DPTO INFORMÁTICA – IES TRAFALGAR

MÓDULO – PROYECTO INTEGRADO

C.F.G.S. DAW

ECOMMERCE

Autor: Judit Quirós Violero

Fecha: 25/04/2025

Tutor: José Antonio Rodríguez Jiménez

HOJA DE RESUMEN

Título del proyecto: Ecommerce	
Autor: Judit Quirós Violero	**Fecha:** 25/04/2025
Tutor: José Antonio Rodríguez Jiménez	
Ciclo Formativo: DAW	**Curso:** 2024/2025
Palabras clave: Laravel, Livewire, ecommerce, MySQL, autenticación, carrito de compras, pedidos, accesibilidad, PHP, pruebas automatizadas.	
Resumen del proyecto: Este proyecto consiste en el desarrollo de una tienda online utilizando el framework Laravel con Livewire, como parte de un trabajo académico del ciclo de Desarrollo de Aplicaciones Web. El sistema incluye funcionalidades típicas de un ecommerce, como la gestión de productos, carrito de compras, pedidos y control de roles para usuarios y administradores. Se ha implementado una base de datos relacional en MySQL y se han considerado aspectos clave como la accesibilidad web y las pruebas de funcionamiento mediante herramientas como PHPUnit, Cypress y ciertas extensiones de Chrome. El objetivo es crear una base sólida que pueda ampliarse o adaptarse a futuros proyectos reales.	

ÍNDICE

INTRODUCCIÓN

1.1 - JUSTIFICACIÓN

Se ha optado por desarrollar una tienda online de ropa como ejemplo clásico de ecommerce, ya que involucra una amplia gama de funcionalidades comunes en este tipo de plataformas, como la gestión de usuarios, productos, pedidos, control de stock, carrito de la compra y control de roles de usuario.

Esta elección permite abordar tanto aspectos funcionales como técnicos, permitiendo aplicar soluciones a retos típicos de desarrollo web profesional, tales como la optimización de la experiencia de usuario y la implementación de un diseño accesible.

Aunque el proyecto tiene un carácter académico, su diseño y desarrollo se fundamentan en principios que lo convierten en una base sólida para futuros proyectos comerciales reales, no solo de ropa, sino de otros sectores del comercio online. Además, la implementación se realiza utilizando tecnologías como **Laravel** y **Livewire**, que ofrecen una estructura robusta y eficiente para crear aplicaciones dinámicas y fáciles de mantener.

1.2 - OBJETIVOS DEL PROYECTO

1.2.1 - OBJETIVO GENERAL

El objetivo principal de este proyecto es aplicar los conocimientos adquiridos a lo largo del ciclo formativo para desarrollar una tienda online completamente funcional, simulando un entorno real de desarrollo y despliegue de aplicaciones.

Se busca construir una plataforma web dinámica, escalable y fácil de mantener, aprovechando al máximo las capacidades de las tecnologías utilizadas para garantizar un rendimiento óptimo y una experiencia de usuario fluida.

1.2.2 - OBJETIVOS ESPECÍFICOS

A continuación, se detallan los objetivos específicos que orientarán el desarrollo de la tienda online, buscando garantizar su funcionalidad, accesibilidad y mantenimiento adecuado:

- Implementar un sistema de autenticación de usuarios.
- Crear un CRUD completo para productos, incluyendo subida de imágenes.
- Desarrollar la funcionalidad de carrito de la compra.
- Gestionar los pedidos con control de estado y edición.
- Implementar control de roles (administrador y usuario normal).
- Asegurar que la aplicación sea accesible siguiendo criterios **WCAG**.
- Realizar pruebas funcionales y de usabilidad para validar el proyecto.
- Documentar todas las decisiones técnicas y funcionales.

ESTUDIO DE VIABILIDAD

1.1 - VIABILIDAD TÉCNICA

Laravel es un framework robusto, muy bien documentado y con gran comunidad. Al haber sido utilizado en clase, su curva de aprendizaje ha sido manejable. Livewire se ha integrado para manejar interacciones dinámicas sin recargar la página, lo que mejora la experiencia de usuario. Se usa MySQL como base de datos y XAMPP como entorno de desarrollo. Además, el proyecto incluye archivos de Vite y Volt, que Laravel incorpora para la gestión de recursos y herramientas de interfaz moderna.

1.2 - VIABILIDAD ECONÓMICA

Desde el inicio del proyecto, se ha garantizado un enfoque sostenible y eficiente en términos económicos. Todo el desarrollo se ha llevado a cabo en un entorno local utilizando herramientas de código abierto, lo que ha permitido prescindir de cualquier inversión en licencias o plataformas de pago. Esto incluye el uso de tecnologías como **Laravel**, **Livewire**, **MySQL**, y servidores locales como **XAMPP**. Gracias a ello, **el proyecto no ha generado ningún coste económico directo hasta la fecha.**

En caso de que el proyecto requiera ser desplegado en un entorno de producción, la inversión seguiría siendo mínima. Únicamente se contemplan los costes asociados a la contratación de un servicio de **hosting compatible con Apache** (servidor que será utilizado para el despliegue) y la compra de un **dominio web**, cuyos precios en el mercado son muy accesibles. Actualmente, existen opciones confiables que ofrecen ambos servicios por una tarifa anual media inferior a los **50 €**, lo que representa una inversión perfectamente asumible para asegurar la disponibilidad pública del sitio.

En resumen, la viabilidad económica del proyecto está plenamente asegurada, tanto en su fase de desarrollo como en una futura fase de despliegue con Apache, y

responde al objetivo de construir una solución funcional, profesional y accesible sin comprometer la eficiencia de los recursos utilizados.

1.3 - VIABILIDAD OPERATIVA

El sistema ha sido diseñado y desarrollado para ofrecer una experiencia fluida, accesible y funcional tanto para usuarios administradores como para usuarios normales. Durante el desarrollo, se ha priorizado la **usabilidad y la claridad de la interfaz**, la cual ha sido construida utilizando **Bootstrap**, un framework frontend ampliamente adoptado que garantiza **diseños responsivos y accesibles desde distintos dispositivos**.

El entorno de desarrollo se ha configurado en **Windows 11**, utilizando el editor **Visual Studio Code**, que proporciona una experiencia ágil y productiva para la programación. Las pruebas del sistema se han llevado a cabo en el navegador **Google Chrome**, en su **última versión estable**, lo cual garantiza la compatibilidad con los estándares actuales de la web y asegura una visualización correcta de todos los elementos del sitio.

Además, el sistema ya cuenta con un **redireccionamiento automático implementado según el rol del usuario**. Esto significa que, tras iniciar sesión, cada perfil es dirigido automáticamente a su panel correspondiente (administrador o usuario), lo que mejora tanto la **experiencia de navegación** como la **seguridad operativa**, al impedir accesos no autorizados a secciones restringidas.

El sistema se ha mantenido estable durante las fases de prueba, lo que confirma su **viabilidad operativa** para ser utilizado en un entorno real. Tanto la arquitectura del software como las decisiones técnicas tomadas permiten que el proyecto pueda escalar o adaptarse fácilmente en caso de requerir nuevas funcionalidades en el futuro.

ALTERNATIVAS Y SELECCIÓN DE LA SOLUCIÓN

Antes de optar por **Laravel + Livewire**, se analizaron las siguientes alternativas:

- **CMS como WordPress + WooCommerce**: son rápidos de implementar y ofrecen soluciones listas para usar, pero presentan limitaciones importantes a la hora de personalizar funcionalidades específicas del negocio, especialmente si se requiere un control más detallado del backend.

