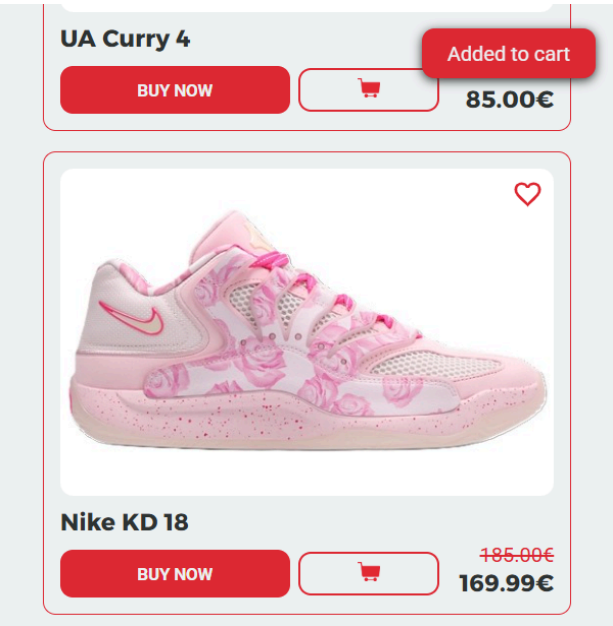
DOCUMENTACIÓN PARTE 4 - SHIFT & GO

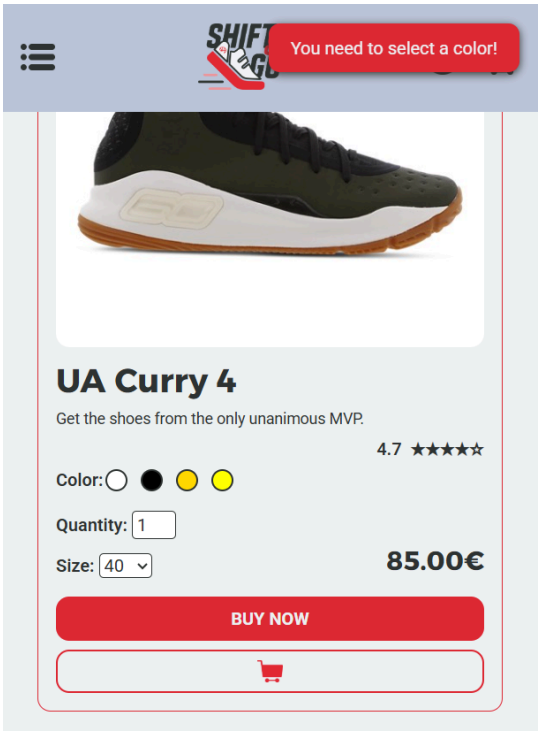
USO DEL CARRITO

Para poder utilizar el carrito desde el Frontend, se ha optado por el uso de **cookies** para comunicarse con la base de datos, generando un customer_id único, pudiendo introducir así datos en la tabla que contiene la información del carrito.

```php
setcookie(
    "customer_id",
    $customer_id,
    time() + (60 * 60 * 24 * 30), //30 días
    "/",
    "",
    $secure,
    true
);
```

Evidentemente, se ha añadido funcionalidad a los botones de **añadir al carrito** tanto en la página principal como en el detalle del producto. Mostrando un mensaje en forma de popup para dar algo de feedback al usuario.





Además, el carrito funciona de manera completamente dinámica con la base de datos, actualizándose cada vez que modificamos algo. De esta manera, el **subtotal se actualiza** cada vez que modificamos la cantidad, no solo visualmente en el lado del cliente, sino también en el servidor, en la base de datos. Lo mismo ocurre para el color y talla seleccionados.

