

Resumen funcional del simulador de compra

Autor: **Prellezo Sergio Emanuel**

Sitio: <https://mammoth-compras.netlify.app/>

Repositorio: <https://github.com/SerePrec/Mammoth-Compra>

1. Introducción

Preparé este resumen a modo de guía para enumerar las funcionalidades que se encuentran en el simulador. Lo que puede ser útil para recordar y poder reutilizar algunas de ellas en proyectos futuros.

Este proyecto nace como una extensión de mi anterior trabajo final en el curso de desarrollo web <https://mammoth-bicicleteria.netlify.app/> , aunque para esta entrega se trata la sección de compras de manera aislada para no mezclar los objetivos. Por eso varios de los links a distintas páginas de mi anterior proyecto se encuentran desvinculadas.

Se dividió el código JS en distintos archivos para organizarlo mejor:

- `clase.js`: contiene la definición de todas las clases.
- `data.js`: contiene las funciones AJAX que solicitan los datos con los que voy a trabajar en el simulador.
- `productos.js`: contiene los scripts de la página `productos.html`.
- `compra.js`: contiene los scripts de la página `compra.html`.
- `base.js`: contiene scripts básicos comunes a todas las páginas (no relacionados a la funcionalidad del simulador sino a elementos de diseño, como por ejemplo el botón fijo de scroll).

También se ordenó e introdujeron diversos comentarios a lo largo del código para hacer más fácil la comprensión de su funcionamiento.

El sitio tiene responsive pasando por distintos break-points con el objeto de cubrir la visualización en pequeños celulares, tabletas, notebooks y computadoras de escritorio.

La codificación de los estilos la hice mediante SCSS, utilizando la división en parciales.

Trabajé con Git para versionar mi proyecto y llevar un control sobre los cambios que fue sufriendo. Generé una rama llamada *desarrollo* sobre la cual iba trabajando y a medida que veía que los cambios me parecían correctos, la fusionaba con la master (principal).

Todo el repositorio fue subido a Github, incluyendo las ramas y sub-ramas.

Finalmente, el proyecto fue subido al servidor Netlify para visualizar la funcionalidad del mismo, ya que, si se prueba localmente, se debe utilizar algún plugin como el Live server de VSC para que la petición AJAX al JSON estático se lleve a cabo correctamente.

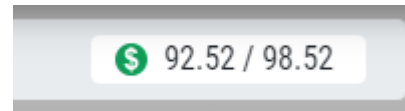
Links a terceros:

- Bootstrap
- Wow animate
- Fontawesome
- Fuentes de Google

2. Obtención de datos del simulador

El simulador para su funcionamiento necesita la información de dos peticiones AJAX. Una de ellas se encarga de traer el array de productos (que luego se instancian con la clase Productos para poder disponer de métodos) desde un JSON estático ubicado en la carpeta /data.

La otra petición, se encarga de obtener el valor oficial del dólar actual desde la API dolarsi.com, ya que la lista de precios de los productos se encuentra en dólares por ser material importado y luego se deben convertir a pesos Argentinos. Una vez obtenida se exhibe con una animación.



Mediante el método .when() de jQuery, se verifica al cargar la página que ambas peticiones resulten exitosas.

De ser así se procede a asignar la mayoría de los eventos relacionados al simulador (ya que de lo contrario no tendría sentido y podría ocasionar resultados no deseados) e inicializarlo con su función principal iniciar().

De lo contrario se muestra un mensaje con animación, de "error de carga"



Nota: ha ocurrido durante el desarrollo, cuando actualizaba muy seguido la página para ver los cambios, que la API de dolarsi.com no enviaba respuesta (seguramente estaban realizando actualizaciones), por lo que la página daba el mensaje de "error de carga"

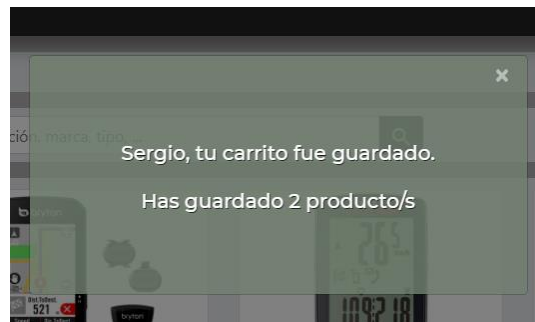
3. Login / Logout del usuario y storage

Decidí que la compra se puede realizar estando o no logueado, porque al hacer el checkout se solicitan todos los datos necesarios.

Si el usuario no se loguea el carrito actual se almacena en el sessionStorage, permitiendo mantener la selección mientras no se cierre la pestaña del navegador.

