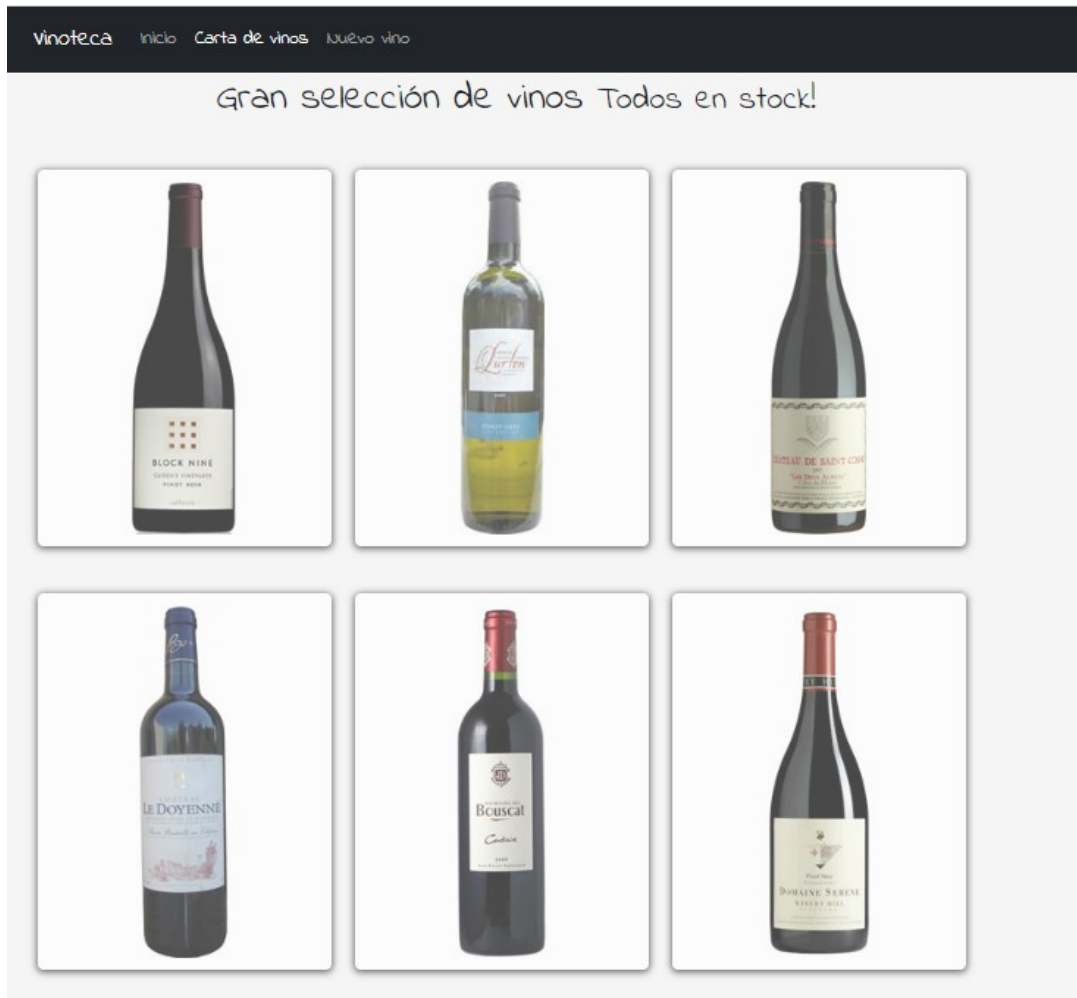
Pantalla de inicio:

Pantalla que se mostrará al entrar en la aplicación



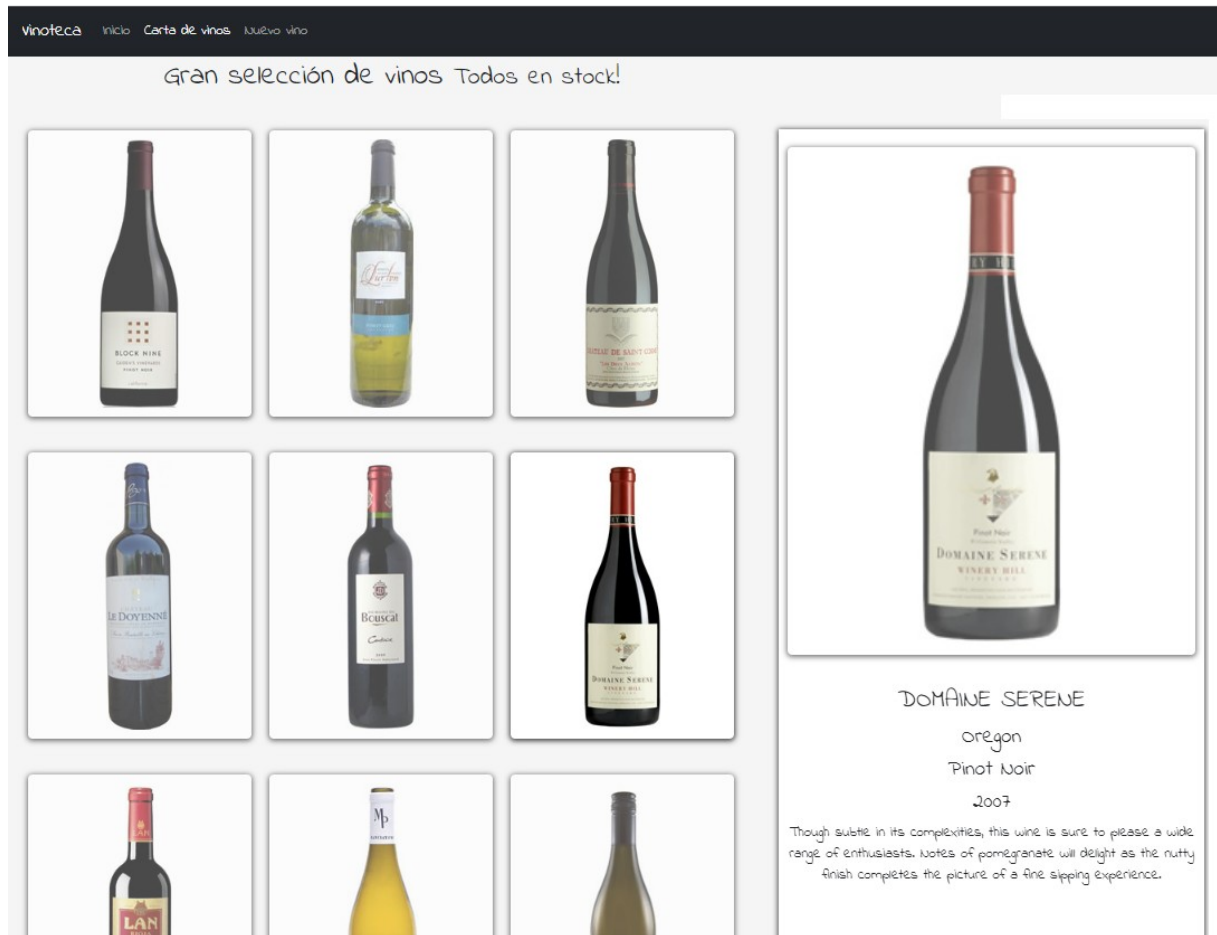
Pantalla de consulta de vinos:

