

# Construyendo una tienda...

## Ejercicio 1 – Integrando nuestro contador

Empezaremos utilizando el componente `Counter` que hemos utilizado en el tutorial de ejemplo. Como vamos a implementar una tienda online, utilizaremos ese componente para determinar el número de productos que queremos añadir a nuestro carrito. Pero para ello será necesario hacer unos cuantos retoques...

### Paso 1

Crear en nuestra carpeta de `components` una subcarpeta que llamaremos `counter` donde crearemos el fichero `Counter.tsx` que contenga el comportamiento del contador visto previamente.

Haciendo uso de los componentes de estilos de material que hemos preparado, nos debería quedaría algo similar a lo siguiente...

```
import { Button, Typography } from '@mui/material';
import React, { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);

  const onIncrementCount = () => {
    setCount(count + 1);
  };
};
```

```
    return (  
      <div>  
        <Typography>Current value: {count}</Typography>  
        <Button onClick={onIncrementCount}>Increment Count</Button>  
      </div>  
    );  
  };  
  
  export default Counter;
```

## Paso 2

El siguiente paso será verlo en funcionamiento. Una vez creado el componente vamos a incluirlo en alguna de nuestras pantallas para poder visualizarlo. Podemos utilizar nuestro componente `Products` para invocar al componente `Counter` y así verlo en la página principal de la aplicación.

## Paso 3 (Opcional)

Añadir estilos a nuestra pantalla de productos. Podemos mejorar la imagen de nuestra pantalla de productos añadiendo unos cuantos estilos. Para ello, podemos crear un fichero `products-styles.ts` en la carpeta de `screens/products` y definir ahí unos cuantos estilos para hacer más bonita e usable la página.

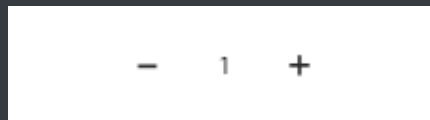
👀 Echa un ojo al componente `<NoScreen />` para ver como aplicar estilos css.

## Paso 4

Añadir un segundo botón que nos permita hacer la acción inversa, es decir, que nos permita reducir el valor del contador. Una vez creado el botón, añadir la lógica necesaria para evitar números negativos y evitar reducir el valor del contador por debajo de 0.

## Paso 5

Haz unos últimos retoques y aplica unos cuantos estilos a nuestro contador para que se vea algo similar a esto:



Y con eso, ya tendríamos nuestro componente listo!!

Una vez terminado debería verse algo como esto...



## Ejercicio 2 – Lista de productos

Este ejercicio consistirá en llenar la pantalla de productos con datos. Para ello realizaremos una petición HTTP para obtener los datos a mostrar y crear un componente que nos permita visualizarlos en pantalla.

### Paso 1

Crearemos un nuevo servicio que se encargue de realizar la llamada de red para obtener los datos. Para ello, en nuestra carpeta `services` empezaremos creando un nuevo fichero que llamaremos `ProductsService.ts` y que se tendrá que ver algo así...

```
import axios from 'axios';

const ProductsService = {
  getProducts: () => {
    return axios.get('https://raw.githubusercontent.com/RoiDopazo/react-tuto/develop/data.json');
  }
};

export default ProductsService;
```

Básicamente, estamos definiendo un método dentro de `ProductsService` (`getProducts`) que se encarga de utilizar `axios` para realizar la petición de red al endpoint especificado.

### Paso 2

En este paso trataremos de invocar el método que acabamos de crear. Como vimos anteriormente en la guía, el hook `useEffect` se usa normalmente para realizar peticiones asíncronas como son las llamadas de red. En este paso, crearemos un `useEffect` en nuestro `Products.tsx` que invoque dicho método y nos muestre en la consola los datos obtenidos.