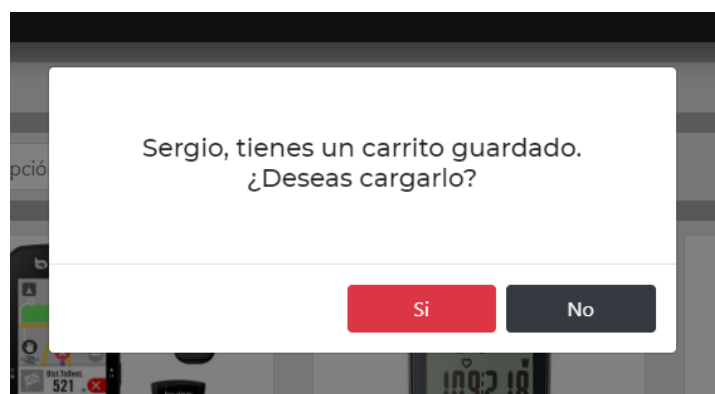
Por el contrario, si el usuario se loguea tiene dos ventajas:

- Permite mantener el carrito actual a pesar de cerrar el navegador ya que el mismo se guarda en el localStorage siempre que permanezca logueado.
- Al desloguearse, si el usuario tiene productos seleccionados, permite guardar distintos carritos para distintos usuarios que se hayan logueado desde el dispositivo, dando la opción de recuperarlo la próxima vez que ese usuario se loguea.



Si se opta por recuperarlo o no, ese carrito ya se borra del localStorage, permitiendo volver a guardar otro al desloguearse nuevamente.

Si el usuario comienza una selección y luego decide loguearse, si tiene un carrito guardado se pregunta que quiere hacer. Si lo carga, sigue con la selección de aquel carrito guardado, de lo contrario, se descarta y continúa con la selección actual.



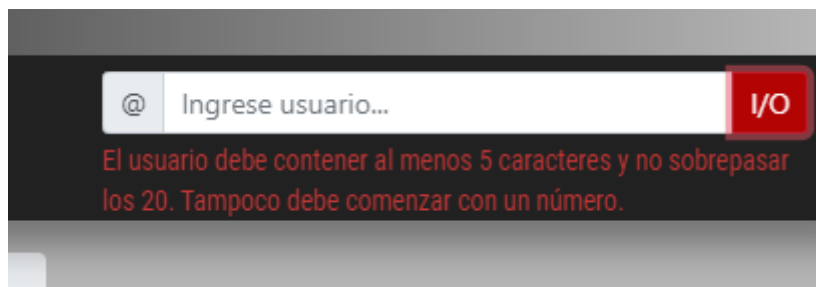
También incluí una verificación importante a la hora de cargar el carrito que tiene que ver con la disponibilidad actual del producto, lo que se puede

simular localmente, agregando productos y luego poniendo su disponibilidad por debajo del valor agregado al carrito.

Si todos los productos están disponibles lo alerta con un mensaje verde, si algún producto no se encuentra disponible en la cantidad solicitada, lo advierte con un mensaje rojo y quita ese ítem por completo del carrito.



De manera arbitraria, y para imponer ciertas condiciones, el usuario no debe comenzar con un número y no puede tener menos de 5 caracteres ni más de 20, lo cual se advierte con una animación por debajo del input, si no se cumple dicha condición.



4. Consola

A lo largo de todo el proceso se van registrando mensajes importantes en la consola asociados a los “movimientos” de la compra. Estos simularían ser mensajes de control para la empresa.

```
products.js:127
  CarritoUsuario {usuario: "", miSeleccion: Array
(0)}
Comprando Mochila Camelback Mini products.js:680
Mule 1.5L // Cant: 1
Comprando Pulsómetro De Bicicleta products.js:680
Bryton Rider 750 T Cadencia, FC Y Veloc. // Cant: 3
Comprando Bicicleta MTB products.js:680
Cannondale Trail 7 2021 // Cant: 1
----- Carrito vaciado ----- products.js:761
Comprando Remera Castelli products.js:680
Competizione Ineos Grenadiers // Cant: 1
Comprando Campera Chaleco products.js:680
Castelli Aria Vest // Cant: 3
Comprando Remera Cube TeamLine products.js:680
L/S // Cant: 1
Comprando Guantes Endura Hummvee products.js:680
Plus 2 // Cant: 1
Restando Campera Chaleco Castelli products.js:698
Aria Vest // Cant: 1
Eliminando Guantes Endura Hummvee products.js:722
Plus 2 // Cant: 1
>
```