Pantalla que se mostrará el pulsar sobre la opción 'Carta de vinos' de la barra de navegación:



Pantalla de detalle de vino:

El componente con la información de detalle de cada vino se mostrará cada vez que el ratón pase por encima de un vino de la carta de vinos



Pantalla de alta de vino

Pantalla que se mostrará al pulsar sobre la opción Nuevo Vino de la barra de navegación

vinoteca

[Inicio](#) [Carta de vinos](#) [Nuevo vino](#)

NOMBRE

País

Región

Uvita


Año

Imagen

Descripción

Añade vino

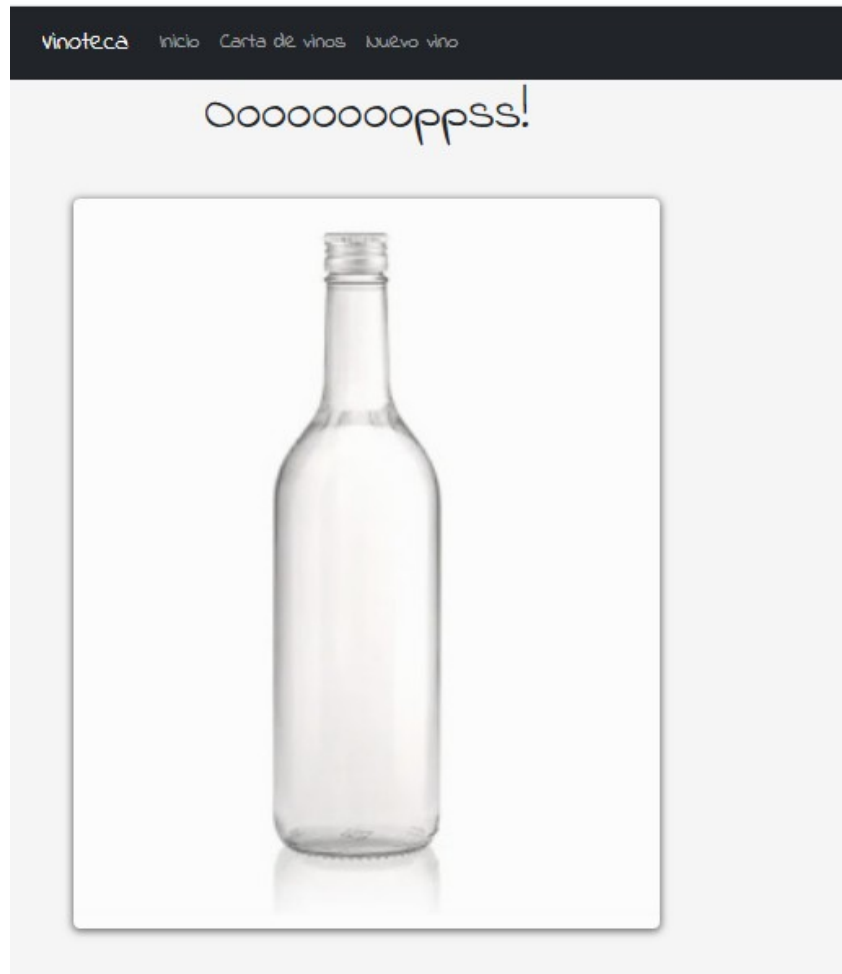
Cancelar



© Fundazapal - Práctica REACT vinoteca

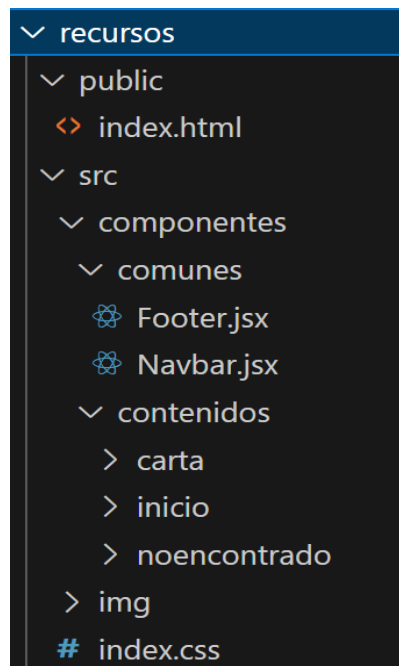
Pantalla de ruta inexistente:

Pantalla que se mostrará si la ruta que se indique en la url no existe (error 404)



Recursos del proyecto

Se proporciona un fichero llamado `recursos.zip`. En este fichero encontraréis el HTML y CSS de los componentes necesarios y la carpeta `img` con todas las imágenes que se necesitan en el proyecto



Creación del proyecto

Abrimos un terminal y nos situamos en la carpeta de nuestros proyectos. Escribimos lo siguiente:

```
create-react-app vinoteca
```

También necesitaremos el sistema de enrutado de React:

```
npm i react-router-dom
```

Librerías requeridas del proyecto

Una vez finalizado el proceso anterior, abrimos el fichero `index.html` y añadimos los CDN's para trabajar con *Bootstrap* y sus dependencias (notad que también estamos utilizando el CDN de *Google Fonts*, ya que vamos a utilizar un tipo especial de fuente llamada *Indie Flower* y la librería de JQuery):

index.html

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min
.css" rel="stylesheet" integrity="sha384-
QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH"
crossorigin="anonymous">

<link href="https://fonts.googleapis.com/css?family=Indie+Flower"
rel="stylesheet">