- **Frameworks como Django, Node.js, Angular o React**: si bien ofrecen gran potencia y versatilidad, no fueron el enfoque del curso y su curva de aprendizaje era más pronunciada para los tiempos del proyecto. Además, requerían una mayor integración manual de aspectos como la autenticación, las migraciones y la estructura de base de datos.

- **Symfony**: se evaluó como una alternativa dentro del ecosistema PHP, pero resultó más complejo de configurar y utilizar en comparación con Laravel. Durante el curso, se detectaron gran cantidad de errores y una curva de aprendizaje más alta. Por ello, se descartó en favor de una solución más accesible.

- **Laravel + Livewire**: fue la opción elegida por ser un framework trabajado en clase, contar con una documentación extensa y amigable, y ofrecer herramientas integradas para funciones clave como autenticación, control de roles, migraciones y más. Además, Livewire permitió añadir interactividad en el frontend sin tener que recurrir a frameworks JavaScript complejos, lo que facilitó enormemente el desarrollo.

ANÁLISIS Y DISEÑO DE LA SOLUCIÓN ADOPTADA

1.1 - ESTRUCTURA DE LA BASE DE DATOS

Se diseñó una base de datos relacional en MySQL compuesta por las siguientes tablas:

- **Users**: contiene los datos básicos del usuario y su rol.
- **products**: almacena la información del producto (nombre, descripción, color, estilo, categoría, género, talla, stock y precio).
- **images**: almacena múltiples imágenes por producto.
- **sizes** y **product_size**: permiten manejar múltiples tallas por producto con su respectivo stock.
- **carts** y **cart_items**: gestionan el carrito de cada usuario con sus productos seleccionados y cantidades.
- **orders** y **order_items**: gestionan los pedidos realizados, permitiendo ver productos, cantidades, tallas y precios, además del estado del pedido.

1.2 - FUNCIONALIDADES PRINCIPALES

- Registro, login y autenticación con Laravel.
- Vista del catálogo de productos filtrables (Vista de la tienda).
- Carrito de compras dinámico usando Livewire:
    - Visualización de los productos en el carrito.
    - Permite seleccionar talla, cantidad y eliminar productos en tiempo real.
- Vista de resumen del carrito con total y botón de pedido siempre visible.
- Panel de administración para:
    - Crear, editar, eliminar productos con subida de imágenes.
    - Visualización de todos los productos.
    - Buscador de producto por id o nombre.
    - Ver pedidos con detalles por usuario.

- Cambiar el estado del pedido desde un `<select>` directamente.
- Eliminar productos de un pedido.

1.3 - CONTROL DE ROLES

El sistema ya implementa un control de roles que distingue entre usuario normal y administrador. Al iniciar sesión, cada usuario es redirigido automáticamente a su panel correspondiente, lo que mejora la seguridad y la experiencia de navegación al restringir el acceso a secciones no autorizadas.

1.4 - FRONTEND Y HERRAMIENTAS ADICIONALES

En las vistas que he creado yo misma, he usado principalmente **Bootstrap** y CSS puro para diseñar y maquetar, buscando un control más directo sobre el estilo.

Las vistas que vienen por defecto con Laravel, en cambio, emplean tecnologías diferentes como **Tailwind CSS** junto con **Flux 1.4** para el manejo de estilos y diseño. Además, Laravel utiliza **Vite** para compilar y optimizar los archivos de frontend.

También se incluye **Laravel Volt** en algunas rutas, que es una herramienta visual moderna para facilitar el desarrollo con Livewire, aunque no se ha utilizado mucho en este proyecto.

IMPLEMENTACIÓN

La implementación de este proyecto ecommerce abarca desde la configuración de la base de datos hasta el desarrollo de la interfaz y la lógica del sistema, integrando funcionalidades clave de forma segura, eficiente y fácil de usar. Se diseñaron tablas para gestionar usuarios, productos y pedidos, y se construyeron vistas personalizadas con Bootstrap y CSS puro, en contraste con las vistas por defecto de Laravel. Además, se utilizó Livewire para crear componentes dinámicos que mejoran la experiencia del usuario.

1.1 - ESTRUCTURA DEL PROYECTO

1.1.1 - CARPETAS IMPORTANTES

- Componentes middleware:
    - /app/Http/Middleware
- Componentes livewire, junto con sus vistas:
    - /app/Http/Livewire
    - /resources/views/livewire
- Mails:
    - /app/Http/Mail
- Modelos:
    - /app/Http/Models
- Migraciones:
    - /database/migrations
- Inserciones a la base de datos:
    - /database/seeders
- Vistas:
    - Componentes: /resources/views/components
    - Mensaje de los emails: /resources/views/emails
    - Usuarios: /resources/views/users
    - Administrador: /resources/views/admin
    - Autenticación: /resources/livewire/auth

- Rutas:
    - Generales: /routes/web.php
    - Autenticación: /routes/auth.php
- Donde se guardan las imágenes de la web:
    - Las que se cargan en el seed: /storage/app/public/fotos_productos
    - Las que se añaden desde la web: /storage/app/public/products

1.2 - BASE DE DATOS

1.2.1 - MODELOS

- Cart.php
    - Campos asignables:
        - **user_id:** Referencia al usuario dueño del carrito. Se usa para asignar el carrito al usuario correspondiente. `protected $fillable = ['user_id'];`
    - Relaciones:
        - items(): Es una relación uno a muchos (hasMany), que se relaciona con el modelo de CartItem, cuya clave foránea es “cart_id” en la tabla cart_items. En resumen: un carrito puede contener varios items (productos añadidos).

```php
public function items(): HasMany
{
    return $this->hasMany(related: CartItem::class);
}
```

        - user(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de User, cuya clave foránea es “user_id” en la tabla carts. En resumen: Cada carrito pertenece a un único usuario.

```php
public function user(): BelongsTo
{
    return $this->belongsTo(related: User::class);
}
```

- Métodos personalizados:
  - addProduct(int $productId, int $sizeId, int $quantity = 1): Añade un producto al carrito con la talla indicada. Si ya existe ese producto con esa talla, simplemente aumenta la cantidad, y si no está, se crea un nuevo carrito (CartItem).

```php
public function addProduct(int $productId, int $sizeId, int $quantity = 1): void
{
    $cartItem = $this->items()
        ->where(column: 'product_id', operator: $productId)
        ->where(column: 'size_id', operator: $sizeId)
        ->first();

    if ($cartItem) {
        $cartItem->quantity += $quantity;
        $cartItem->save();
    } else {
        $this->items()->create(attributes: [
            'product_id' => $productId,
            'size_id' => $sizeId,
            'quantity' => $quantity,
        ]);
    }
}
```

- CartItem.php
  - Campos asignables:
    - **cart_id**: Referencia al carrito al que pertenece este ítem.
    - **product_id**: ID del producto añadido al carrito.
    - **size_id**: ID de la talla seleccionada para ese producto.
    - **quantity**: Cantidad de unidades del producto con esa talla.

```php
protected $fillable = ['cart_id', 'product_id', 'size_id', 'quantity'];
```

  - Relaciones:
    - cart(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de Cart, cuya clave foránea es “cart_id” en la tabla cart_items. En resumen: el ítem pertenece a un carrito específico.

```php
public function cart(): BelongsTo
{
    return $this->belongsTo(related: Cart::class);
}
```

        - product(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de Product, cuya clave foránea es “product_id” en la tabla cart_items. En resumen: el ítem del carrito está relacionado con un producto de la tienda.

```php
public function product(): BelongsTo
{
    return $this->belongsTo(related: Product::class);
}
```