También al finalizar la misma, se realiza una verificación del stock de todos los productos, y se advierte aquellos en que su disponibilidad se encuentra por debajo de su punto de repedido

```
compra.js:101
  {usuario: "", miSeleccion: Array(3)}
----- PAGO ACEPTADO ----- compra.js:222
Mi Carrito: compra.js:236
{usuario: "", miSeleccion: Array(3)}
Es necesario reponer el stock de: compra.js:387
- Velocimetro Cateye Inalámbrico ST-12.
Se encuentra 4 unidades por debajo del punto de
repedido
Es necesario reponer el stock de: compra.js:387
- Bicicleta Urbana Ryme Dubai 28'' 2021
Se encuentra 1 unidades por debajo del punto de
repedido
Es necesario reponer el stock de: compra.js:387
- Freno Sram Code RSC A1 Delantero
Se encuentra 2 unidades por debajo del punto de
repedido
Es necesario reponer el stock de: compra.js:387
- Cubierta Specialized Ground Control Grid Tubeless
Ready 29x2.3
Se encuentra 4 unidades por debajo del punto de
repedido
Es necesario reponer el stock de: compra.js:387
- Cadena Shimano Deore CN-M6100 12V
Se encuentra 3 unidades por debajo del punto de
repedido
Es necesario reponer el stock de: compra.js:387
- Casco Bell Avenue
Se encuentra 2 unidades por debajo del punto de
repedido
Es necesario reponer el stock de: compra.js:387
- Remera Kimoa Lab 01
Se encuentra 2 unidades por debajo del punto de
repedido
```

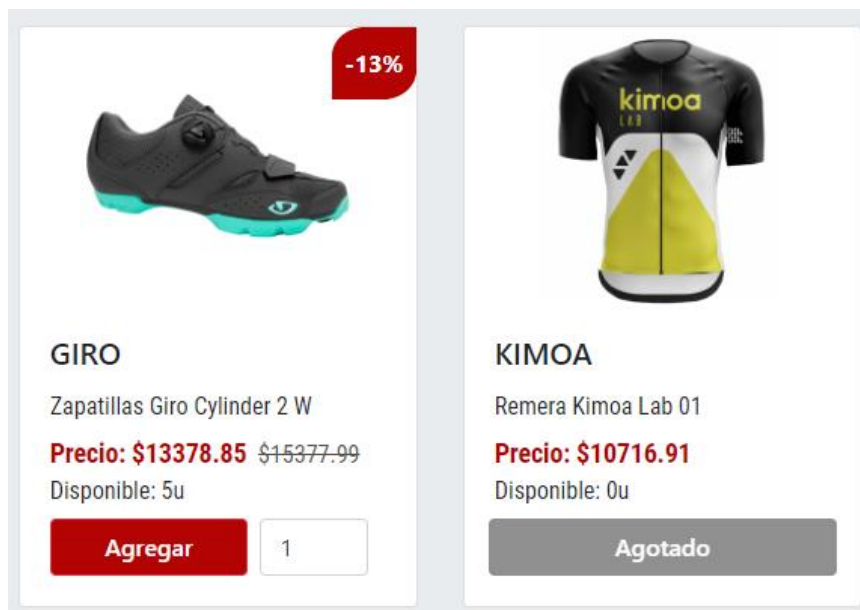
5. Tarjetas de productos

Se generan dinámicamente en función del archivo productos.JSON obtenido inicialmente.

Si posee descuento genera el html necesario para el “globo” del porcentaje de descuento y tacha el precio original exhibiendo el nuevo.

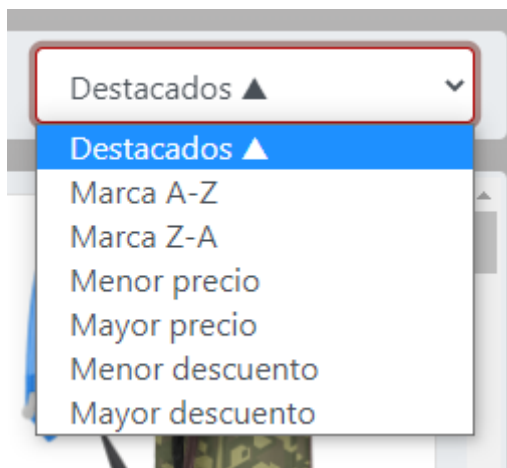
Si el producto no tiene disponibilidad, cambia el botón de agregar y el input de cantidad por un botón con otro formato y disabled.

A su vez, el rango de valores máximos de cada input asociado a cada producto se va estableciendo de acuerdo al valor de disponibilidad, si bien luego se valida la cantidad por otro lado también



6. Orden de productos

Existen 7 opciones de ordenamiento de los productos. Por defecto la página inicia con la opción “Destacados primero” donde ubica primero a los productos destacados ordenados por marca y descripción, luego los no destacados con el mismo orden.



Las otras opciones son orden alfabético A-Z o Z-A por marca y luego descripción, precio ascendente o descendente y por menor o mayor descuento.