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtT
E1Pi6jizo" crossorigin="anonymous"></script>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundl
e.min.js" integrity="sha384-
YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDz0xhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
```

Creación de componentes comunes

Todas las páginas de la aplicación van a tener una barra de navegación en la parte superior y un pie de página en la parte inferior. Dado que estos dos componentes son omnipresentes, pues aparecen siempre, independientemente de la página en que nos encontremos, los crearemos en una subcarpeta llamada `componentes/comunes`:

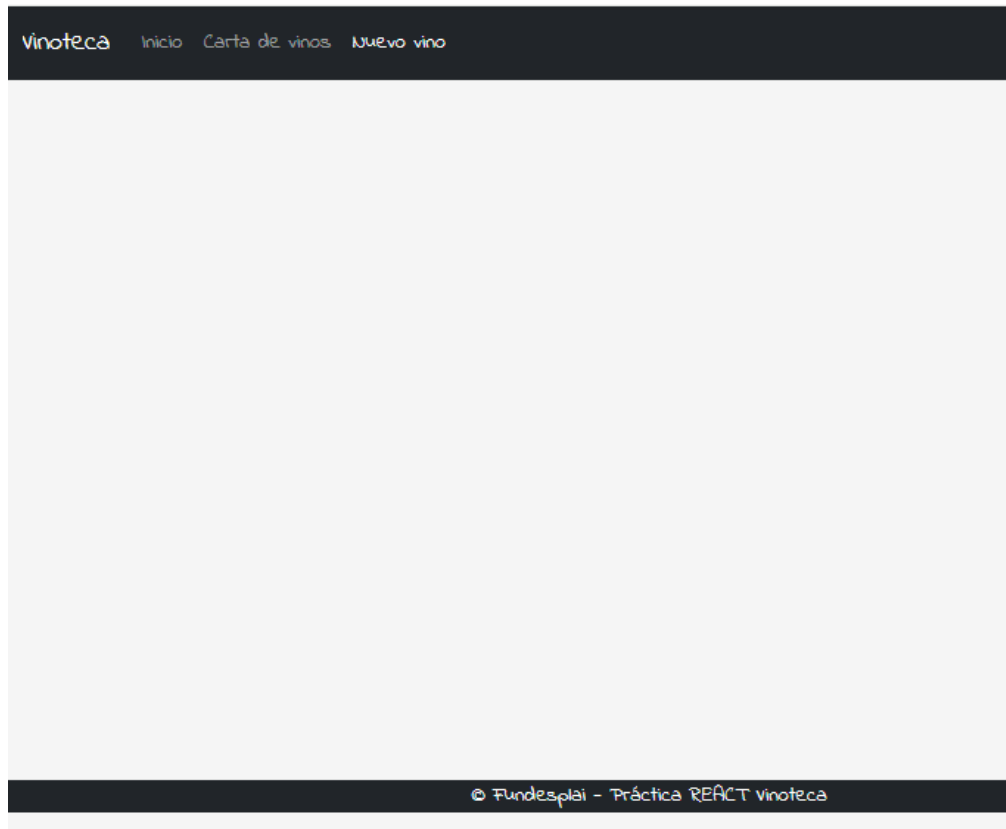
Una vez incorporados los dos componentes comunes (Navbar y Footer), para ver el resultado de lo que tenemos hecho hasta ahora, vamos a añadir a la plantilla del componente raíz de la aplicación los selectores de los dos componentes que acabamos de crear.

Abrimos el fichero `App.jsx`, y añadimos los dos componentes:

App.jsx

```
<Navbar></Navbar>  
<Footer></Footer>
```

Si ahora levantamos el servidor y nos dirigimos al navegador deberíamos ver lo siguiente:



API consulta de vinos

Para obtener los datos de los vinos utilizaremos la API:

<https://alcyon-it.com/APIS/vinoteca/index/vinos/> → para consulta de todos los vinos

<https://alcyon-it.com/APIS/vinoteca/index/vinos/id> → para consulta de un vino por su id

Por otro lado, es una buena práctica crear un fichero **api.js** dentro de una carpeta que podríamos llamar **src/entorno** para especificar valores constantes que vayamos a utilizar en todo el proyecto.

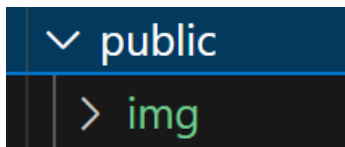
Por tanto vamos a crear el fichero **api.js** y a especificar la ruta para el acceso a nuestro *backend*:

api.js

```
const API = "https://alcyon-it.com/APIS/vinoteca/index/vinos/"
export default API
```

Imágenes a utilizar en el proyecto

Vamos a copiar la carpeta de imágenes del proyecto dentro de la carpeta **public** en lugar de en la carpeta **src** ya que nos va a facilitar mucho el acceso a las mismas y, además es la forma más correcta de hacerlo



Y, además, crearemos un fichero **img.js** dentro de la carpeta **entorno** en donde indicaremos la ruta fija de todas las imágenes a utilizar en el proyecto

img.js

```
const IMAGENES = '../img';
export default IMAGENES;
```

NOTA: La API que utilizaremos tiene también una carpeta donde se encuentran las imágenes, concretamente en:

<https://alcyon-it.com/APIS/vinoteca/assets/img/>

Si, posteriormente realizamos la operativa de alta de vinos y deseamos asignar una imagen se recomienda utilizar la API ya que en localhost no podremos guardar las imágenes a asociar al vino de forma automática

Componentes que vamos a necesitar

Nuestra aplicación va a tener las siguientes páginas (componentes):

- **Inicio:** Es la página inicial, la que aparece al acceder al servidor sin especificar ninguna ruta.
- **Notfound:** Es la página que se muestra si especificamos una URL incorrecta
- **Lista de vinos:** Es la carta de vinos. Realmente no es una página individual, pues está formada por tres componentes que se interrelacionan para conseguir la funcionalidad deseada. A saber:
 - **Vino:** Representa un vino, básicamente es una etiqueta `` con estilos CSS. No suministra ninguna información sobre el vino, tan solo la imagen enmarcada.
 - **Detallevino:** Ya que el componente anterior no proporciona información sobre el vino, cuando el usuario pasa el ratón por encima de un vino, se activa este componente para mostrar en un marco aparte la información relativa al vino actual.
 - **Listavinos:** Es el componente padre de la carta de vinos. Incluye y orquesta los dos componentes anteriores para mostrar y dar funcionalidad a la galería de vinos.
- **Altavino:** Es la página en donde podremos dar de alta un nuevo vino

Procedemos con la creación de todos los componentes.

Inicio.jsx

Carece de lógica de programación y únicamente muestra la imagen de entrada a la aplicación.

Puesto que las imágenes se encuentran en la carpeta **public**, para acceder a ellas tendremos que importar la constante que hemos creado antes en donde indicamos la ruta de las mismas

`import IMG from '../..../entorno/img'` (nota: la ruta dependerá de la estructura de carpetas que hayáis utilizado)

Y acceder a la imagen en la etiqueta `img` utilizando esta constante:

```
<img src={`/${IMG}/home.png`} alt=""/>
```

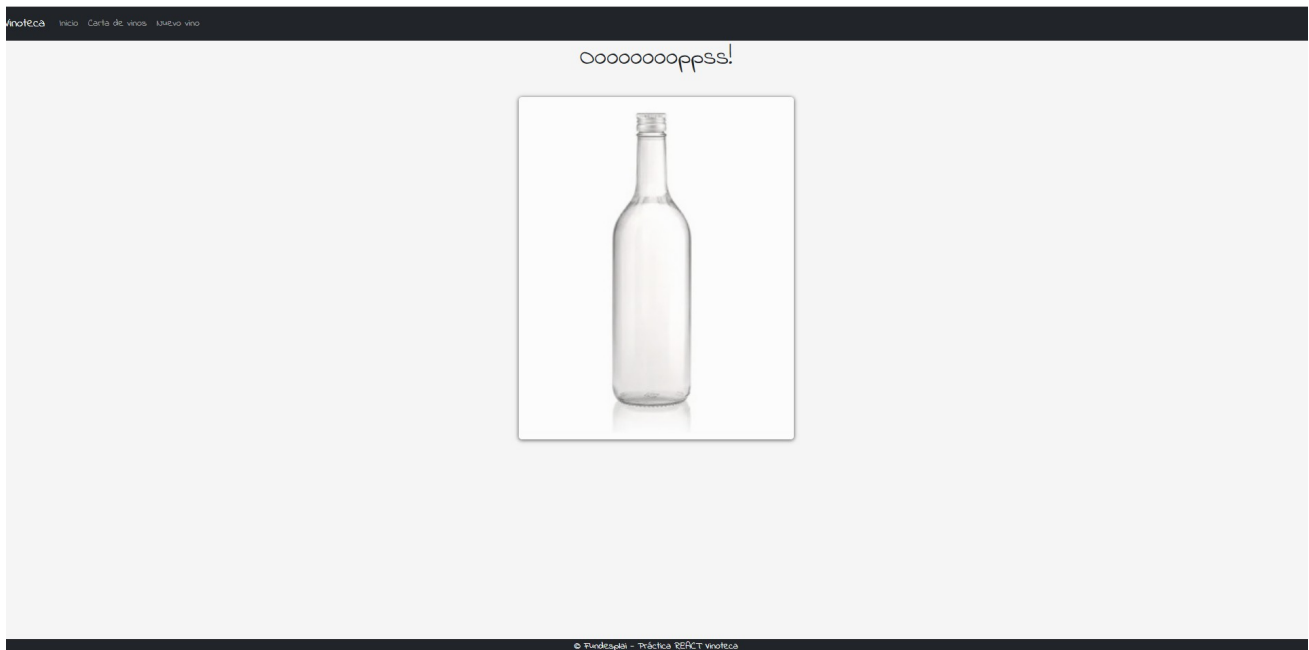
Para probarlo, inyectamos el componente en **App.js** y deberíamos ver lo siguiente:



NotFound.jsx

Al igual que en el componente de Inicio, este componente solo tendrá una imagen estática que incorporaremos utilizando exactamente el mismo procedimiento (importar constante con la ruta u utilizarla en el atributo `src` de la etiqueta `img`):

Para probarlo, inyectamos el componente en **App.js** y deberíamos ver lo siguiente:



Iniciar la carta de vinos

Esta es la parte más compleja de la aplicación, ya que tenemos que crear e integrar elementos, tales como servicios y componentes, que han de colaborar juntos para ofrecer la funcionalidad requerida.

Dado que son tres componentes los que intervienen para crear la carta de vinos, los crearemos en una carpeta específica denominada `carta`.

Comenzamos creando el componente padre **Listavinos.jsx**:

Listavinos.jsx

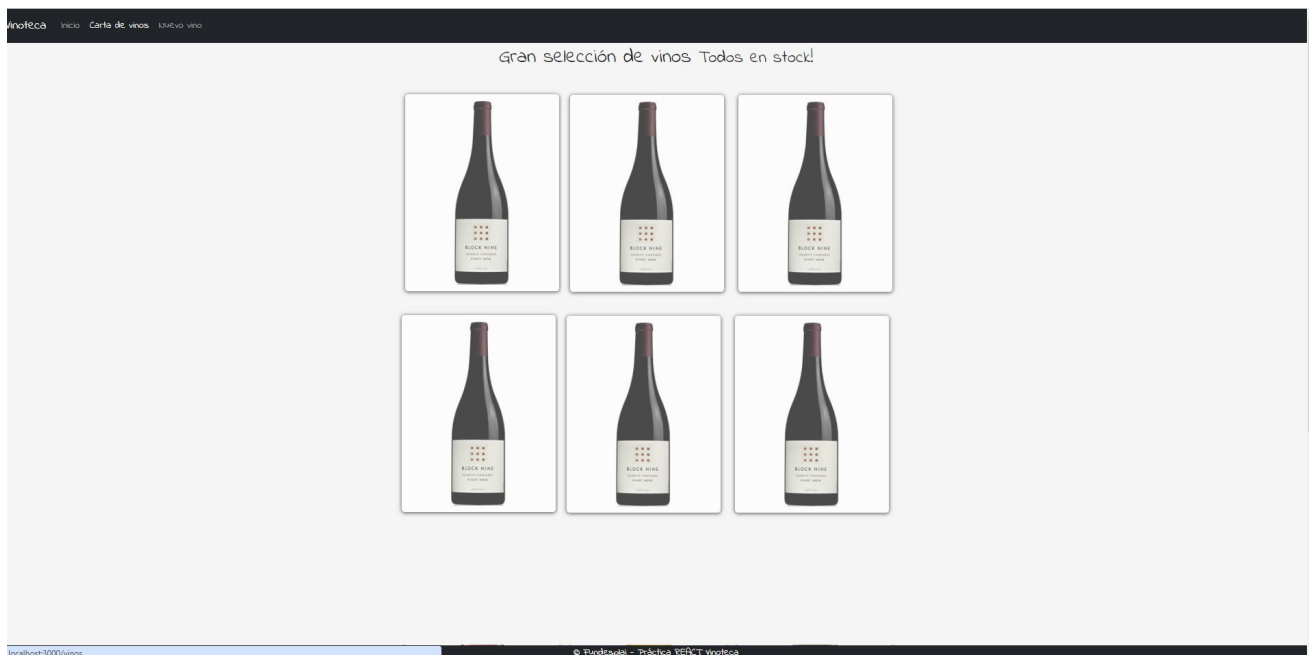
La estrategia que emplearemos será comenzar con una implementación básica, que nos permita configurar el sistema de rutas y comprobar que el servicio recupera correctamente los vinos del

backend. Cuando esto funcione pasaremos a crear los otros dos componentes y a distribuir adecuadamente responsabilidades.

Dicho lo anterior, veamos el código para el fichero de clase del componente padre, pero no sin antes hacer algunos comentarios:

- Tendremos que crear una variable de estado en donde guardaremos el array de objetos que nos va a devolver la API que vamos a utilizar
- Al cargar el componente se realizará la petición a la API comentada antes (cuya ruta tenemos en la constante que hemos creado en el fichero **api.js** en la carpeta **entorno**)
- Con la respuesta de la API informamos la variable de estado que hemos creado antes
- Inyectamos un componente `<Vino>` por cada uno de los objetos del array de respuesta de la API

De momento, para poder probar lo anterior, vamos a inyectar el componente en **App.jsx** y deberíamos ver lo siguiente



Creación de las rutas

Necesitamos establecer las rutas de la aplicación, que como se dijo serán cuatro: inicio, carta de vinos, alta de vino y página no encontrada.

Para ello incorporaremos el componente de enrutado `<BrowserRouter>` en el fichero **App.jsx**

App.jsx

Necesitaremos el siguiente grupo de rutas:

- Ruta raíz `'/'` que cargará el componente `<Inicio>`
- Ruta `'/vinos'` que cargará el componente `<Listavinos>`
- Ruta `'/alta'` que cargará el componente `<Altavino>`
- Ruta `'/*'` que cargará el componente `<NotFound>` si la ruta no existe

Finalmente, tendremos que modificar la barra de navegación para añadir los correspondientes enlaces a las rutas que hemos creado antes:

Navbar.jsx

Añadiremos las rutas para los enlaces de Inicio, Carta de Vinos y Alta de vino. Recordad utilizar el componente `<NavLink>` e incorporar el componente `<Navbar>` dentro del `<BrowserRouter>` en **App.jsx**

Si probamos la aplicación deberíamos ser capaces de navegar entre los componentes Inicio, Lista de vinos y Nuevo vino

Avanzar la construcción de la carta de vinos

En estos momentos ya tenemos los cimientos de nuestra aplicación.

Lo primero que vamos a hacer es dotar de contenido dinámico al componente **Vino.jsx**, que tendrá la responsabilidad de tratar cada vino individualmente. Recibirá los datos de cada vino de su componente padre, esto es, de **Listavinos.jsx**.

Nos podemos preguntar cuál es la ganancia de separar estas responsabilidades. La razón es que cada componente debe estar especializado en una única tarea, así **ListaVinos** debe ser el encargado de la manipulación de la lista de vinos pero no de tratar los aspectos individuales de cada uno de ellos.

Listavinos.jsx

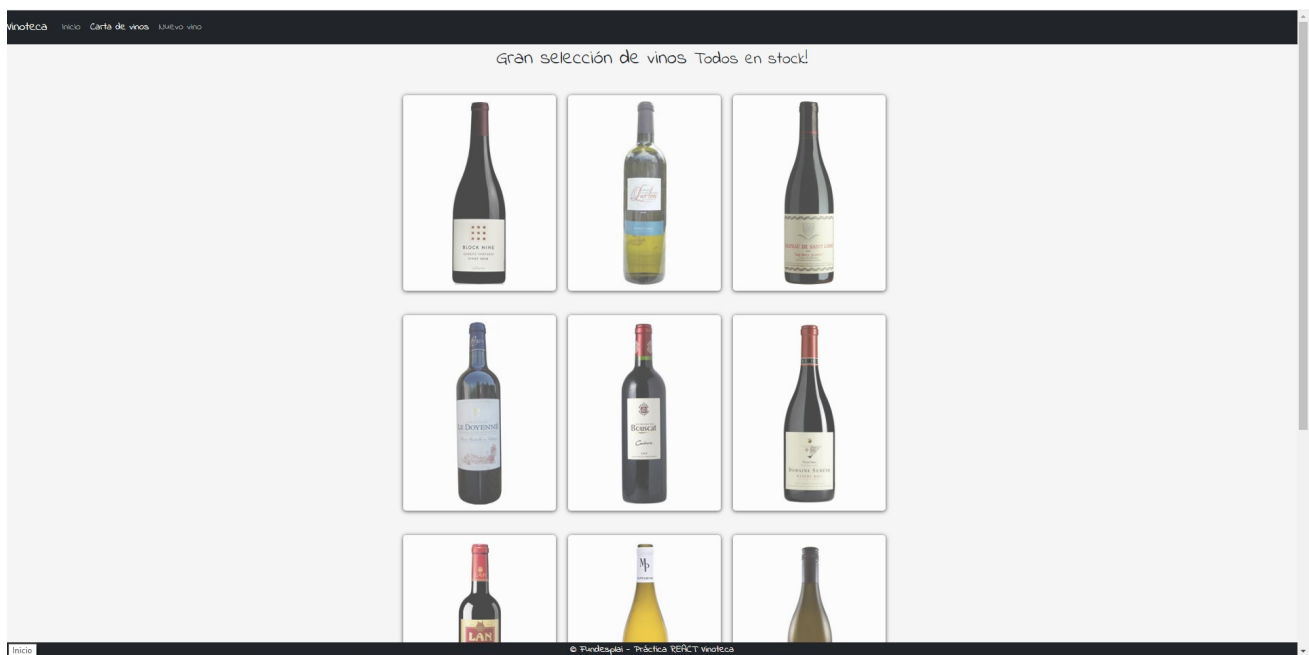
Dicho lo anterior, desde **Listavinos**, inyectamos un componente `<Vino>` para cada elemento del array que hemos obtenido en la API y pasando como parámetro cada uno de los objetos vino (recordad también indicar el atributo `key` con el id de cada vino o con el índice del array, indistintamente):