        - size(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de Size, cuya clave foránea es “size_id” en la tabla cart_items. En resumen: el ítem del carrito está asociado a una talla específica del producto.

```php
public function size(): BelongsTo
{
    return $this->belongsTo(related: Size::class);
}
```

- Image.php
    - Campos asignables:
        - **product_id**: ID del producto al que pertenece la imagen.
        - **url**: Ruta o dirección de la imagen almacenada.

```php
protected $fillable = ['product_id', 'url'];
```

    - Relaciones:
        - product(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de Product, cuya clave foránea es “product_id” en la tabla Images. En resumen: cada imagen está asociada a un único producto.

```php
public function product(): BelongsTo
{
    return $this->belongsTo(related: Product::class);
}
```

- Order.php
    - Traits usados:
        - HasFactory: Habilita el uso de factorías para generar instancias de prueba.

```php
use HasFactory;
```

- Campos asignables:
  - **user_id**: ID del usuario que realizó el pedido.
  - **total**: Total monetario del pedido (considerando descuentos).
  - **status**: Estado actual del pedido (por ejemplo: pendiente, enviado, cancelado).

```php
protected $fillable = [
    'user_id',
    'total',
    'status',
];
```

- Relaciones:
  - user(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de User, cuya clave foránea es “user_id” en la tabla orders. En resumen: el pedido pertenece a un usuario específico.

```php
public function user(): BelongsTo
{
    return $this->belongsTo(related: User::class);
}
```

  - items(): Es una relación uno a muchos (hasMany), que se relaciona con el modelo de OrderItem, cuya clave foránea es “order_id ” en la tabla order_items. En resumen: el pedido contiene múltiples productos, incluyendo cantidad, talla y precio unitario.

```php
public function items(): HasMany
{
    return $this->hasMany(related: OrderItem::class);
}
```

- Métodos personalizados:
  - createFromCartItems($user, $cartItems): Crea un pedido nuevo a partir de los ítems del carrito de un usuario. Calcula el total (aplicando descuentos), registra cada ítem como OrderItem, actualiza el stock en la tabla pivot product_size y elimina los ítems del carrito. Todo se hace dentro de una transacción para garantizar la integridad de los datos.

```php
public static function createFromCartItems($user, $cartItems): mixed
{
    return DB::transaction(callback: function () use ($user, $cartItems): Order {

        // Calculamos el total con descuentos
        $total = 0;
        foreach ($cartItems as $item) {
            $price = $item->product->discount > 0
                ? round(num: $item->product->price * (1 - $item->product->discount / 100), precision: 2)
                : $item->product->price;

            $total += $price * $item->quantity;
        }

        // Creamos el pedido
        $order = self::create(attributes: [
            'user_id' => $user->id,
            'total' => $total,
            'status' => 'pendiente',
        ]);
```

```php
        // Creamos los order_items y actualizamos stock
        foreach ($cartItems as $item) {
            $price = $item->product->discount > 0
                ? round(num: $item->product->price * (1 - $item->product->discount / 100), precision: 2)
                : $item->product->price;

            OrderItem::create(attributes: [
                'order_id' => $order->id,
                'product_id' => $item->product_id,
                'size_id' => $item->size_id,
                'quantity' => $item->quantity,
                'price' => $price,
            ]);

            // Reducir stock en tabla pivot product_size
            DB::table(table: 'product_size')
                ->where(column: 'product_id', operator: $item->product_id)
                ->where(column: 'size_id', operator: $item->size_id)
                ->decrement(column: 'stock', amount: $item->quantity);

            // Eliminar item del carrito
            $item->delete();
        }

        return $order;
    });
}
```

- OrderItem.php
  - Traits usados:
    - HasFactory: Habilita el uso de factorías para generar instancias de prueba.

```php
use HasFactory;
```

  - Campos asignables:
    - **order_id:** ID del pedido al que pertenece este ítem
    - **product_id**: ID del producto comprado
    - **size_id**: ID de la talla seleccionada del producto
    - **quantity**: Cantidad de este producto en el pedido
    - **price**: Precio unitario del producto en el momento del pedido

```php
protected $fillable = [
    'order_id',
    'product_id',
    'size_id',
    'quantity',
    'price',
];
```

  - Relaciones:
    - order(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de Order, cuya clave foránea es “order_id” en la tabla order_items. En resumen: un producto/ítem pertenece a un pedido específico.

```php
public function order(): BelongsTo
{
    return $this->belongsTo(related: Order::class);
}
```

        - product(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de Product, cuya clave foránea es “product_id” en la tabla order_items. En resumen: el ítem del pedido está relacionado con un producto de la tienda.

```php
public function product(): BelongsTo
{
    return $this->belongsTo(related: Product::class);
}
```

        - size(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de Size, cuya clave foránea es “size_id” en la tabla order_items. En resumen: el ítem del pedido está asociado a una talla específica del producto.

```php
public function size(): BelongsTo
{
    return $this->belongsTo(related: Size::class);
}
```

- Product.php
    - Campos asignables:
        - name
        - description
        - color
        - gender
        - style
        - size
        - category
        - stock
        - price
        - discount

    - Relaciones:
        - items(): Es una relación uno a muchos (hasMany), que se relaciona con el modelo de CartItem, cuya clave foránea es “cart_id” en la tabla cart_items. En resumen: un carrito puede contener varios items (productos añadidos).

```php
public function items(): HasMany
{
    return $this->hasMany(related: CartItem::class);
}
```

        - user(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de User, cuya clave foránea es “user_id” en la tabla carts. En resumen: Cada carrito pertenece a un único usuario.

```php
public function user(): BelongsTo
{
    return $this->belongsTo(related: User::class);
}
```

- Métodos personalizados:
  - addProduct(int $productId, int $sizeId, int $quantity = 1): Añade un producto al carrito con la talla indicada. Si ya existe ese producto con esa talla, simplemente aumenta la cantidad, y si no está, se crea un nuevo carrito (CartItem).

- Cart.php
  - Campos asignables:
    - **user_id:** Referencia al usuario dueño del carrito. Se usa para asignar el carrito al usuario correspondiente. `protected $fillable = ['user_id'];`
  - Relaciones:
    - items(): Es una relación uno a muchos (hasMany), que se relaciona con el modelo de CartItem, cuya clave foránea es “cart_id” en la tabla cart_items. En resumen: un carrito puede contener varios items (productos añadidos).

```php
public function items(): HasMany
{
    return $this->hasMany(related: CartItem::class);
}
```

    - user(): Es una relación inversa uno a muchos (belongsTo), que se relaciona con el modelo de User, cuya clave foránea es “user_id” en la tabla carts. En resumen: Cada carrito pertenece a un único usuario.

```php
public function user(): BelongsTo
{
    return $this->belongsTo(related: User::class);
}
```

  - Métodos personalizados:

- addProduct(int $productId, int $sizeId, int $quantity = 1): Añade un producto al carrito con la talla indicada. Si ya existe ese producto con esa talla, simplemente aumenta la cantidad, y si no está, se crea un nuevo carrito (CartItem).

1.2.2 - MIGRACIONES

Roles definidos (ej: admin = 1, usuario = 0)

Middleware personalizado para control de acceso

Redirecciones post-login según el rol

Autenticación y verificación

1.2.3 - SEEDERS

Roles definidos (ej: admin = 1, usuario = 0)

Middleware personalizado para control de acceso

Redirecciones post-login según el rol

Autenticación y verificación

1.3 - COMPONENTES LIVEWIRE

En esta sección se describen todos los componentes Livewire implementados en el proyecto. Cada componente contiene una clase PHP ubicada en ***/app/Http/Livewire*** y su vista asociada en ***/resources/views/livewire***. A través de estos componentes se gestiona la interacción dinámica del usuario con el sistema, como la visualización de productos, gestión del carrito, administración de pedidos, entre otras funcionalidades.