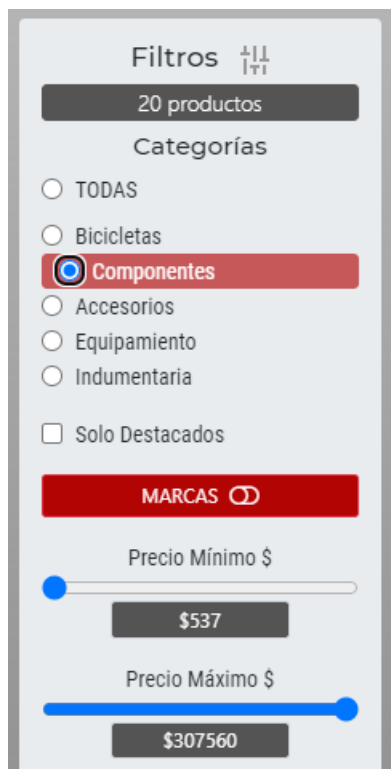
Este ordenamiento se puede hacer en cualquier proceso del filtrado de productos

7. Filtrado de productos

Existen 5 formas de filtrar los productos:

- Por búsqueda de palabra
- Por categoría
- Por destacados
- Por marcas (correspondiente a la **selección Madre**)
- Por rango de precios (correspondiente a la **selección Madre**)

La **selección Madre** la compone la combinación de los tres primeros filtros, es decir: búsqueda, categoría y destacados.



Es en base a esta **selección Madre** que se genera un array de productosFiltradosCliente.

Este array es el punto de partida “fijo” para los otros dos filtros: marcas y rango precios. Es decir, estos dos filtros trabajan siempre sobre esa **selección Madre**, pero no la modifican, solo agregan un filtrado extra a partir de ella.

Por eso los extremos del rango de precio se mantienen inalterados siempre que no se toque ninguno de los tres primeros filtros.

Lo mismo con el listado de marcas que se mantiene junto a las cantidades de productos de cada marca.

Es decir, ambas cosas se mantienen porque la **selección Madre** es la misma, pero si se toca cualquiera de los 3 filtros que la componen, se resetean los valores de los filtros de marcas y rango de precios, tomando los valores

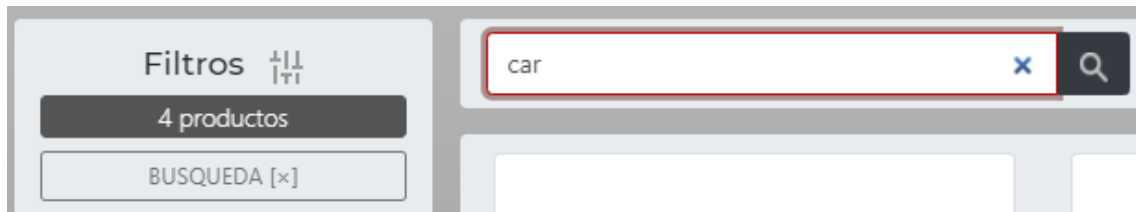
correspondientes a la nueva selección **Madre** escogida por el usuario.

Un ejemplo que muestra el funcionamiento de lo anterior es el siguiente:

Supongamos elegir una determinada palabra, dentro de una determinada categoría y luego si quiero ver solo los destacados o no. Bien, con estos tres filtros ya definimos nuestra **selección Madre**. Ahora, si vemos nuestro listado de marcas o nuestra escala de precios, comprobamos que se corresponden a esta selección y van a permanecer inalteradas siempre y cuando no cambiemos la misma al cambiar cualquiera de los 3 filtros que la componen. Por eso por ejemplo si tenía 5 marcas con sus respectivas cantidades de productos y realizo un cambio en el rango de precios, este listado de marcas permanece igual, aunque dentro de ese precio no existan determinadas marcas, porque recordemos que se corresponde a la **selección Madre** que permanece inalterada. Obviamente, si selecciono una marca que no tiene precios en ese rango, va a mostrar el mensaje de que no se encontraron productos con nuestros criterios de búsqueda.

Conclusión, los productos que finalmente se exhiben luego de interactuar con estos filtros son el resultado de la combinación de todos ellos (los 5).