Al tratarse de una llamada asíncrona, necesitamos resolverla como tal. Echa un vistazo a como manejar funciones asíncronas en JavaScript en este [link](#)

```
// Dentro de nuestro useEffect podemos declarar una funcion asíncrona de
la siguiente forma
const getAsyncProducts = async () => {
  log(1);
  const a = ProductService.getProducts();
  log(2);
  // Aquí tendríamos los valores retornados para la {Funcion a invocar}
  y ya los podríamos imprimir por pantalla.
}

// Una vez definida nuestra función, solo tendríamos que invocarla para
que se ejecute.
getAsyncProducts();
```



Recuerda lo visto sobre `useEffect` y su segundo argumento (el array de dependencias). Cuántas veces nos interesa realizar esta petición? Cada vez que se produzca un re-renderizado? o solamente una vez?

### Paso 3

Ahora somos capaces de mostrar nuestros productos por consola, el siguiente paso consistirá en guardarlos en un estado de React. Al guardarlos en un estado estaremos diciéndole a React que se repinte automáticamente en cuanto modifiquemos ese estado.

Empezaremos creando un estado en nuestro `Products.tsx` que podremos llamar... `products` e inicializarlo por defecto como una lista vacía `[]`. Ahora, al terminar la petición web, en vez de mostrar los datos por consola, podemos almacenarlos en este nuevo estado y ver que se guardan correctamente.

Añade un `console.log` justo antes del `return` que imprima por consola el valor de dicho estado. Si todo va bien... debería mostrar dos logs:

- Primer renderizado -> `products = []`
- Segundo renderizado -> `products = [{...}]`

### Paso 4

Ummm... los datos de los productos tardan unas milésimas de segundos en llegar, sería muy interesante poner un `loader` mientras no tenemos los datos. Vamos con ello.

Para empezar... necesitaremos un componente de `loader`, algo que nos indique que está cargando. Tenemos dos opciones, podemos crearlo nosotros mismos y hacerlo a nuestra idea y semejanza o, ya que hemos introducido Material UI, buscar en la documentación si existe algún componente que nos ofrezca lo que buscamos (os adelanto que si que existe) => <https://mui.com/getting-started/usage/>

Una vez tengamos nuestro componente de `loader` vamos a mostrarlo en la pantalla y ver como nos queda... podemos mostrarlo justo debajo del `Counter` en nuestra pantalla de `Products`

Se tendría que ver algo así...



Perfecto! Ahora solo nos faltaría mostrarlo solo cuando no tenemos productos... Muy fácil! Con un simple `if` podemos hacer que se muestre una cosa u otra en función de nuestra variable `products` .

📌 La forma más sencilla sería comprobando su 'length'

Una vez implementado el `if` el flujo de React debería quedarnos más o menos así.

- Primer renderizado -> `products = []` -> mostramos únicamente el loader
- Segundo renderizado -> `products = [{...}]` -> mostramos la página de productos tal cual la teníamos (con el título, el counter y el botón)

## Paso 5

Tenemos el `loader` listo para mostrarse mientras no tenemos los datos, el siguiente paso consistirá en mostrar la lista de productos y nos preguntaremos... Cómo queremos que se vean? Qué datos voy a mostrar? Lo primero que debemos hacer es investigar un poco los datos que nos llegan y pensar un par de ideas para mostrarlo. Las soluciones típicas suelen ser en forma de `cards` o como elementos de una `lista`. Fuese lo que fuese la idea que tengas en mente, crea un componente (dentro de la carpeta `components` si, puesto que es algo que vamos a reusar) que nos sirva para mostrar un `product` (podemos llamarle `ProductItem`).



Recuerda que estamos creando un componente lo más genérico y reutilizable posible. Si queremos mostrar en él el nombre del producto o su precio, la mejor forma sería pasándolo a través de las props.

Podemos probar cómo se vería ese componente llamándolo desde nuestro componente de `Products`.

```
return (  
  ...  
  { /* Podemos pasarle los datos de uno de nuestros productos a través de  
    las props */}  
  <ProductItem item={products[0]} />  
  ...  
);
```



Una posible forma de mostrar nuestros productos es la siguiente

## Página de productos



## Paso 6

Una vez tenemos un item creado solo nos faltaría preparar un bucle que recorra toda la lista de productos y los renderice en pantalla. Agregar este bucle es muy sencillo y lo podemos hacer dentro del propio `return` de `Products`.