1.3.1 - ADDTOCART.PHP

FUNCIONALIDAD: El componente **AddToCart** permite al usuario autenticado seleccionar una talla y una cantidad de un producto determinado, y añadirlo al carrito. Está diseñado

para ser utilizado en la página de la tienda donde se muestran todos los productos y en la que se muestran los detalles del producto.

- Variables:
  - **product**: instancia del modelo Product que representa el producto mostrado.
  - **sizeId**: ID de la talla seleccionada por el usuario.
  - **quantity**: cantidad de productos a añadir, con valor por defecto de 1.
  - **message y messageType:** usados para mostrar mensajes de éxito o error tras la acción de añadir al carrito.

```
2 references
public $product;
2 references
public $sizeId;
1 reference
public $quantity = 1;
```

```
5 references
public $message = null;
5 references
public $messageType = 'success';
```

- Método addToCart():
  - Verifica si el usuario está autenticado.
  - Valida que exista un producto y que se haya seleccionado una talla.

```
public function addToCart(): void
{
    // Obtener el usuario autenticado
    $user = auth()->user();

    // Validar que el usuario esté logueado
    if (!$user) {
        $this->message = 'Debes iniciar sesión para añadir productos al carrito.';
        $this->messageType = 'error';
        return; // Detener ejecución si no hay usuario
    }

    // Validar que el producto sea válido (exista)
    if (!$this->product) {
        $this->message = 'Producto no válido.';
        $this->messageType = 'error';
        return;
    }

    // Validar que se haya seleccionado una talla
    if (!$this->sizeId) {
        $this->message = 'Por favor selecciona una talla.';
        $this->messageType = 'error';
        return;
    }
```

  - Obtiene o crea un carrito asociado al usuario usando $user->getOrCreateCart().
  - Añade el producto al carrito con la talla y cantidad seleccionada usando $cart->addProduct(...), que está en el modelo del carrito: Cart.php
  - Muestra un mensaje de éxito o error según el resultado.

```php
    try {
        // Obtener o crear el carrito asociado al usuario
        $cart = $user->getOrCreateCart();

        // Añadir el producto con la talla y cantidad indicada al carrito
        $cart->addProduct(productId: $this->product->id, sizeId: $this->sizeId, quantity: $this->quantity);

        // Mensaje de éxito para el usuario
        $this->message = 'Producto añadido al carrito.';
        $this->messageType = 'success';

    } catch (\Exception $e) {
        // Capturar cualquier error y mostrar mensaje de error
        $this->message = 'Error: ' . $e->getMessage();
        $this->messageType = 'error';
    }
}
```

- Renderizado: Retorna la vista livewire.add-to-cart.

```php
public function render(): View
{
    return view(view: 'livewire.add-to-cart');
}
```

VISTA: La vista muestra el formulario interactivo que permite seleccionar talla y cantidad, y ejecutar la acción de añadir el producto al carrito. Incluye mensajes de alerta y consideraciones de accesibilidad.

- Secciones principales:
  Mensaje de estado ($message): Se muestra un div con clase alert y color dinámico según $messageType (success o danger).

```blade
@if ($message)
    <div id="mensaje" class="alert alert-{{ $messageType == 'success' ? 'success' : 'danger' }} mb-4"
        role="alert" aria-label="Mensaje del sistema">
        {{ $message }}
    </div>
@endif
```

- Selección de talla: Tiene un encabezado con texto “Selecciona una talla”, y una lista de radio buttons que se enlazan con wire:model="sizeId", y cuyo contenido es generado dinámicamente a partir de $product->sizes.

```blade
<p id="talla-encabezado" class="mt-3 mb-2 fw-bold" >Selecciona una talla:</p>
<div id="tallas" class="d-flex flex-wrap gap-2" role="radiogroup" aria-label="Opciones de tallas disponibles">
    @foreach ($product->sizes as $size)
        <div class="form-check">
            <input
                class="form-check-input border border-dark focus-ring"
                type="radio"
                wire:model="sizeId"
                id="size-{{ $size->id }}"
                name="size"
                value="{{ $size->id }}"
                aria-label="Talla {{ strtoupper(string: $size->name) }}"
                title="Selecciona la talla {{ strtoupper(string: $size->name) }}"
            >
            <label class="form-check-label" for="size-{{ $size->id }}">
                {{ strtoupper(string: $size->name) }}
            </label>
        </div>
    @endforeach
</div>
```

- Selección de la cantidad: Es un campo numérico enlazado con wire:model="quantity", con mínimo de 1.

```blade
<p id="cantidad-encabezado" class="mb-1 fw-bold">Cantidad:</p>
<div class="d-flex gap-2 justify-content-between align-items-end">
    <label for="cantidad" class="visually-hidden">Introduce la cantidad</label>
    <input
        type="number"
        id="cantidad"
        name="cantidad"
        wire:model="quantity"
        min="1"
        class="form-control border border-dark"
        aria-label="Campo para introducir la cantidad"
        title="Introduce la cantidad"
    />
```

- Botón “Añadir al carrito”: Al pulsarlo se ejecuta el método addToCart().
  Pero si el usuario no ha iniciado sesión (auth()->check()), el botón se deshabilita .

```blade
<button
    wire:click="addToCart"
    id="boton-add-cart"
    class="btn btn-primary form-control"
    {{ auth()->check() ? '' : 'disabled' }}
    aria-label="Añadir al carrito"
    title="Añadir al carrito"
>
    <i class="bi bi-cart-plus" aria-hidden="true"></i>
    Añadir al carrito
</button>
```

- Mensajes alternativos:

- Si no hay $product, se muestra mensaje de advertencia “Producto no disponible”.

```blade
<div class="alert alert-warning" role="alert" aria-label="Producto no disponible">
    Producto no disponible.
</div>
```

- Si el usuario no está autenticado, se muestra mensaje indicando que debe iniciar sesión para añadir productos.

```blade
@if (!auth()->check())
    <div id="alerta-login" class="d-flex justify-content-end mt-2">
        <div class="fw-bold alert alert-warning"
            style="font-size: x-small; color: black"
            role="alert"
            aria-label="Debes iniciar sesión para añadir productos al carrito"
            title="Debes iniciar sesión para añadir productos al carrito"
        >
            *Inicia sesión para poder añadir productos al carrito.
        </div>
    </div>
@endif
```

1.3.2 - ADMINDASHBOARD.PHP

FUNCIONALIDAD: Este componente representa el **dashboard del administrador**, mostrando un resumen general del sistema: productos, pedidos, usuarios, actividad reciente y alertas por bajo stock.