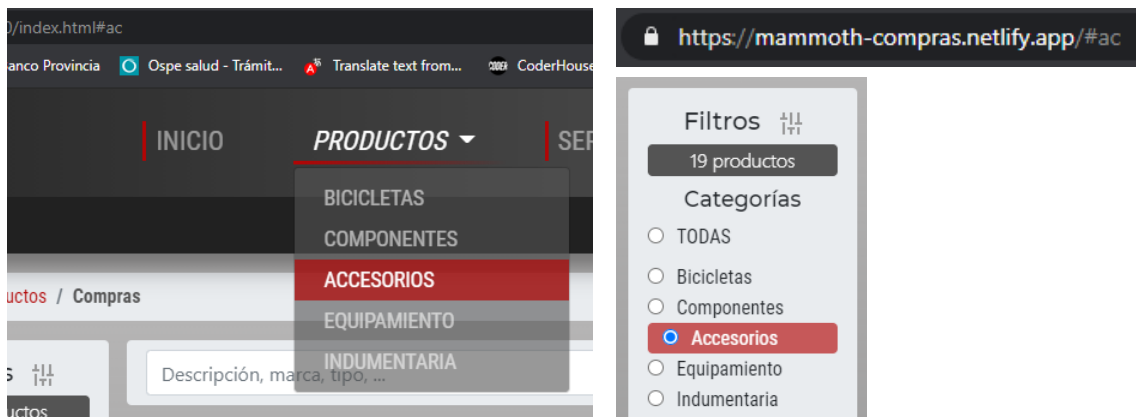
Búsqueda por palabra: se activa a medida que el usuario va introduciendo caracteres en el input correspondiente, y también es posible activarla al presionar el botón de la lupa (por si algún navegador no fuese compatible con el método anterior).



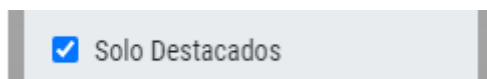
Al activar esta búsqueda, aparece un botón en la barra de filtros dando a entender que este método se encuentra activo. Si se borran por completo los caracteres o se toca el botón de "Búsqueda [x]", se limpia este filtro.

Filtro por categoría: Permite circunscribir los productos entre las categorías existentes o todas.

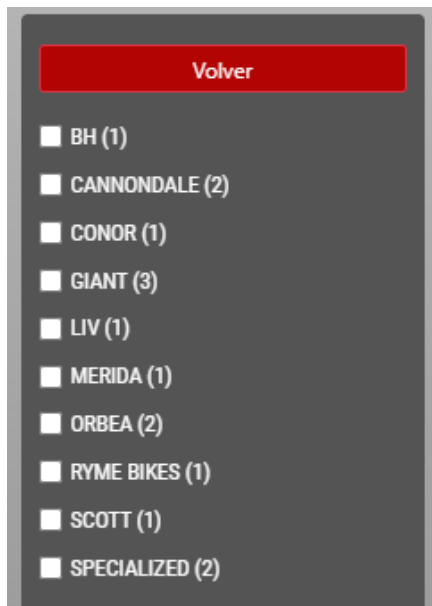
Este también tiene interacción con la carga de la página o cambio del Hash #, es decir, que si se accede a la página a través de una url con el link al # correspondiente, se pasa a esa selección. Esto puede darse al seleccionar en el navbar a qué parte de los productos quiero "ir"



Filtro por destacados: Solo muestra los productos que son destacados al activar el checkbox.



Filtrado por marcas: Muestra un listado con las marcas que contiene la **selección Madre** como también la cantidad de productos que las componen.



Se pueden seleccionar una o varias marcas y el filtro mostrará los productos de la selección madre que contengan dichas marcas y se encuentren dentro del rango de precios elegido.

Se puede volver a ingresar y cambiar dicha selección a gusto.

Importante: Esta selección se resetea automáticamente al cambiar cualquiera de los 3 primeros filtros enumerados.

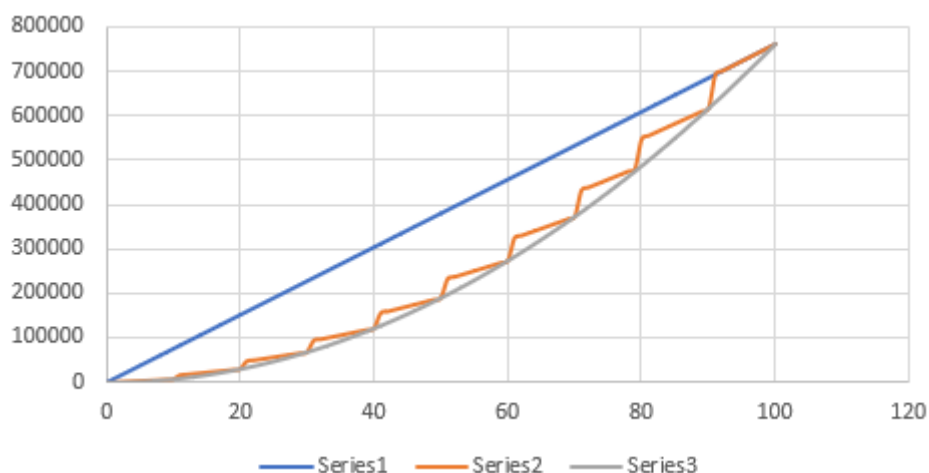
Para saber si tenemos activa alguna selección del filtro marcas el logo a la derecha del botón pasa del estado OFF al ON si hay marcas seleccionadas.