Lo ideal sería crear un `div` que nos sirva de "contenedor" para todos los ítems tal que así:


```
return (  
  ...  
  <div className={classes.productsContainer}>  
    ...  
  </div>  
  ...  
)
```

Dentro del `div` podemos utilizar lenguaje JavaScript (recuerda, esto es JSX, no HTML). Para poder invocar sentencias JavaScript debemos incluirlas entre `{}`. Para recorrer la lista utilizaremos la función `map` de JavaScript ([documentación](#)). Esta función nos permitirá iterar nuestro array y crear uno nuevo con el componente que queremos utilizar, veamos cómo:

```
return (  
  ...  
  <div className={classes.productsContainer}>  
    {products.map((item) => {  
      return <ProductItem product={item} />;  
    })}  
  </div>  
  ...  
)
```

Con todos los productos debería quedarnos algo así...


Página de productos



Tortillas de trigo Wraps 6 unidades

2.69€


AÑADIR A LA CESTA



Sazonador para fajitas suave

1.15€


AÑADIR A LA CESTA



Pot japanese miso

1.95€


AÑADIR A LA CESTA



Vinagre de arroz

2.49€

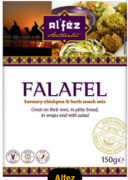
AÑADIR A LA CESTA



Dim Sum gyoza de carne

2.85€


AÑADIR A LA CESTA



Falafel estilo libanés

3.05€


AÑADIR A LA CESTA



Salsa yakitori tradicional japonesa

2.49€


AÑADIR A LA CESTA



Arroz tres delicias

3.1€


AÑADIR A LA CESTA



Fideos sabor curry

0.9€


AÑADIR A LA CESTA



Salsa al estilo marroquí con un sabor dulce y a hierbas

3.7€


AÑADIR A LA CESTA



Bebida de malta líquida con gas

1.79€

AÑADIR A LA CESTA



Salsa wok

4.15€

AÑADIR A LA CESTA

## Ejercicio 3 – Creando la página de detalles

### Paso 1

El pimer paso consistirá en enviar los datos del producto a la pantalla de detalles. Actualmente, en el componente `Products` tenemos un botón que nos permite viajar a la página de detalles pero ahora necesitamos que esa funcionalidad se aplica a cada producto que tenemos. Por ello, es necesario mover esa lógica al componente `ProductItem` que simboliza cada uno de los ítems.

Añade un `onClick` al elemento que cuando lo clickemos nos lleve a la página de detalles (puede ser el div de toda la carta o solo la imagen...) y ejecuta el `navigate` para cambiarnos de página.

### Paso 2

El siguiente paso consistirá en enviar los datos del `item` que hemos clicado a componente de detalles. La documentación de React Router nos muestra unos ejemplos de como podría hacerse:

Passing data using navigate function.

```
let navigate = useNavigate();  
navigate("/users/123", { state: partialUser });
```

And on the next page you can access it with `useLocation` :

```
let location = useLocation();  
location.state;
```

Comprueba con un `console.log` que los datos recibidos son los correctos y corresponde al item que acabas de clickar.

### Paso 3

Crea una interfaz para mostrar los datos del producto.

Debe quedar algo así...



#### Old El Paso

Tortillas de trigo Wraps 6 unidades  
350g

**2.69€**

#### Ingredientes

Harina de trigo, agua, estabilizante: glicerina; aceite refinado de girasol, emulgente: monoglicéridos y diglicéridos de ácidos grasos; dextrosa, gasificantes: carbonato ácido de sodio, difosfatos; sal

#### Conservación

Conservar en lugar fresco y seco. Una vez abierto, cerrar el envase, refrigerar y consumir antes de 3 días. Puede congelarse en casa. Descongelar completamente antes de usar

## Paso 4

Pues ya tenemos lista la pantalla de detalles pero... que pasa si un usuario abre la ruta `/details` directamente en su navegador. No tenemos los productos!!

Lo que sucede es que nos va dar un error porque el producto que usamos nosotros es `undefined` puesto que a nosotros nos viene de la pantalla anterior (la de `Products` ) Este caso es muy muy típico y hay varias soluciones para afrontarlo.