- Variables:
  - **lowStockProducts**: almacena productos que tienen alguna talla con stock igual o menor a 5.

```php
// Lista de productos con stock bajo en alguna talla
1 reference
public $lowStockProducts = [];
```

- mount(): Se ejecuta automáticamente al cargar el componente.
  - Filtra y carga todos los productos que tienen al menos una talla con stock <= 5.
  - Solo se cargan las tallas con bajo stock, optimizando el rendimiento y visualización.

```php
public function mount(): void
{
    // Filtrar productos que tengan al menos una talla con poco stock (<= 5)
    $this->lowStockProducts = Product::whereHas(relation: 'sizes', callback: function (Builder<TRelatedModel> $query)… {
        $query->where(column: 'product_size.stock', operator: '<=', value: 5); // Condición sobre la tabla intermedia product_size
    })
        ->with(relations: [
            'sizes' => function ($query): void {
                $query->where('product_size.stock', '<=', 5); // Cargar solo las tallas con bajo stock
            }
        ])
        ->get(); // Obtener los productos que cumplen con esa condición
}
```

- render(): Devuelve la vista livewire.admin-dashboard con los siguientes datos:
    - **products**: Todos los productos de la base de datos.
    - **orders:** Todos los pedidos registrados.
    - **users:** Todos los usuarios del sistema.
    - **recentOrders**: Los 5 pedidos más recientes.
    - **recentProducts**: Los 5 productos más recientes.

```php
public function render(): View
{
    return view(view: 'livewire.admin-dashboard', data: [
        'products' => Product::all(),
        'orders' => Order::all(),
        'users' => User::all(),
        'recentOrders' => Order::latest()->take(value: 5)->get(),
        'recentProducts' => Product::latest()->take(value: 5)->get(),
    ]);
}
```

VISTA: Muestra un resumen general del sistema: productos, pedidos, usuarios, actividad reciente y alertas por bajo stock.

- Encabezado: Muestra información del administrador autenticado: nombre, fecha de registro y enlace para editar su perfil.

```blade
<div
    class="flex items-center gap-6 p-6 bg-white dark:bg-neutral-900 rounded-xl border border-neutral-200 dark:border-neutral-700 shadow-md">
    <img src="https://www.shutterstock.com/image-vector/admin-icon-strategy-collection-thin-600nw-2307398667.jpg"
        alt="Foto de perfil" class="w-16 h-16 rounded-full border-4 border-indigo-500 shadow"
        style="width: 125px; height: 125px;">
    <div>
        <h2 class="text-2xl font-semibold text-gray-900 dark:text-white">¡Hola, {{ Auth::user()->name }}!</h2>
        <p class="text-gray-600 dark:text-gray-300">Administrador desde
            {{ ucfirst(string: Auth::user()->created_at->translatedFormat('F \d\e Y')) }}</p>
        <p class="mt-2 text-sm text-indigo-600 dark:text-indigo-400 font-medium"><a
                href="{{ route(name: 'settings.profile') }}">Editar perfil</a></p>
    </div>
</div>
```

- Tarjetas informativas: Visualiza el resumen de los productos, pedidos y usuarios.
    - Gestión de productos: total y enlace al crud de productos.

```
<div
    class="bg-white dark:bg-neutral-900 p-4 rounded-xl border border-neutral-200 dark:border-neutral-700 shadow-sm">
    <h3 class="text-lg font-semibold mb-2 text-gray-900 dark:text-white">Gestión de Productos</h3>
    @if ($products->count())
        <p class="text-sm text-gray-600 dark:text-gray-300">Total de productos en la tienda:
            {{ $products->count() }}</p>
    @else
        <p>No hay productos en la tienda.</p>
    @endif

    <a href="{{ route(name: 'admin.product') }}"
        class="text-sm text-indigo-600 dark:text-indigo-400 font-medium mt-2 inline-block">Ver productos</a>
</div>
```

- Gestión de pedidos: total y enlace al crud de pedidos.

```
<div
    class="bg-white dark:bg-neutral-900 p-4 rounded-xl border border-neutral-200 dark:border-neutral-700 shadow-sm">
    <h3 class="text-lg font-semibold mb-2 text-gray-900 dark:text-white">Gestión de Pedidos</h3>
    <p class="text-sm text-gray-600 dark:text-gray-300">Total de pedidos realizados: {{ $orders->count() }}
    </p>
    <a href="{{ route(name: 'admin.order') }}"
        class="text-sm text-indigo-600 dark:text-indigo-400 font-medium mt-2 inline-block">Ver pedidos</a>
</div>
```

- Usuarios registrados: total y enlace al crud de usuarios.

```
<div
    class="bg-white dark:bg-neutral-900 p-4 rounded-xl border border-neutral-200 dark:border-neutral-700 shadow-sm">
    <h3 class="text-lg font-semibold mb-2 text-gray-900 dark:text-white">Usuarios Registrados</h3>
    <p class="text-sm text-gray-600 dark:text-gray-300">Total de usuarios: {{ $users->count() }}</p>
    <a href="{{ route(name: 'admin.users') }}"
        class="text-sm text-indigo-600 dark:text-indigo-400 font-medium mt-2 inline-block">Ver
        usuarios</a>
</div>
```

- Actividad reciente: Contiene una lista de los pedidos más recientes y su fecha, y los nuevos productos que se han añadido recientemente.

```
<div
    class="bg-white dark:bg-neutral-900 p-6 rounded-xl border border-neutral-200 dark:border-neutral-700 shadow-md"
    <h3 class="text-xl font-semibold mb-4 text-gray-900 dark:text-white">Actividad reciente</h3>
    <ul class="space-y-2 text-sm text-gray-700 dark:text-gray-300">
        @foreach ($recentOrders as $order)
            <li>📦 Pedido #{{ $order->id }} realizado el {{ $order->created_at->format('d/m/Y') }}</li>
        @endforeach
        @foreach ($recentProducts as $product)
            <li>🛒 Producto {{ $product->name }} añadido el {{ $product->created_at->format('d/m/Y') }}</li>
        @endforeach
    </ul>
</div>
```

- Mensaje de alerta de bajo stock:

Si hay productos con tallas en stock crítico (≤5), se muestra una alerta con el nombre del producto, talla y cantidad restante.

Si no hay problemas de stock, se muestra un mensaje positivo de confirmación.

```
@if ($lowStockProducts->count() > 0)
    <div class="alert alert-warning shadow-sm">
        <h5><i class="bi bi-exclamation-triangle-fill me-2"></i>Productos con poco stock:</h5>
        <ul class="mb-0">
            @foreach ($lowStockProducts as $product)
                <li>
                    <strong>{{ $product->name }}</strong>
                    <ul>
                        @foreach ($product->sizes as $size)
                            @if ($size->pivot->stock <= 5)
                                <li>
                                    <strong>{{ $size->name }}:</strong> {{ $size->pivot->stock }} unidades
                                    restantes
                                </li>
                            @endif
                        @endforeach
                    </ul>
                </li>
            @endforeach
        </ul>
    </div>
@else
    <div class="alert alert-success shadow-sm">
        <i class="bi bi-check-circle-fill me-2"></i>Todos los productos tienen suficiente stock.
    </div>
@endif
```

1.3.3 - INDEXPAGE.PHP

FUNCIONALIDAD:

- Variables:
  - **categories**: Lista de categorías únicas de productos.
  - **newProducts**: Últimos productos añadidos.
  - **discountedProducts**: Productos con descuento.

```
public $categories = [];
1 reference
public $newProducts = [];
1 reference
public $discountedProducts = [];
```

- mount(): Se ejecuta automáticamente al iniciar el componente y realiza las siguientes tareas:

Carga de categorías únicas: Extrae las categorías distintas de la base de datos desde el modelo Product, elimina los valores nulos, las capitaliza, las ordena alfabéticamente y reinicia los índices del array para facilitar su uso en la vista.

```
public function mount(): void
{
    // Obtener las categorías únicas no nulas, capitalizarlas, ordenarlas alfabéticamente y resetear los índices
    $this->categories = Product::whereNotNull(columns: 'category')
        ->distinct()
        ->pluck(column: 'category')   // Trae solo los valores únicos de la columna 'category'
        ->map(callback: fn($cat): string => ucfirst(string: $cat))    // Capitaliza la primera letra
        ->sort()    // Ordena alfabéticamente
        ->values();     // Reinicia los índices para que el array sea limpio
```