Filtro por rango de precio: Permite elegir el valor del producto desde un precio mínimo a uno máximo.

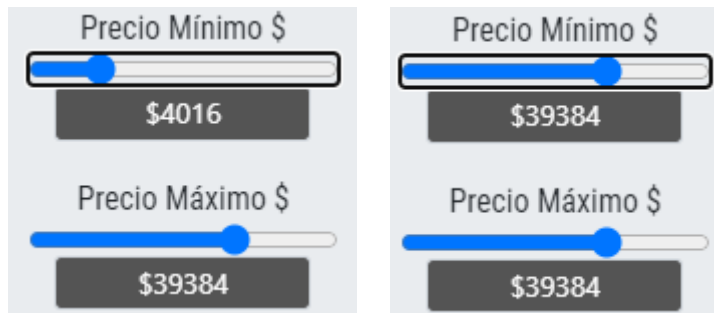
Respecto a la escala quiero decir dos cosas:

- Los valores mín y máx se setean de acuerdo a la **selección Madre**. Se recorre el array de productos que la componen y se sacan los valores extremos de los precios
- Como puede haber mucha diferencia entre los extremos de precio, generé una escala exponencial, en donde crece más suavemente al comienzo para permitir más fácilmente precios del rango inferior y luego crece más rápido a medida que se acerca al extremo máximo.



En azul se ve una escala lineal y la que se adoptó en la curva, color gris.

Realicé mediante funciones asociadas a los eventos del control del rango de precios la verificación para que no permita la inconsistencia de que el precio mín se elija por encima del máximo. En caso de suceder esto, inmediatamente al salir de la selección, los indicadores se acomodan automáticamente al valor máximo permitido para el cruce (min = max).



Ejemplo al querer pasar con el precio mínimo el valor del precio máximo.

Solo se ajusta al valor máximo válido para esta situación.

También generé eventos asociados al mouse y al teclado para estos input, así se pueden manejar también mediante Tab y las flechas del teclado, logrando un ajuste fino.

Por último, estos filtros en versión mobile, se ven contraídos y si se apaisa el dispositivo dando origen a la columna izquierda de filtros, vuelven a ser visibles.

En el caso del celular vertical, inicialmente aparece contraído y tocando sobre la leyenda de filtros, se despliega completamente haciéndose visible. Lo mismo con el botón de marcas.



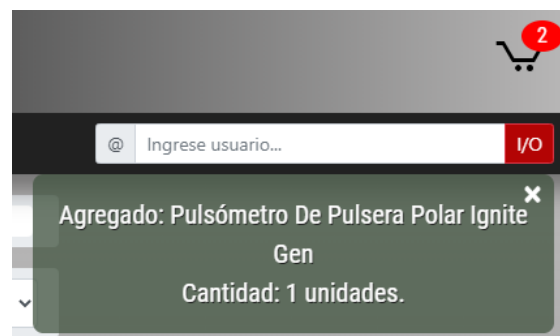
8. Agregando o quitando elementos del carrito

Agregando productos:

Para incorporar productos al carrito se puede simplemente apretar el botón agregar, la cantidad de veces correspondiente a la cantidad de unidades a comprar o seleccionar las unidades desde el input derecho y luego apretar agregar.

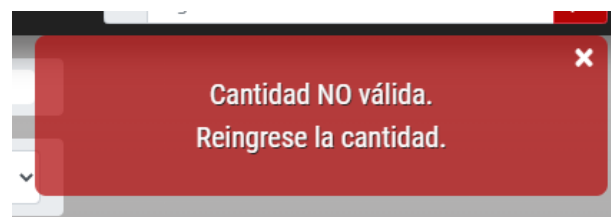
En cualquiera de los casos, las opciones repetidas se incrementan en cantidad en el carrito, sin repetir el mismo producto.

Al agregar el producto exitosamente, aparece un mensaje confirmando la operación y el icono animado del carrito se actualiza.

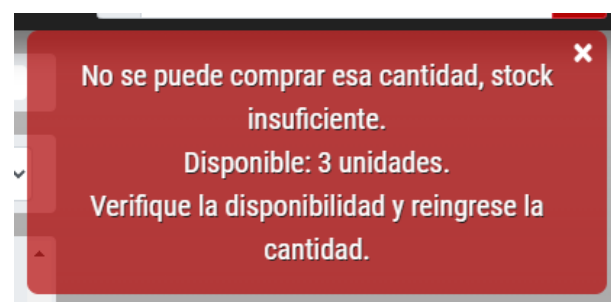


Si bien constantemente con cada operación se van actualizando los valores y atributos de las tarjetas y con las flechitas del input number no es posible seleccionar una cantidad no acorde, si es posible hacerlo al ingresar el valor a mano, por eso las siguientes validaciones.