- La primera solución podría ser que nuestra ruta `/details` se convierta en un recurso único para cada producto del estilo `/details/{id}` . Como estamos usando productos de ejemplo que no tienen `id` podríamos emplear el `name` como identificador.
  - Nuestra ruta sería pues algo así `/details/tortillas-de-trigo-wraps-6-unidades`
  - Ahora lo que nos faltaría sería obtener sus datos. Al igual que hacemos en los `Products` con un `useEffect` podemos emplear ese `id` para solicitar a nuestro backend los datos del producto y así poder mostrarlos.
- Una segunda solución sería comprobar si el producto que estamos a recibir en la pantalla de detalles está a `undefined` o contiene datos. En caso de ser `undefined` lo que podemos hacer es mover al usuario a `/` .

Intentemos hacer esta segunda opción. La solución pasaría por usar un `useEffect` (que se ejecuta una única vez) y que llame al `navigate` para ir a la pantalla principal. Pruébalo!

Pero... va seguir fallando porque el `useEffect` se ejecuta después del render y el primer render falla porque `product` es `undefined` . Pero... al igual que hicimos en la pantalla de `Products` para mostrar el `Loader` podemos hacer algo similar aquí. Podemos incluir una condición que compruebe el valor de `products` y, en caso de que no sea un valor válido, pues pintamos otra cosa (puede ser un `Loader` o lo que se quiera).

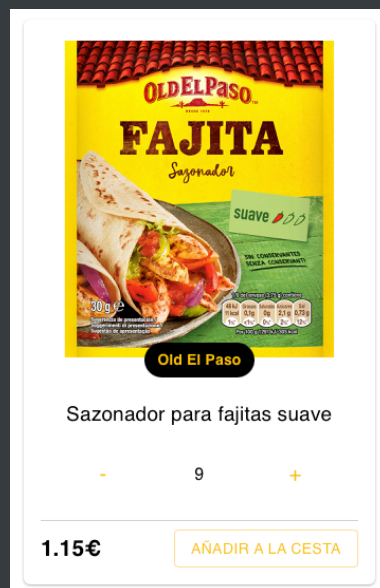
## Ejercicio 4 – Añadiendo productos a un carrito.

### Paso 1

El objetivo de este ejercicio será incluir la funcionalidad de añadir productos a al carrito. Vamos empezar incluyendo el counter en cada unos de nuestros `ProductItem` que forma la lista de productos.

Como estamos incluyendo un contador con dos botones dentro de nuestra card, sería interesante hacer que solo podamos ir a la pantalla de detalles si clicamos en la imagen para evitar conflictos con las otras acciones.

El item debería verse ahora tal que así.



### Paso 2

Necesitamos tener un sitio donde poder guardar los elementos que vayamos añadiendo a la cesta, por lo que necesitaremos un estado.

Intenta hacer que nuestro componente `App` imprima por consola el valor de este nuevo estado que tenemos que crear. Al principio estaría vacío => `[]` pero podemos utilizar el botón de `añadir a la cesta` para llenar esa lista (Por simplicidad, en este paso asumiremos que solo añadimos 1 item, por lo que podemos ignorar el valor del `Counter` de momento).

Idealmente, cada vez que añadamos un item a la cesta, el componente `App` debe imprimir por pantalla el estado actualizado.



Vas a tener que usar el hook `useState` para crear un estado nuevo y su función modificadora te permitirá actualizar su valor y así poder añadir a la lista.

Por consola, cada vez que clickemos en el botón debería aparecernos lo siguiente:

```
my carrito > [{-}] App.tsx:17
my carrito > (2) [{-}, {-}] App.tsx:17
my carrito > (3) [{-}, {-}, {-}] App.tsx:17
my carrito > (4) [{-}, {-}, {-}, {-}] App.tsx:17
my carrito > (5) [{-}, {-}, {-}, {-}, {-}] App.tsx:17
my carrito > (6) [{-}, {-}, {-}, {-}, {-}, {-}] App.tsx:17
>
```

### Paso 3

Evitar que si añadimos al carrito el mismo producto varias veces se introduzca un nuevo elemento en la lista. La mejor opción pasaría por:

- Guardar un elemento nuevo junto con los datos del producto que nos diga cuantos hemos añadido, ejemplo:

```
{ brand: 'xxxx', name: 'yyyy', units: 1 }
```

- Cada vez que añadamos un elemento mirar si se encuentra ya en el carrito y
  - Si ya está -> incrementar el valor de `units` en 1.
  - Si no está -> añadirlo e incluir el valor del campo `units` a 1.