```
<Vino vino={vino} key={vino.id}></Vino>
```

Vino.jsx

En este componente recuperaremos el parámetro que nos llegará desde el componente padre e informaremos cada uno de los datos que necesitamos para cada uno de los vinos (y que básicamente son únicamente la imagen para la el atributo `src` de la etiqueta `img` el nombre para el atributo `alt`

Si ahora probamos la aplicación deberíamos ver como aparece la foto de todos los vinos



Consulta de detalle de cada vino

Ahora tenemos que añadir el componente que aparecerá en un marco a la derecha de la pantalla, a modo de zoom, cuando el usuario pase el cursor sobre alguna imagen:

Detallevino.jsx

En este componente:

- Recibiremos como parámetro en la propia función del componente el id del vino a consultar (y que enviaremos desde el componente padre **Listavinos.jsx** más adelante)

- Necesitamos una variable de Estado para guardar el objeto del vino que consultaremos en la API que tenemos en nuestra variable de entorno
- Al cargar el componente, y cada vez que cambie el valor del id del vino a consultar, realizaremos la petición a la API pasando por la url el id del vino a consultar (y que todavía no tenemos)
- Con la respuesta informamos la variable de estado que hemos creado antes
- Con esta variable informamos los datos del vino

Si, por cualquier motivo, el id del vino no existe en la base de datos, la API nos devolverá un status 404 que tendríamos que comprobar en el primer `.then()` de respuesta del servidor.