Si se intenta ingresar un valor inválido de cantidad (ej. Número negativo), se advierte de ello.

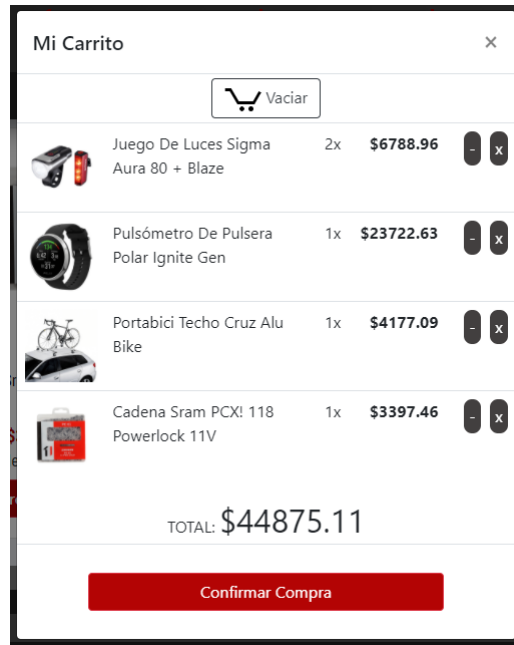


Lo mismo si se intenta ingresar a mano una cantidad mayor a la que se encuentra disponible.

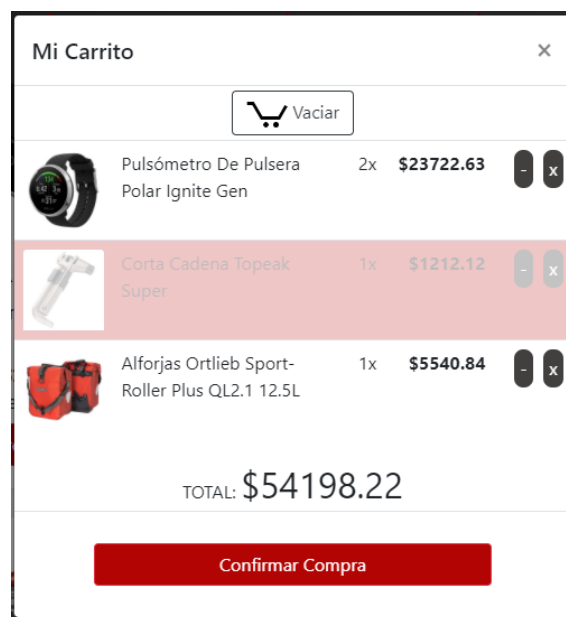


Quitando productos:

Hay 3 formas de quitar elementos del carrito: restando de a una unidad, eliminando el elemento completo o vaciando el carrito. Esto se hace a través de los correspondientes botones.



En cualquiera de estos casos, como sucede al agregar, se actualiza la información de las tarjetas, del contenido del carrito y del icono del carrito, visualizándose los cambios inmediatamente. Al desaparecer uno o más elementos, se produce una animación en donde se desvanecen rápidamente tornándose de color rojo y luego que se completa, se regenera el carrito junto con el importe total.



Como se mencionó anteriormente, con cada interacción del carrito ya sea agregando o quitando elementos, se regenera la información asociada.

Respecto a las tarjetas en lugar de usar la función `mostrarProductos`, generé una nueva que es `actualizarInfoTarjetas`, porque al tener imágenes y ser muchas, podía ocasionar parpadeos o demoras en la reconstrucción de toda la información, pero como en este caso todo sigue en el mismo lugar (no como al ordenar las tarjetas), era más óptimo reescribir solo lo que se modificaba. Es más, en algunos casos, con el efecto de regenerar todo el contenido aparte del parpadeo se perdía el scroll actual dentro del contenedor.

Entonces mediante comprobaciones respecto a la disponibilidad, se regeneran los atributos de máximo de los inputs cantidad, la disponibilidad y el diseño del botón. Pasando de uno a otro diseño de acuerdo a si el producto tiene o no stock disponible. De esta manera logro ahorrar recursos respecto a lo que sería el renderizado total de este contenedor.

9. Pasando al resumen de compra – checkout

Una vez que el usuario confirma la compra desde el botón que se encuentra dentro del modal del carrito, se le redirige a otra página: `compra.html`

Para eso, se hace uso del `sessionStorage` para pasarle a esta página toda la información necesaria para poder continuar en base a la selección del cliente.