Los Arrays JavaScript tienen un método `find` que puedes utilizar para resolver esto -  
> [Array.find](#)

Así es como debería verse por consola nuestro carrito.

```
my carrito                               App.tsx:29
▼ (2) [{...}, {...}] ⓘ
  ▼ 0:
    amount: "170g"
    brand: "Ta-Tung"
    conservation: "Conservar entre 0 y 4°C. Una vez abierto, consumir en 24 hora..."
    howToUse: "¡Recomendado! Al vapor: Coloca el producto en una vaporera e intr..."
    image: "https://sgfm.elcorteingles.es/SGFM/dctm/MEDIA03/201803/01/0011881270..."
    ingredients: "GYOZAS (154g): Masa (45%): harina de trigo, agua, gluten de tr..."
    name: "Dim Sum gyoza de carne"
    unitPrice: 2.85
    units: 11
    ► [[Prototype]]: Object
  ▼ 1:
    amount: "200ml"
    brand: "Tiger Khan"
    conservation: "Conservar en un lugar seco. Una vez abierto mantener en refri..."
    image: "https://sgfm.elcorteingles.es/SGFM/dctm/MEDIA03/201609/21/0011808930..."
    ingredients: "Agua, vinagre de arroz"
    name: "Vinagre de arroz"
    unitPrice: 2.49
    units: 8
    ► [[Prototype]]: Object
  length: 2
  ► [[Prototype]]: Array(0)
```

## Paso 4

En vez de hacer el incremento de 1 en 1, haz que el número de productos que queremos añadir sea el valor de nuestro `Counter`.

Modifica el `Counter` para que el valor mínimo que queremos que acepte sea 1 en vez de 0.