Código para actualizar la base de datos:

backend > cart > update_quantity.php

```php
<?php
  header("Content-Type: application/json; charset=UTF-8");
  require($_SERVER['DOCUMENT_ROOT'].'/student012/shop/backend/config/db_connect.php');
  require($_SERVER['DOCUMENT_ROOT'].'/student012/shop/backend/functions/get_customer_id.php');

  $input = json_decode(file_get_contents('php://input'), true);

  if (!isset($input['product_id'], $input['selected_color'], $input['size'], $input['quantity'])) {
    http_response_code(400);
    echo json_encode(['error' => 'Datos incompletos']);
    exit;
  }

  $customer_id = getCustomerId($conn);

  $stmt = $conn->prepare("
    UPDATE `012_shopping_cart`
    SET quantity = ?
    WHERE customer_id = ?
      AND product_id = ?
      AND selected_color = ?
      AND size = ?
  ");

  $stmt->bind_param(
    "iiiss",
    $input['quantity'],
    $customer_id,
    $input['product_id'],
    $input['selected_color'],
    $input['size']
  );

  $stmt->execute();

  echo json_encode(['success' => true]);
?>
```

Código en JavaScript para manejar esos cambios:

```javascript
//Cambios dentro del carrito: actualizar subtotal, cantidades, colores, tallas, eliminar.
function recalculateSubtotal() {
  let subtotal = 0;
  document.querySelectorAll('.product-cart').forEach(item => {
    const price = parseFloat(item.dataset.price);
    const qty = parseInt(item.querySelector('.cart-qty').value);
    subtotal += price * qty;
  });
  document.querySelector(".subtotal span").textContent =
  subtotal.toFixed(2) + "€";
}

document.querySelectorAll(".cart-qty").forEach(select => {
  select.addEventListener("change", async e => {
    const card = e.target.closest(".product-cart");

    await fetch(`${API_URL}/cart/update_quantity.php`, {
      method: "POST",
      headers: {"Content-Type":"application/json"},
      body: JSON.stringify({
        product_id: card.dataset.id,
        selected_color: card.dataset.color,
        size: card.dataset.size,
        quantity: e.target.value
      })
    });

    recalculateSubtotal();
  });
});
```

```
document.querySelectorAll(".color-radio, .cart-size").forEach(input => {
  input.addEventListener("change", async e => {
    const card = e.target.closest(".product-cart");

    const newColor = card.querySelector(".color-radio:checked")?.value || '';
    const newSize = card.querySelector(".cart-size").value;

    await fetch(`${API_URL}/cart/update_variant.php`, {
      method: "POST",
      headers: {"Content-Type":"application/json"},
      body: JSON.stringify({
        product_id: card.dataset.id,
        old_color: card.dataset.color,
        old_size: card.dataset.size,
        new_color: newColor,
        new_size: newSize
      })
    });

    card.dataset.color = newColor;
    card.dataset.size = newSize;
  });
});
```

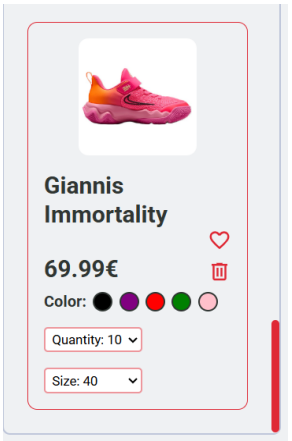
Cabe añadir que también tiene funcionalidad para eliminar productos del carrito:

```
//Eliminar producto mejorado -- La página no se refresca cada vez
document.querySelectorAll(".remove").forEach(btn => {
  btn.addEventListener("click", async e => {
    const card = e.target.closest(".product-cart");

    await fetch(`${API_URL}/cart/remove_from_cart.php`, {
      method: "POST",
      headers: {"Content-Type":"application/json"},
      body: JSON.stringify({
        product_id: card.dataset.id,
        selected_color: card.dataset.color,
        size: card.dataset.size
      })
    });

    card.remove();
    recalculateSubtotal();
  });
});
```