Generé una validación en la que si no dispone de los datos necesarios o no existe una selección del cliente (lo que puede suceder al pretender acceder directamente a esta url, sin antes pasar por el sector de compras y generar algún carrito), la página redirige automáticamente a la original (en el caso de esta entrega, `index.html`).

En el caso de que lo anterior se complete con éxito, se exhiben con una animación todos los productos del carrito del usuario en una columna a la izquierda de la pantalla.

A la derecha aparece un formulario que se debe llenar con todos los datos relevantes a la compra y la posibilidad de elegir un plan de pago en cuotas. Este último también aparece con una animación luego que culmina la de la presentación de los productos elegidos.

El usuario posee dos opciones: llenar el formulario y **aceptar la compra** o **volver hacia el showroom** para cancelar o modificar su selección.

[Inicio](#) / [Productos](#) / [Compras](#) / [Checkout](#)

Resumen de compra

	Mochila Camelback Mini Mule 1.5L	1x	\$5483.62
	Pulsómetro De Bicicleta Bryton Rider 750 T Cadencia, FC Y Veloc.	1x	\$40411.92
	Calza Cube WLS TeamLine W	1x	\$5900.86
	Remera Cube TeamLine L/S	2x	\$5039.33
	Pulsómetro De Pulsera Garmin Fenix 6 Solar Black Strap	1x	\$83327.23
Total de compra			\$145202.29

Nombre

Apellido

Email

Dirección

Datos de su tarjeta

Número

Vto

Cuotas

- ☒ 1 Cuota de \$145202.29
- ☐ 3 Cuotas de \$52272.82 (Int.: 8%) Total: \$156818.47
- ☐ 6 Cuotas de \$27104.43 (Int.: 12%) Total: \$162626.56
- ☐ 12 Cuotas de \$14520.23 (Int.: 20%) Total: \$174242.75
- ☐ 18 Cuotas de \$10486.83 (Int.: 30%) Total: \$188762.98

[Volver a ShowRoom](#)
[Aceptar la compra](#)

En relación al formulario todos sus campos poseen una validación por HTML 5. Hubiese sido mejor hacerla por JS, pero me enfoqué en generar código bien orientado al funcionamiento del simulador propiamente dicho y me valgo de esta herramienta ya predefinida para ahorrar tiempo en función de mi objetivo principal.

Todos los campos del formulario poseen el atributo "required" por lo que no permite su "envío" hasta que no se completen y siguiendo el patrón correspondiente al tipo de input. En varios input (número de tarjeta, fecha de vencimiento, etc) introduje un patrón propio que se debe respetar para hacer más real el valor requerido para que se considere válido.

10. Aceptando la compra y detalle final

Una vez que el usuario completa el formulario debidamente, escoge el plan de cuotas y aprieta el botón para aceptar la compra, se captura el evento "submit" del mismo y se pasan a ejecutar una serie de acciones previamente eliminando el comportamiento por default del submit con el método `e.preventDefault()`.

En esta etapa, se genera el envío de una petición AJAX del tipo POST al sitio <https://jsonplaceholder.typicode.com/posts>, con el objeto de simular el envío de información a un servidor para procesar el pago. Para esto, paso anterior, se captura toda la información introducida por el usuario en el formulario.

Se ejecuta una animación mostrando que los datos son enviados, y una vez que recibimos respuesta de este sitio, la animación pasa a "validando pago" por un tiempo de 3s como simulando el proceso de validación por una entidad de cobro.



Finalmente, transcurrido ese tiempo, se muestra el detalle final de la compra en una tabla junto con un agradecimiento por la misma y una leyenda citando al nombre del cliente y su e-mail donde se dice que luego recibirá las instrucciones para el envío.

MUCHAS GRACIAS POR SU COMPRA!

Sergio Emanuel, en unos minutos recibirás en tu casilla de e-mail prelezose@yahoo.com.ar los detalles para coordinar la entrega

Detalle de compra

#	PRODUCTO	CANTIDAD	SUBTOTAL
1	 Pulsómetro De Bicicleta Bryton Rider 750 T Cadencia, FC Y Veloc.	1	\$40411.92
2	 Mochila Camelback Mini Mule 1.5L	1	\$5483.62
3	 Remera Castelli Competizione Ineos Grenadiers	2	\$19061.55
TOTAL			\$64957.09

Forma De Pago

6 Cuota/s de: \$12125.32

Total a pagar (*): \$72751.94

(*) Incluye impuestos y financiación

Luego de esto se vacía el carrito del usuario, actualizándose el icono del mismo y se muestran por consola los mensajes que serían orientados a la empresa indicando que productos sería bueno ir pidiendo al proveedor ya que se encuentra por debajo de su punto de repedido.

Si una recorre el array de productos en este punto, puede verificarse como las cantidades son las originales menos las de la compra llevada a cabo