Carga de productos recientes: Recupera los 10 productos más recientes mediante el método latest() y los guarda en la propiedad $newProducts para mostrarlos en el carrusel.

```
    // Obtener los 10 productos más recientes
    $this->newProducts = Product::latest()->take(value: 10)->get();
```

Carga de productos en oferta: Filtra los productos con un valor de discount mayor que 0, obtiene los primeros 10, y les calcula un campo adicional final_price que representa el precio con el descuento aplicado. Estos productos se almacenan en $discountedProducts.

```
    // Obtener los productos con descuento y calcular su precio final
    $this->discountedProducts = Product::where(column: 'discount', operator: '>', value: 0)
        ->take(value: 10)
        ->get()
        ->map(callback: function (Product $product): Product {
            // Calcular el precio con descuento y asignarlo al producto
            $product->final_price = round(num: $product->price * (1 - $product->discount / 100), precision: 2);
            return $product;
        });
}
```

- getProductDescription(Product $product): Genera una descripción aleatoria personalizada para cada producto, según su categoría.

```
public function getProductDescription($product): string
{
    $isShoe = strtolower(string: $product->category) === 'zapatos'; // Verifica si el producto es de categoría "zapatos"

    // Frases para productos de calzado
    $shoeTexts = [
        "¡Descubre el nuevo modelo de zapatos que está marcando tendencia! Hechos para acompañarte con estilo y comodidad todo el día.",
        "¿Buscas un calzado único? Estos zapatos son la combinación perfecta de diseño moderno y calidad.",
        "¡Tu próxima pisada con estilo empieza aquí! Disponible en tallas " . $product->sizes->pluck('name')->join(', ') . ".",
        "Dale un giro a tu estilo con este calzado nuevo. Comodidad y diseño que se adapta a ti.",
        "¿A la moda y cómodo? Estos zapatos tienen todo lo que necesitas para tus pasos diarios.",
        "Elige tu talla favorita y camina con seguridad. Disponible en: " . $product->sizes->pluck('name')->join(', ') . ".",
    ];

    // Frases para otras prendas (ropa en general)
    $clothingTexts = [
        "Renueva tu armario con esta prenda imprescindible. Ideal para combinar con cualquier look.",
        "¡Moda que habla por ti! Esta novedad en " . strtolower(string: $product->category) . " te hará destacar.",
        "Diseño, estilo y comodidad en una sola prenda. Disponible en tallas " . $product->sizes->pluck('name')->join(', ') . ".",
        "Perfecta para cualquier ocasión. Dale un toque fresco y actual a tu outfit.",
        "Te encantará cómo se ve y se siente esta nueva prenda. ¡No puede faltar en tu colección!",
        "Confeccionada con materiales de calidad. Disponible en: " . $product->sizes->pluck('name')->join(', ') . ".",
    ];

    // Devuelve un texto aleatorio dependiendo de si es zapato u otra prenda
    return $isShoe
        ? $shoeTexts[array_rand(array: $shoeTexts)]
        : $clothingTexts[array_rand(array: $clothingTexts)];
}
```

  - Si el producto pertenece a la categoría "zapatos", se selecciona una frase aleatoria del array $shoeTexts.
  - Si es cualquier otra categoría, se usa una frase del array $clothingTexts.
  - Las frases incluyen menciones dinámicas a las tallas disponibles usando pluck('name')->join(', ').

- render(): Devuelve la vista livewire.index-page.

```
public function render(): View
{
    return view(view: 'livewire.index-page');
}
```

VISTA:

- Carrusel de Novedades: Utiliza Bootstrap para mostrar el carrusel, y en él se muestran los productos más recientes.

```blade
<div class="d-flex justify-content-center position-relative my-4">
    <div id="carouselNovedades" class="carousel slide w-75" data-bs-ride="carousel">
        <div class="carousel-inner">
            @foreach ($newProducts as $index => $product)
                <div class="carousel-item @if ($index == 0) active @endif">
                    <div class="row g-0 align-items-center mb-5">
                        <!-- Imagen con cartel de Novedades -->
                        <div
                            class="col-12 col-md-6 position-relative d-flex justify-content-center align-items-center">
                            <img src="{{ asset(path: 'storage/' . optional(value: $product->images->first())->url) }}"
                                class="img-fluid w-100 object-fit-cover rounded-start" style="max-height: 600px"
                                alt="{{ $product->name }}">
                        </div>

                        <!-- Info del producto al lado -->
                        <div
                            class="col-12 col-md-6 d-flex flex-column justify-content-center gap-2 align-items-center text-center p-4">
                            <h3 class="fw-bold">{{ $product->name }}</h3>
                            <p class="text-muted fs-5 fst-italic">{{ $this->getProductDescription($product) }}</p>
                            <a href="{{ route(name: 'productos.show', parameters: $product->id) }}" class="btn btn-primary mt-3">Ver
                                Producto</a>
                        </div>
                    </div>
                </div>
            @endforeach
        </div>
    </div>
```

Cada diapositiva incluye una imagen destacada, el nombre del producto, una descripción generada dinámicamente, y un botón para ver el producto completo.

- Sección Informativa: Muestra cuatro tarjetas con íconos representativos (estilo, precios, tallas, envío) y un pequeño texto explicativo de los valores diferenciales de la tienda. Está puesto de relleno para la página de inicio.

```blade
<h2 class="mb-4 text-center fw-bold">Lo que nos hace únicos</h2>
<div class="row my-5 text-center">
    <div class="col-md-3 mb-4">
        <i class="bi bi-palette-fill" style="font-size: 60px;"></i>
        <h3 class="fw-bold h5 mt-3">Variedad de Estilos</h3>
        <p>Desde casual hasta elegante, para todos los gustos y ocasiones.</p>
    </div>
    <div class="col-md-3 mb-4">
        <i class="bi bi-tags" style="font-size: 60px;"></i>
        <h3 class="fw-bold h5 mt-3">Precios Accesibles</h5>
            <p>Moda de calidad que cuida tu bolsillo sin sacrificar estilo.</p>
    </div>
    <div class="col-md-3 mb-4">
        <i class="bi bi-rulers" style="font-size: 60px;"></i>
        <h3 class="fw-bold h5 mt-3">Tallas para Todos</h5>
            <p>Variedad en tallas y colores para que encuentres tu prenda ideal.</p>
    </div>
    <div class="col-md-3 mb-4">
        <i class="bi bi-truck" style="font-size: 60px;"></i>
        <h3 class="fw-bold h5 mt-3">Envíos Rápidos</h5>
            <p>Compra fácil y recibe tu pedido en tiempo récord.</p>
    </div>
</div>
```

- Galería de Ofertas: Muestra tarjetas con productos que tienen descuento.

```blade
<h2 class="mb-5 text-center fw-bold">Ofertas</h2>
<div class="row row-cols-2 row-cols-md-3 row-cols-lg-4 g-4">
    @foreach ($discountedProducts as $product)
        <div class="col">
            <a href="{{ route(name: 'productos.show', parameters: $product->id) }}" class="text-decoration-none text-dark">
                <div class="card h-100 shadow-sm">
                    <img src="{{ asset(path: 'storage/' . optional(value: $product->images->first())->url) }}"
                        class="card-img-top img-fluid" alt="{{ $product->name }}"
                        style="object-fit: cover; height: 180px;">
                    <div class="card-body d-flex flex-column">
                        <p class="h5 card-title">{{ $product->name }}</p>
                        <p class="card-text text-truncate" style="max-height: 4.5rem;">
                            {{ $product->description }}</p>
                        <p class="card-text">Tallas disponibles:
                            {{ $product->sizes->pluck('name')->join(', ') . '.' }}</p>
                        <p class="mb-0">
                            <span
                                class="text-muted text-decoration-line-through">{{ number_format(num: $product->price, decimals: 2) }}€</span>
                            <span class="fw-bold text-danger ms-4 fs-4 ms-2">
                                {{ number_format(num: $product->price * (1 - $product->discount / 100), decimals: 2) }}€
                            </span>
                        </p>
                    </div>
                </div>
            </a>
        </div>
    @endforeach
</div>
```

Cada tarjeta incluye imagen, nombre, descripción truncada, tallas disponibles y precio original tachado con el nuevo precio destacado en rojo.