CAMBIOS EN EL DISEÑO

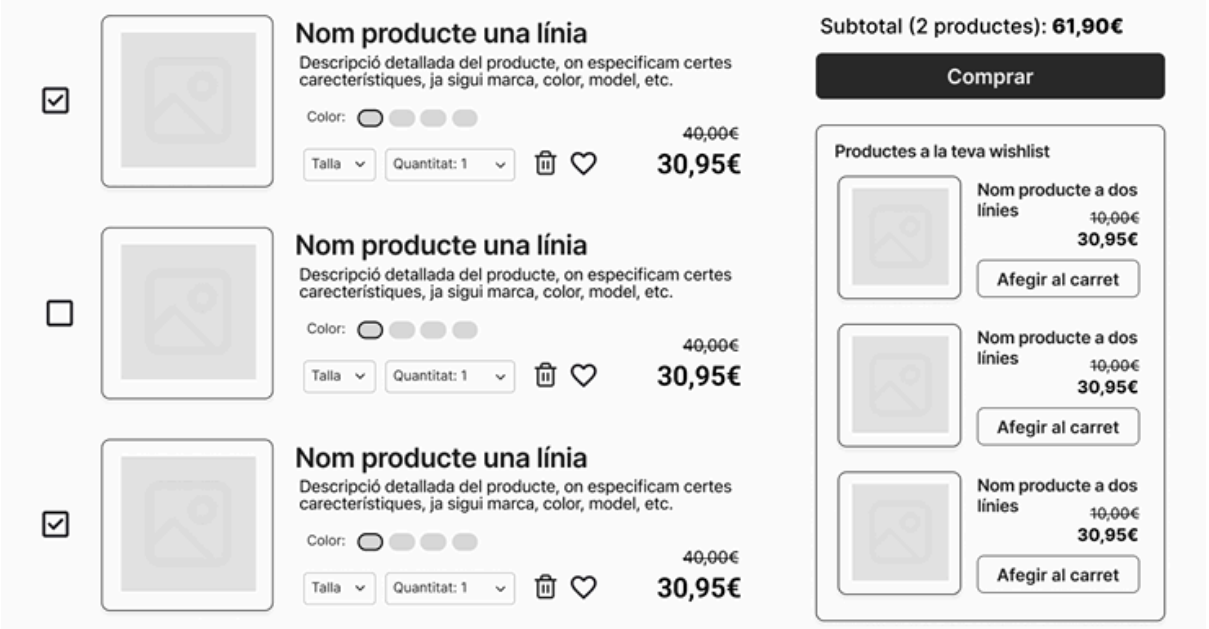
Se ha eliminado el checkbox para seleccionar qué productos incluir en el pedido. Por el funcionamiento de la base de datos, se entiende que al hacer click en “ORDER NOW”, se vuelca todo el contenido del carrito en el pedido:

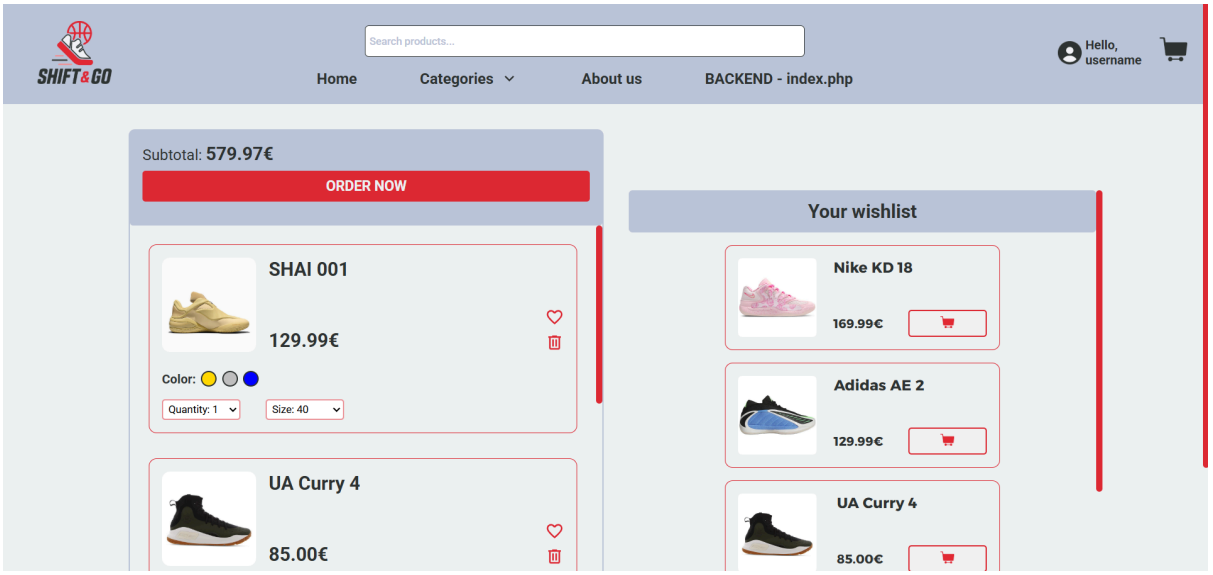
Para **Mobile**, en caso de tener un dispositivo con una pantalla muy pequeña, se ha reordenado la colocación de los elementos para que nada se desborde en la pantalla:



Para **Desktop**, se ha remodelado la distribución de los elementos, ya que carecía de sentido que el subtotal se mostrase encima de la lista de deseos, pudiendo inducir a confusión, además de complicar en exceso la recolocación de elementos entre los distintos breakpoints.

Por ello, se ha optado por extender el diseño utilizado en Tablet, dando algo más de espacio en los laterales para que el contenido respire más, aprovechando el mayor espacio del que se dispone.





CAMBIOS GENERALES

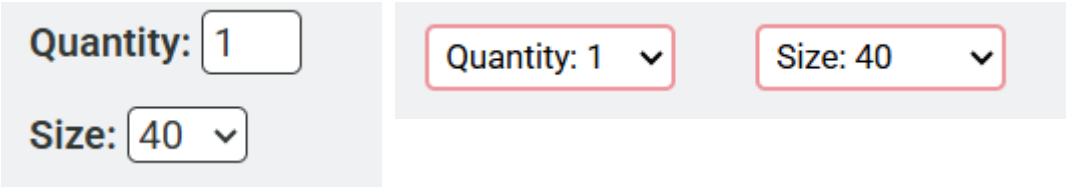
Se ha rediseñado la muestra de los colores en toda la web, optando finalmente por **radio buttons** personalizados, añadiendo uno por cada color presente en la base de datos. Además, se ha personalizado su selector, mostrando un check (✔) que, según el color, se mostrará de color **blanco o negro** para verse con la mejor claridad posible.

Es importante que cada color tenga su clase con su nombre, ya que eso es lo que nos sirve para hacerlo dinámico con el nombre de los colores que leemos desde la base de datos.

```css
/*RADIO BUTTONS*/
.color-radio {
    appearance: none;
    width: 20px;
    height: 20px;
    border-radius: 50%;
    border: 2px solid var(--onyx);
    cursor: pointer;
    position: relative;
    display: inline-block;
}

/*Colores que están en la base de datos -- Ir añadiendo según necesidad*/
.color-radio.gold { background-color: gold; }
.color-radio.black { background-color: black; }
.color-radio.white { background-color: white; }
.color-radio.silver { background-color: silver; }
.color-radio.purple { background-color: purple; }
.color-radio.blue { background-color: blue; }
.color-radio.red { background-color: red; }
.color-radio.yellow { background-color: yellow; }
.color-radio.green { background-color: green; }
.color-radio.orange { background-color: orange; }
.color-radio.pink { background-color: pink; }

/*Estilo del check (✓) que se muesrta al hacer click*/
.color-radio::before {
    content: "✓";
    font-size: 16px;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%) scale(0);
    transition: transform 0.2s;
}
```

```css
/*Estilo del check (✓) que se muesrta al hacer click*/
.color-radio::before {
    content: "✓";
    font-size: 16px;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%) scale(0);
    transition: transform 0.2s;
}

/*Grupo de colores que tienen el check (✓) de color BLANCO*/
.color-radio.black:checked::before,
.color-radio.purple:checked::before,
.color-radio.blue:checked::before,
.color-radio.red:checked::before,
.color-radio.green:checked::before {
    color: white;
}

/*Grupo de colores que tienen el check (✓) de color NEGRO*/
.color-radio.white:checked::before,
.color-radio.silver:checked::before,
.color-radio.yellow:checked::before,
.color-radio.gold:checked::before,
.color-radio.pink:checked::before,
.color-radio.orange:checked::before {
    color: black;
}

/*Animación para el check (✓)*/
.color-radio:checked::before {
    transform: translate(-50%, -50%) scale(1);
}
```

También, respecto a lo mostrado en la entrega anterior, para el carrito se ha reimaginado el estilo de los **select**, dándoles una apariencia más acorde a los colores de la página, un tamaño idéntico y haciendo que el input de **cantidad**, pase a ser también un select, donde el usuario podrá comprar de **1 a 10 unidades** para cada producto.