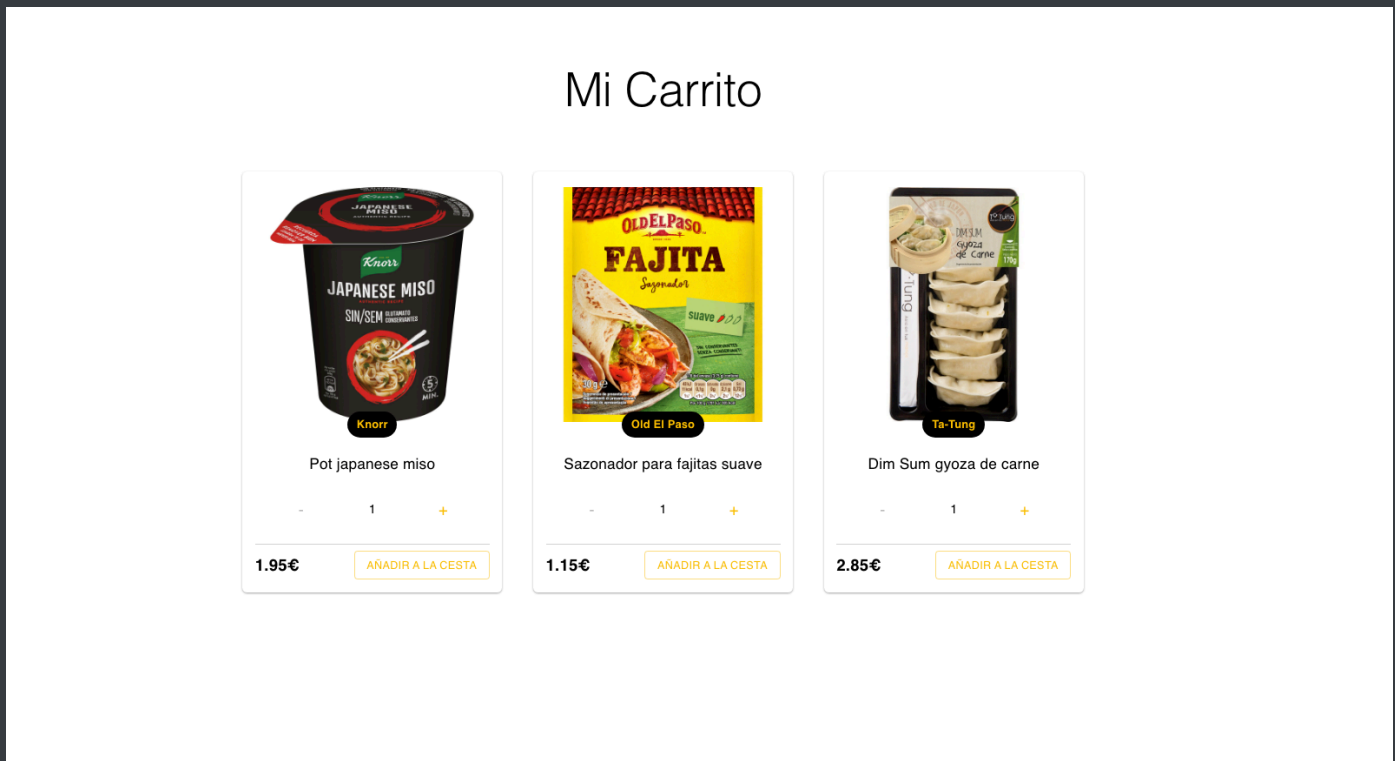


# Ejercicio 5 – Ver mi carrito

## Paso 1

Una vez que tengamos la lógica lista para añadir elementos al carrito, vamos crear una página que nos permite ver nuestro carrito actual.

Añade un botón en la lista de productos que nos lleve a esa nueva página (usaremos la ruta /cart ). Utilizar el componente `ProductItem` que ya tenemos creado para mostrar los ítems que hay en el carrito. El resultado de esa página sería algo como la siguiente:



## Paso 2

Vamos añadir la prop `isItemAddedToCart` al componente `ProductItem` para diferenciar los ítems que están añadidos al carrito de los que no. Con esta prop, podremos dar un estilo diferente a elementos que mostramos en la página del carrito haciendo que se vea algo así (el botón de eliminar lo implementamos luego).



Esta prop nos sirve para modificar `ProductItem` de manera condicional, de tal forma que los elementos que vemos en `Products` queden tan cual estaban

## Mi Carrito

 Tiger Khan	 Old El Paso	 Ta-Tung	 Alfetz	 Knorr
Vinagre de arroz	Sazonador para fajitas suave	Dim Sum gyoza de carne	Falafel estilo libanés	Pot japanese miso
1 ítem en el carrito	1 ítem en el carrito	1 ítem en el carrito	1 ítem en el carrito	6 ítems en el carrito
2.49€ ELIMINAR DE LA CESTA	1.15€ ELIMINAR DE LA CESTA	2.85€ ELIMINAR DE LA CESTA	3.05€ ELIMINAR DE LA CESTA	1.95€ ELIMINAR DE LA CESTA

## Paso 3

Utilizar esa misma prop para indicar en la lista de `Products` cuales son los que ya están añadidos al carrito y cuales no. De tal forma que se vea así. (Se pueda usar la función `find` de los arrays para saber si un `item` está añadido al carrito o no)

## Página de productos

MI CARRITO



Old El Paso

Tortillas de trigo Wraps 6 unidades

- 1 +

2.69€

AÑADIR A LA CESTA



Old El Paso

Sazonador para fajitas suave

- 1 +

1.15€

AÑADIR A LA CESTA



Knorr

Pot japonese miso

4 ítems en el carrito

1.95€

ELIMINAR DE LA CESTA



Tiger Khan

Vinagre de arroz

1 ítem en el carrito

2.49€

ELIMINAR DE LA CESTA



Ta-Tung

Dim Sum gyoza de carne

- 1 +

2.85€

AÑADIR A LA CESTA

### Paso 4

Implementar la lógica de eliminar de la cesta.