Además se usa un <a> envolviendo cada tarjeta para que al pulsarla redirija a la vista donde se muestran los detalles del producto.

1.3.4 - ORDERCONFIRMATION.PHP

FUNCIONALIDAD: Permite al usuario confirmar su pedido con los productos previamente seleccionados desde el carrito. Incluye resumen detallado del pedido, cálculo del total (considerando descuentos), selección del método de pago (simulado) y opciones para confirmar o cancelar. También envía un correo de notificación al administrador con los detalles del pedido realizado.

- Variables:
    - **selectedItems**: guarda los ids de los productos seleccionados por el usuario desde el carrito.
    - **cartItems**: guarda las instancias completas de los productos seleccionados.
    - **totalPrice**: guarda el precio total de los productos seleccionados.

```php
public $selectedItems = [];
3 references
public $cartItems;
1 reference
public $totalPrice = 0;
```

- mount(): Carga los ítems seleccionados desde la sesión, obtiene las instancias completas de los ítems del carrito actual del usuario autenticado, y calcula el total de la compra al iniciar el componente.

```php
public function mount(): void
{
    // Recuperamos los IDs de los productos seleccionados desde la sesión
    $this->selectedItems = session(key: 'selectedItems', default: []);

    // Cargamos los productos seleccionados del carrito del usuario autenticado
    $this->cartItems = auth()->user()
        ->cart
        ->items()
        ->whereIn('id', $this->selectedItems)
        ->get();

    // Calculamos el total del pedido
    $this->updateTotal();
}
```

- Método updateTotal(): Recorre los productos seleccionados, calcula el precio total aplicando descuentos si tiene, y multiplica el precio final por la cantidad de unidades seleccionadas.

```php
public function updateTotal(): void
{
    $total = 0; // Inicializamos el total en 0

    foreach ($this->cartItems as $item) {
        // Verificamos si el ítem está entre los seleccionados
        if (in_array(needle: $item->id, haystack: $this->selectedItems)) {

            // Precio base del producto (sin descuento)
            $price = $item->product->price;

            // Si el producto tiene un descuento, se aplica
            if ($item->product->discount && $item->product->discount > 0) {
                $discount = $item->product->discount;
                $price = $price - ($price * $discount / 100); // Precio con descuento aplicado
            }

            // Sumamos al total la cantidad multiplicada por el precio (con o sin descuento)
            $total += $item->quantity * $price;
        }
    }

    // Asignamos el total calculado a la propiedad del componente
    $this->totalPrice = $total;
}
```

- Método confirmOrder(): Crea un nuevo pedido en la base de datos, envía un correo electrónico al administrador (juditquirosviolero@gmail.com) con los detalles del pedido usando la clase EmailPedidos, y por último redirige al usuario al dashboard con un mensaje de éxito.

```php
public function confirmOrder(): mixed|RedirectResponse
{
    $order = Order::createFromCartItems(user: auth()->user(), cartItems: $this->cartItems);

    Mail::to(users: 'juditquirosviolero@gmail.com')->send(mailable: new EmailPedidos(pedido: $order));

    session()->flash(key: 'success', value: '¡Pedido confirmado con éxito!');

    return redirect()->route(route: 'user.index');
}
```

- Método cancelOrder(): Redirige al usuario de vuelta al carrito si decide no completar la compra.

```php
public function cancelOrder(): mixed|RedirectResponse
{
    return redirect()->route(route: 'user.cart');
}
```

- render(): Devuelve la vista order-confirmation con todos los datos necesarios para mostrar el pedido y las opciones.

VISTA:

- Mostrar los productos que hay en el carrito: Se recorre con un foreach los elementos que hay en un carrito, para poder mostrarlos juntos con sus datos individualmente en tarjetas.

```blade
@foreach ($cartItems as $item)
    <div class="col-12 mb-3">
        <div class="card p-3" role="group" aria-label="Producto {{ $item->product->name }}">
```

- Dentro de cada tarjeta se muestra:
    - Imagen del producto (primera imagen asociada).

```blade
<div class="col-md-4 col-12 my-3">
    <img src="{{ asset(path: 'storage/' . $item->product->images->first()->url) }}"
        alt="Imagen de {{ $item->product->name }} - {{ $item->product->category }}, para {{ $item->product->gender }}"
        aria-label="Imagen del producto {{ $item->product->name }}"
        title="Imagen del producto {{ $item->product->name }}"
        class="img-fluid rounded shadow-sm d-block mx-auto" style="max-height: 250px;">
</div>
```

    - Nombre, descripción, color, estilo, categoría y género.

```blade
<div class="col-md-8 col-12">
    <h3 class="h5" title="Nombre del producto">{{ $item->product->name }}</h3>
    <p class="text-muted" title="Descripción del producto">
        {{ $item->product->description }}</p>

<div class="col-6 col-md-4" role="list"
    aria-label="Características básicas del producto">
    <p role="listitem"><strong>Género:</strong>
        <span>{{ $item->product->gender }}</span>
    </p>
    <p role="listitem"><strong>Estilo:</strong>
        <span>{{ $item->product->style }}</span>
    </p>
    <p role="listitem"><strong>Categoría:</strong>
        <span>{{ $item->product->category }}</span>
    </p>
    <p role="listitem"><strong>Color:</strong> <span>{{ $item->product->color }}</span>
    </p>
</div>
```

- Talla seleccionada y cantidad.

```
<div class="col-6 col-md-4" role="list"
    aria-label="Talla, cantidad y precios del producto">
    <p role="listitem">
        <strong>Talla:</strong>
        <span title="Talla seleccionada">
            {{ $item->product->sizes->firstWhere('id', $item->size_id)->name }}
        </span>
    </p>
    <p role="listitem">
        <strong>Cantidad:</strong>
        <span title="Cantidad seleccionada">{{ $item->quantity }}</span>
    </p>
```

- Precio individual con descuento si aplica (tachado el precio original).

```
<p role="listitem">
    <strong>Precio Unidad:</strong>
    @if ($item->product->discount && $item->product->discount > 0)
        <span class="text-decoration-line-through text-secondary me-2"
            title="Precio original">
            {{ number_format(num: $item->product->price, decimals: 2) }} €
        </span>
        <span class="text-danger fw-bold" title="Precio con descuento">
            {{ number_format(num: $item->product->price - ($item->product->price * $item->product->discount) / 100, decimals: 2) }}
            €
        </span>
    @else
        {{ number_format(num: $item->product->price, decimals: 2) }} €
    @endif
</p>
```

- Total individual por producto.

```
<p role="listitem">
    <strong>Total:</strong>
    @if ($item->product->discount && $item->product->discount > 0)
        {{ number_format(num: ($item->product->price - ($item->product->price * $item->product->discount) / 100) * $item->quantity, decimals: 2) }}
        €
    @else
        {{ number_format(num: $item->product->price * $item->quantity, decimals: 2) }} €
    @endif
</p>
```