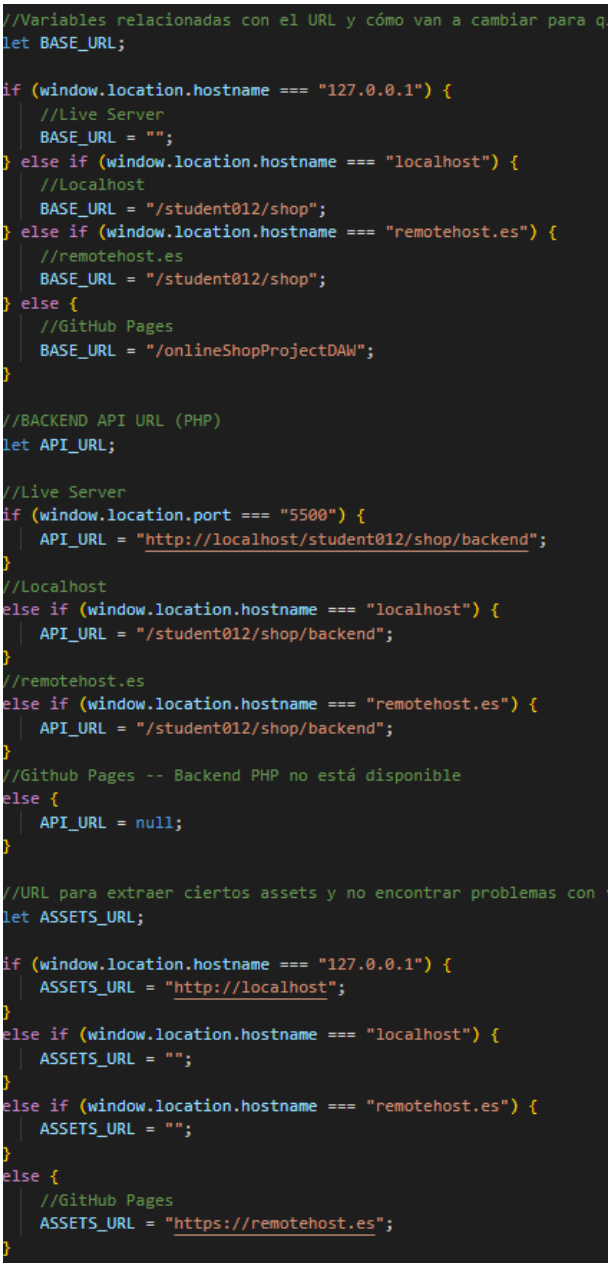
SASS - MIXINS UTILIZADOS

Aunque se ha procurado no abusar de esta funcionalidad, ya que al final los elementos tienden a sufrir pequeñas modificaciones en cada breakpoint, se ha tratado de elaborar **tres mixins**, dos de ellos para la **estructura básica** de la página (header, main y footer) y otro para los **productos del carrito**, dándoles un ancho, un borde y un padding común para que su diseño, tal y como se había estipulado en la primera parte del proyecto, sea altamente similar en todas las versiones.

```scss
/*Header y footer*/
@mixin header-footer($width, $bg-color){
  width: $width;
  background-color: $bg-color;
  padding: 20px;
}

/*Main*/
@mixin main($min-height, $color, $bg-color){
  min-height: calc(100vh - 140px);
  color: $color;
  background-color: $bg-color;
}

/*Cartas de producto en el shopping cart*/
@mixin product-card($width, $border, $border-radius){
  width: $width;
  border: $border;
  border-radius: $border-radius;
  padding: 1em;
}
```