```
If (resp.status == '404') {throw('Vino no existe')}
```

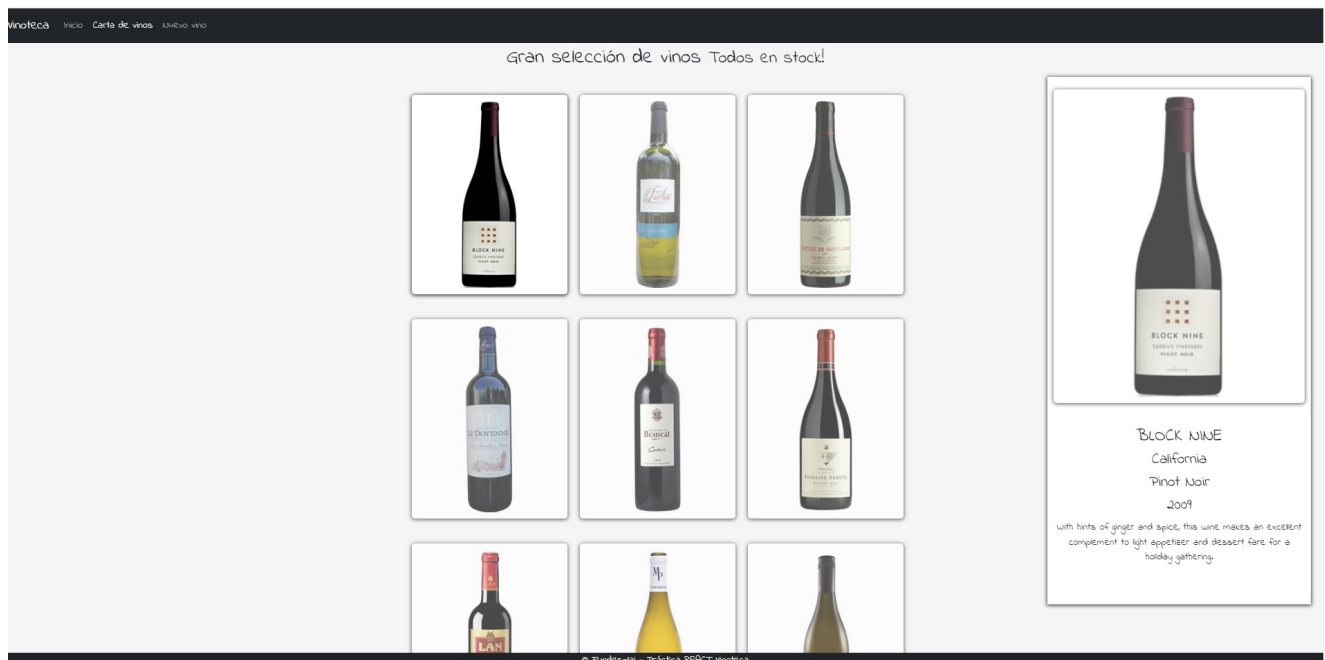
Listavinos.jsx

En este componente inyectaremos el componente anterior para que aparezca a la derecha de nuestra carta de vinos.

De momento, y para probar que funciona, enviaremos como parámetro el id de un vino cualquiera:

```
<Detallevino id=1></Detallevino>
```

Y deberíamos ver:



Pero necesitamos que este componente solo se muestre cuando el cursor pase por encima de algún vino y, además, que se muestren los datos del vino seleccionado.

Para ello necesitamos de algún mecanismo que nos permita:

- Detectar cuando el cursor pasa por encima de un vino para mostrar el componente y ocultarlo cuando el cursor salga de la imagen del vino.
- Guardar el id del vino a consultar cuando el cursor entra en la imagen

Estas tareas la tendremos que realizar en el componente **Vino.jsx** ya que es aquí donde podemos detectar los eventos del cursor y conocer sobre que vino se encuentra éste y, para ello, necesitaremos utilizar el Contexto ya que tendremos que crear dos variables para:

- Indicar si se tiene que mostrar el componente de detalle o no
- Guardar el id del vino a consultar

Contexto.jsx

Crearemos el fichero de contexto dentro de la carpeta **contexto** para compartir las siguientes variables (y sus correspondientes setters):

- **detalle**: si contiene el valor `false` el componente de detalle estará oculto y si contiene el valor `true` se mostrará
- **id**: para guardar el id del vino a consultar

Vino.jsx

En este fichero añadiremos las siguientes modificaciones:

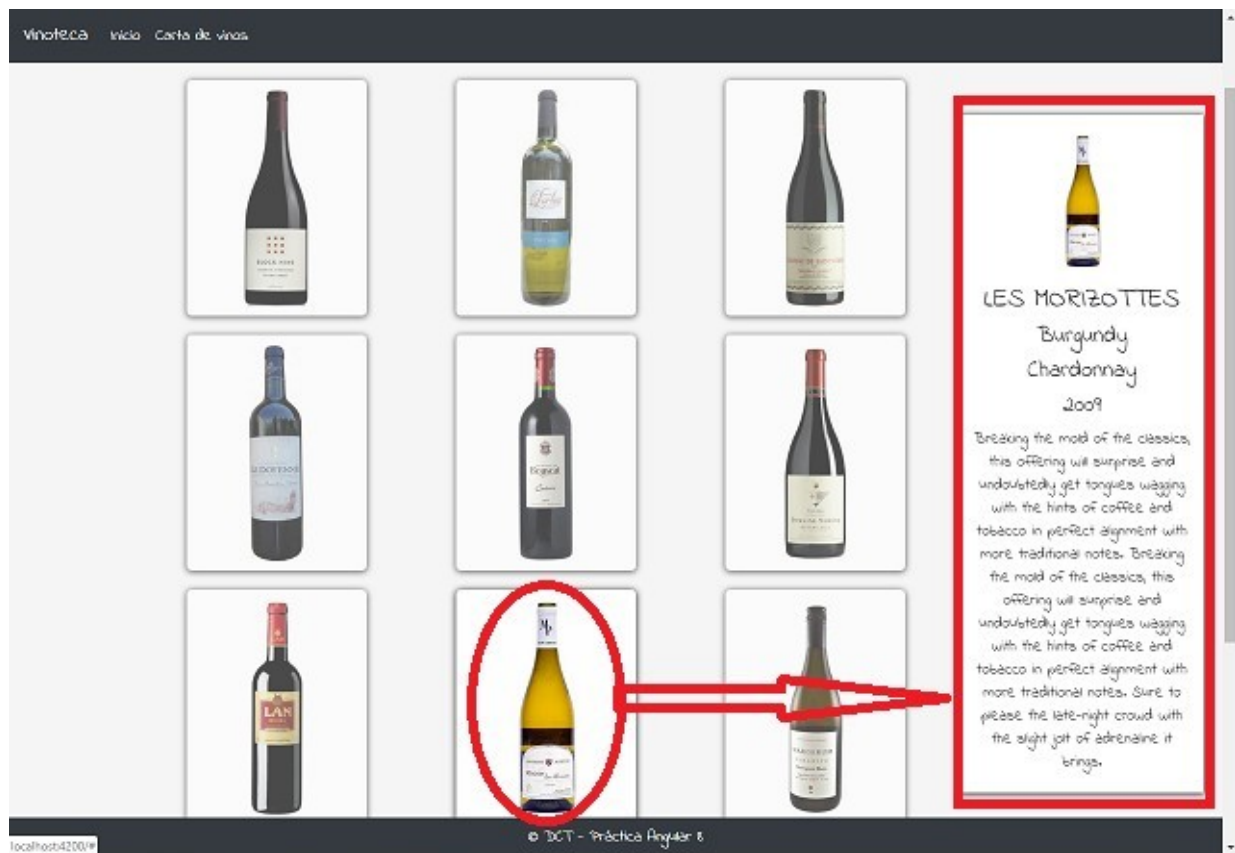
- Recuperemos los setters de las dos variables de contexto
- Activamos un evento `onMouseEnter` sobre la imagen (o su caja contenedora) que ejecutará una función que informe a `true` la variable **detalle** e informe el id del vino a consultar en la variable de contexto `id` (recordad que a este componente nos llega como parámetros de la función el objeto vino que enviamos desde el componente padre **Listavinos**)
- Activamos un evento `onMouseLeave` sobre la imagen (o su caja contenedora) que ejecutará el setter de **detalle** para informarlo con el valor `false`

Listavinos.jsx

Ahora veamos las modificaciones que necesitamos en el componente padre para hacer posible que cuando el usuario pase el cursor por la imagen de un vino, el componente padre sepa qué vino es y se lo pase al componente que hace el zoom:

- Necesitamos recuperar del contexto las variables de **detalle** e **id**
 - Si la variable `detalle` contiene el valor `true` mostramos el componente `<Detallevino>` pasando como parámetro el `id` que obtenemos de la variable de contexto `id`
- ```
{ detalle && <Detallevino id={id}></Detallevino> }
```

Deberíamos ver lo siguiente en el navegador:



## Alta de vino

AL acceder a la opción de 'Nuevo vino' accederemos al formulario de alta en donde podremos añadir una nueva referencia a nuestra carta

Vinoteca

Inicio

Carta de vinos

Nuevo vino

NOMBRE

NOMBRE

País

País

Región

Región

Uvas

Uvas

Año

Año

Imagen


Seleccionar archivo

Ningún archivo selec.

Descripción

Alta vino

Limpiar



© Fundesplai - Práctica REACT Vinoteca

Para ello necesitaremos instalar el hook de formularios con:

`npm i react-hook-form`

E incorporar todos los hooks que necesitaremos:

```
const {register, handleSubmit, formState: {errors}, setFocus, reset} = useForm()
```

Los recordamos:

**register** → Array con todos los valores del formulario

**handleSubmit** → Método donde indicaremos la función que se encargará de recoger los datos

**formState** → Hook para validar los datos del formulario

**setFocus** → Hook para poder situar el foco en un control del formulario.

**reset** → Hook para poder resetear el formulario.

El evento que tendremos que utilizar para recoger los datos en la función que asociaremos al método **handleSubmit** tiene que ser obligatoriamente el **onSubmit()**

```
<form onSubmit={handleSubmit(recogerDatos)}>
```

Asignaremos, en cada control de formulario, el nombre del dato a recuperar de cada uno de ellos y, para ello, utilizamos el hook **register** con el spread operator (para asegurarnos que se vayan añadiendo todos los datos al array)

```
<input type="text" placeholder="NOMBRE" autoFocus {...register('nombre')}/>
```

NOTA: Para el primer control correspondiente al nombre también situaremos el foco al cargar el componente utilizando el hook **autoFocus**

Y, por último, añadimos las validaciones a cada control utilizando el hook **useState**. Ejemplo para el control nombre:

```
<input type="text" placeholder="NOMBRE" autoFocus {...register('nombre',{required: true,
maxLength: 30})}/>
```

```
{errors.nombre?.type === 'required' &&
```

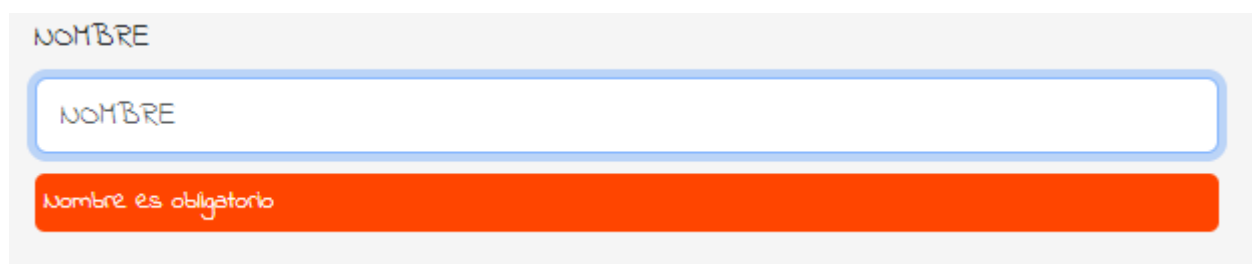
```
 <div className='errores'>Nombre es obligatorio</div>
```

```
}
```

```
{errors.nombre?.type === 'maxLength' &&
```

```
 <div className='errores'>La longitud máxima no puede superar 30 caracteres</div>
```

```
}
```



The screenshot shows a web form with a light gray background. At the top, the word 'NOMBRE' is written in a light gray font. Below it is a text input field with a blue border and the placeholder text 'NOMBRE'. Underneath the input field is a red rectangular box containing the text 'Nombre es obligatorio' in white, indicating a required field error.

## Recoger los datos del formulario

Necesitamos recoger los datos del formulario para enviarlos al servidor utilizando la API:

<https://alcyon-it.com/APIS/vinoteca/index/vinos/>

Para ello recuperamos los datos del formulario dentro del parámetro de entrada de la función que hemos asociado al hook `handleSubmit`:

```
const recogerDatos = (datosFormulario) => { ... }
```

EL parámetro de entrada es un objeto con la siguiente estructura:

(index)	Value	
nombre	'El baturrico'	
pais	'España'	
region	'Murcia'	
uvas	'Tempranillo'	
anyo	'2022'	
imagen		File
descripcion	'Vino de mesa ideal...	

Y, para enviar los datos al servidor necesitaremos un objeto especial JavaScript llamado FormData:

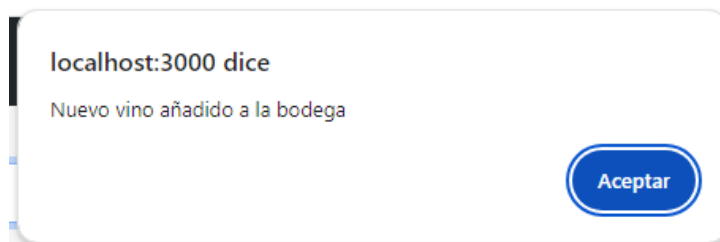
```
let datos = new FormData()
```