PROBLEMAS ENCONTRADOS

Para hacer todo el **contenido dinámico**, se ha intentado llevarlo más allá de **Localhost o Remotehost**, tratando de que también funcionase en **Live Server y Github Pages**, algo que ha resultado imposible (al menos con lo que he probado), dado que desde esos entornos no hay manera de comunicarse bien con la base de datos y llevar a cabo todas las funcionalidades necesarias.



```
//Variables relacionadas con el URL y cómo van a cambiar para q
let BASE_URL;

if (window.location.hostname === "127.0.0.1") {
    //Live Server
    BASE_URL = "";
} else if (window.location.hostname === "localhost") {
    //Localhost
    BASE_URL = "/student012/shop";
} else if (window.location.hostname === "remotehost.es") {
    //remotehost.es
    BASE_URL = "/student012/shop";
} else {
    //GitHub Pages
    BASE_URL = "/onlineShopProjectDAW";
}

//BACKEND API URL (PHP)
let API_URL;

//Live Server
if (window.location.port === "5500") {
    API_URL = "http://localhost/student012/shop/backend";
}
//Localhost
else if (window.location.hostname === "localhost") {
    API_URL = "/student012/shop/backend";
}
//remotehost.es
else if (window.location.hostname === "remotehost.es") {
    API_URL = "/student012/shop/backend";
}
//Github Pages -- Backend PHP no está disponible
else {
    API_URL = null;
}

//URL para extraer ciertos assets y no encontrar problemas con
let ASSETS_URL;

if (window.location.hostname === "127.0.0.1") {
    ASSETS_URL = "http://localhost";
}
else if (window.location.hostname === "localhost") {
    ASSETS_URL = "";
}
else if (window.location.hostname === "remotehost.es") {
    ASSETS_URL = "";
}
else {
    //GitHub Pages
    ASSETS_URL = "https://remotehost.es";
}
```

Aparte, se ha tenido que aprender a **manejar cookies** para poder jugar con las tablas en la base de datos, tanto para la de clientes (generar un customer_id nuevo) como para la del carrito (rellenar todos los datos necesarios).

Siguiendo con la generación del carrito, su estructura imposibilita tener el mismo producto repetido pero con distinta talla o color, un error cuyo origen se ha detectado en la base de datos y que se desarrolla en el apartado de "Futuras mejoras e implementaciones".

Por último, destacar la dificultad de generar todos los archivos necesarios, tanto en PHP como en JavaScript, para que todo funcione como es debido y manteniendo cierto orden en la estructura de directorios del proyecto.

Respecto a **SASS**, no ha habido ninguna dificultad reseñable más allá de encontrar casos de uso para los mixins.

FUTURAS MEJORAS E IMPLEMENTACIONES

- Añadir un **contador al icono del carrito** para mostrar en todo momento el número de productos presentes en él.
- Hacer que el apartado "Similar product" del producto en detalle también se genere de **forma dinámica** en base a alguna condición por determinar (mostrar modelos de la misma marca o del mismo jugador, por ejemplo).

- **Desarrollar la lista de deseos en la base de datos** para así poder hacerla dinámica en el Frontend, mostrando así una lista verdadera en la página del carrito.
- Siguiendo con lo anterior, **añadir funcionalidad al botón de añadir al carrito para los elementos de la lista de deseos** cuando estos se generen de manera dinámica.
- **Reestructurar la tabla *012_shopping_cart*** para que la **clave primaria** no solo contenga el *customer_id* y el *product_id*, sino también los apartados *color* y *size*, ya que sino resulta imposible añadir el mismo producto con distinta talla o color, ya que la clave primaria se repite. Este cambio, no obstante, se deberá realizar a la vuelta de vacaciones de Navidad para que el profesor de entorno servidor actualice la base de datos con las modificaciones pertinentes.
- **Replantear el diseño de la página principal**, dado que desde allí no se puede especificar color, talla y cantidad al añadir el producto. Simplemente se hace click en el botón y el producto se añade sin color seleccionado, con la cantidad mínima (1) y con la talla mínima (40).