```
datos.append('nombre', datosFormulario.nombre)
datos.append('anyo', datosFormulario.anyo)
datos.append('uvas', datosFormulario.uvas)
datos.append('region', datosFormulario.region)
datos.append('pais', datosFormulario.pais)
datos.append('imagen', datosFormulario.imagen[0].name)
datos.append('descripcion', datosFormulario.descripcion)
datos.append('imagenbmp', datosFormulario.imagen[0])
```

El método a utilizar para el alta será el POST por lo que tendremos que crear el objeto `parámetros` para la petición a realizar con `fetch()`

```
const parametros = { method: 'POST', body: datos }
```

Y la respuesta esperada del servidor es un JSON. Si es correcta mostraremos una alerta



NOTA: La API no está preparada para recoger la imagen que asociemos al vino. Por lo tanto, tenemos que asegurarnos que la imagen a asociar la tengamos dentro de la carpeta **public/img** del proyecto REACT

Sería interesante también que, si el alta es satisfactoria:

- Situar el cursor de nuevo en el campo nombre con el hook `setFocus()`
- Limpiar el formulario con el hook `reset()`

## Previsualizar imagen

Una mejora significativa de la aplicación consistiría en previsualizar la imagen que seleccionamos para asociarla al vino

Para ello:

- Confeccionamos una caja a la derecha del formulario en donde mostraremos la imagen seleccionada

```
<div className='column col-4'>

</br>

</div>
```

- Puesto que tendremos que cambiar en la etiqueta `<img>` el atributo `src` con la imagen a previsualizar necesitaremos utilizar el hook `useRef()`

```
const previo = useRef()
```

- Necesitaremos detectar cuando el usuario selecciona una imagen desde el formulario y, para ello, utilizaremos el evento `onChange()`

```
<input type="file" className="form-control" {...register('imagen')} onChange={previsualizar}
accept='image/*' />
```

- Y confeccionar la función que se encargará de la previsualización de la imagen

```
const previsualizar = (ev) => { ... }
```

- En esta función:

- Recuperamos la imagen seleccionada desde el objeto `event.target`:

```
const imagen = ev.target.files[0]
```

- Si no hay ningún archivo seleccionado mostraremos la imagen por defecto `sinbotella.jpg`:


```
if (!imagen) {
 previo.current.src = "../img/sinbotella.jpg";
 return;
}
```

- Convertimos la imagen a un objeto de tipo `objectURL`

```
const objectURL = URL.createObjectURL(imagen);
```

- Y lo enviamos a la etiqueta `<img>` de la caja situada a la derecha

`previo.current.src = objectURL;`

NOMBRE	<input type="text" value="NOMBRE"/>	
País	<input type="text" value="País"/>	
Región	<input type="text" value="Región"/>	
Uvas	<input type="text" value="Uvas"/>	
Año	<input type="text" value="Año"/>	
Imagen	<div><div>Seleccionar archivo</div><div>lurton-pinot-gris.jpg</div></div>	

NOTA: SI el alta de vino es correcta deberíamos volver a mostrar la imagen por defecto con `previo.current.src = "../img/sinbotella.jpg";`

## Baja de vino

Vamos a añadir una última funcionalidad para poder dar de baja referencias de vinos desde el componente de carta de vinos



NOTA: Los vinos que ya se encuentran por defecto en la aplicación no se borrarán realmente de la base de datos y veremos como, al cargar de nuevo el componente, vuelven a aparecer. Solo los vinos que se añadan nuevos si que se borrarán físicamente de la base de datos

### Modificación en componente carta de vinos

Modificaremos el componente **Vino.jsx** para añadir una imagen en la esquina superior derecha de cada vino de forma que, al pulsar sobre ella, se realice una petición **fetch** al servidor para eliminar el vino seleccionado

```
<div onMouseEnter={consultaDetalle} onMouseLeave={() => setDetalle(false)} className='vino'>

</div>
```





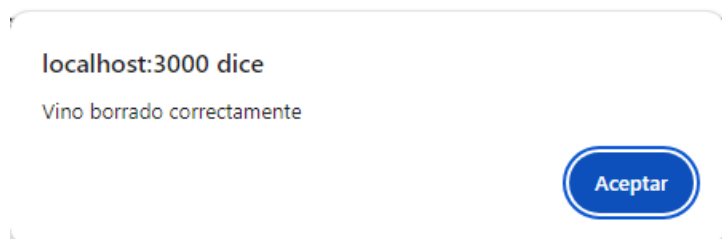
NOTA: Situaremos la imagen en la esquina superior derecha utilizando un posicionamiento absoluto. Asignamos un evento `onClick` sobre la imagen de la papelera para ejecutar la función en donde realizaremos la petición al servidor

```
const eliminarVino = () => { ... }
```

Y, en esta función:

- Utilizaremos la API que ya tenemos para la consulta y el alta pero en este caso, para la baja, tendremos que pasar como parámetro de la misma el id del vino a dar de baja. Ejemplo: <https://alcyon-it.com/APIS/vinoteca/index/vinos/4>
- Necesitaremos también el objeto con los parámetros de la petición ya que el método a utilizar en este caso será el DELETE:  

```
const parametros = { method: 'DELETE' }
fetch(API + '/' + vino.id, parametros)
```
- La respuesta del servidor será un JSON y, si todo a ido correctamente, mostraremos un mensaje de baja efectuada



Si probamos la baja veremos que, aunque aparezca el mensaje de baja efectuada, el vino realmente no desaparece de la página hasta que no volvemos a recargar el componente. Esto habría que corregirlo ya que no podemos mantener en la pantalla una referencia que ya no existe en la base de datos. Para ello:

- Recuperemos en la función el objeto `event` que se genera con el evento `onClick`  

```
const eliminarVino = (ev) => { ... }
```
- Si la baja se ha realizado correctamente, eliminaremos el nodo que corresponde al vino seleccionado:
  - Buscamos la caja contenedora que corresponde al nodo a borrar (recordad que el objeto `event.target` señala la etiqueta `img` con la papelera)  

```
const aBorrar = ev.target.closest('.vino')
```

- Desde la caja contenedora accedemos al padre (con `parentNode`) y desde éste borramos el nodo hijo:

```
aBorrar.parentNode.removeChild(aBorrar)
```

## Indicadores autoevaluación

### **Creación componentes**

Componentes comunes → 2pt

Componentes contenido → 6pt

### **Creación rutas**

Rutas barra de navegación → 3pt

Ruta página no encontrada → 1pt

### **Contenido**

Contenido correcto Inicio → 1pt

Contenido correcto página no encontrada → 1pt

Contenido correcto carta de vinos → 6pt

Contenido correcto detalle de vino → 4pt

Contenido correcto alta de vino → 1pt

### **Carga dinámica de componentes**

Carga dinámica componente con la imagen de cada vino → 1pt

Carga dinámica componente detalle de vino → 4pt

### **Alta de vino**

Alta de vino con todos sus datos → 6pt

Previsualización de la imagen → 2pt

**TOTAL** → 38pt