- Total general del pedido: Calculado y mostrado al final en un <p> destacado.

```
<!-- Total general -->
<div class="text-center my-4" role="contentinfo" aria-label="Total general a pagar">
    <p class="fw-bold fs-4">Total a pagar: {{ number_format(num: $totalPrice, decimals: 2) }} €</p>
</div>
```

- Método de pago: Es un desplegable con tres opciones: Tarjeta de Crédito, PayPal y Transferencia Bancaria. No se almacena ni se usa funcionalmente (por ahora es solo decorativo), aunque se podría implementar en un futuro.

```
    <div class="col-md-6 col-12 mb-3 d-flex align-items-center">
        <label for="paymentMethod" class="w-25 fw-bold" id="paymentMethodLabel">Método de pago:</label>
        <select class="form-control" id="paymentMethod" aria-labelledby="paymentMethodLabel"
            title="Selecciona método de pago">
            <option value="creditCard">Tarjeta de Crédito</option>
            <option value="paypal">PayPal</option>
            <option value="bankTransfer">Transferencia Bancaria</option>
        </select>
    </div>
```

- Botones:
  - Confirmar y pagar: ejecuta confirmOrder()

- Cancelar: ejecuta cancelOrder()

```
<button class="btn btn-success" wire:click="confirmOrder" aria-label="Confirmar y pagar pedido"
    title="Confirmar y pagar pedido">
    <i class="bi bi-credit-card" aria-hidden="true"></i> Confirmar y pagar
</button>
<button class="btn btn-danger" wire:click="cancelOrder" aria-label="Cancelar pedido"
    title="Cancelar pedido">
    <i class="bi bi-x-square" aria-hidden="true"></i> Cancelar
</button>
```

1.3.5 - ORDERCRUD.PHP

FUNCIONALIDAD: Gestión de pedidos en el panel administrativo del ecommerce: permite visualizar, filtrar, actualizar, eliminar y reembolsar pedidos, así como eliminar productos de un pedido.

- Variables:
  - **orders**: Colección de pedidos cargados.
  - **totalreturn**: Array para llevar la cuenta de totales devueltos por pedido.
  - **message**: Mensaje que se mostrará al usuario-
  - **messageType**: Tipo de mensaje (success, warning, error).
  - **showMessage**: Controla si se muestra el mensaje o no.
  - **statusFilter**: Filtro para el estado del pedido.
  - **userName**: Filtro para buscar pedidos por nombre de usuario.
  - **startDate**: Fecha inicial para filtrar pedidos por rango de fechas.
  - **endDate**: Fecha final para filtrar pedidos por rango de fechas.

```
public $orders;
5 references
public $totalreturn = [];

1 reference
public $message = null;
1 reference
public $messageType = 'success';
1 reference
public $showMessage = false;

3 references
public $statusFilter = '';
3 references
public $userName = '';
3 references
public $startDate = '';
3 references
public $endDate = '';
```

- mount(): Carga los pedidos al inicializar el componente. Muestra un mensaje si no hay pedidos.

```php
public function mount(): void
{
    $this->loadOrders();

    if ($this->orders->isEmpty()) {
        $this->setMessage(message: 'No hay ningún Pedido', messageType: 'warning');
    }
}
```

- loadOrders(): Consulta todos los pedidos con sus relaciones necesarias.

```php
public function loadOrders(): void
{
    $this->orders = Order::with(relations: ['user', 'items.product', 'items.size'])->get();
}
```

- filter(): Aplica filtros a la consulta de pedidos por estado (pendiente, pagado, enviado, cancelado, reembolsado), usuario o por rango de fechas. Si no se encuentran resultados con los filtros aplicados, se muestra un mensaje tipo flash.

```php
public function filter(): void
{
    $query = Order::with(relations: ['user', 'items.product', 'items.size']);

    if (!empty($this->statusFilter)) {
        $query->where(column: 'status', operator: $this->statusFilter);
    }

    if (!empty($this->userName)) {
        $query->whereHas(relation: 'user', callback: function (Builder<TRelatedModel> $q)… {
            // Busca usuarios cuyo nombre contenga el texto introducido (búsqueda parcial)
            $q->where(column: 'name', operator: 'like', value: '%' . $this->userName . '%');
        });
    }

    if (!empty($this->startDate) && !empty($this->endDate)) {
        // Filtro por rango de fechas creado_at, incluyendo ambos extremos
        $query->whereBetween(column: 'created_at', values: [$this->startDate, $this->endDate]);
    }

    // Ejecuta la consulta con todos los filtros aplicados
    $this->orders = $query->get();

    // Si no se encontraron pedidos con los filtros, muestra un mensaje de error temporal
    if ($this->orders->isEmpty()) {
        session()->flash(key: 'error', value: 'No se encontraron pedidos con los filtros aplicados');
    }
}
```

- resetFilters(): Limpia los filtros y recarga todos los pedidos.

```php
public function resetFilters()
{
    $this->statusFilter = '';
    $this->userName = '';
    $this->startDate = '';
    $this->endDate = '';
    $this->loadOrders();
}
```

- updateStatus(): Actualiza el estado de un pedido específico.

```php
public function updateStatus($orderId, $status): void
{
    $order = Order::find(id: $orderId);
    if ($order) {
        $order->status = $status;
        $order->save();
    }
}
```

- removeItem(): Su función principal es eliminar un producto de un pedido, actualizar el total de la compra, y si el estado del pedido es pagado o reembolsado devuelve el stock. Además si tras la eliminación no quedan productos ni importe pendiente de devolución, se borra el pedido por completo.

```php
public function removeItem($itemId): void
{
    $item = OrderItem::find(id: $itemId);

    if ($item) {
        $order = $item->order;

        // Si el pedido está pagado o reembolsado, devuelve el stock al inventario
        if ($order->status === 'pagado' || $order->status === 'reembolsado') {
            \DB::table(table: 'product_size')
                ->where(column: 'product_id', operator: $item->product_id)
                ->where(column: 'size_id', operator: $item->size_id)
                ->increment(column: 'stock', amount: $item->quantity); // Suma la cantidad devuelta al stock


            // Inicializa el total de devolución para ese pedido si no existe
            if (!isset($this->totalreturn[$order->id])) {
                $this->totalreturn[$order->id] = 0;
            }

            // Acumula el importe que se debe devolver para ese pedido
            $this->totalreturn[$order->id] += $item->price * $item->quantity;
        }

        // Elimina el ítem del pedido
        $item->delete();

        // Recalcula y actualiza el total del pedido basado en los ítems restantes
        $order->total = $order->items()->sum(\DB::raw(value: 'price * quantity'));
        $order->save();

        // Si no quedan ítems en el pedido y no hay devolución pendiente, elimina el pedido
        if ($order->items()->count() === 0 && (!isset($this->totalreturn[$order->id]) || $this->totalreturn[$order->id] == 0)) {
            $order->delete();
            $this->setMessage(message: 'Pedido #' . $order->id . ' eliminado porque no tiene productos.', messageType: 'warning');
        }
    }

    // Refresca la lista de pedidos para reflejar los cambios
    $this->loadOrders();
}
```

- returned(): Marca el pedido como reembolsado, y elimina el pedido si no quedan productos. Además se muestra un mensaje con el resultado de la operación.

```php
public function returned($orderId): void
{
    $order = Order::find(id: $orderId);

    if ($order) {
        $order->status = 'reembolsado';
        $order->save();

        // Si no quedan productos, elimina el pedido
        if ($order->items()->count() === 0) {
            $order->delete();
            $this->setMessage(message: 'Pedido #' . $order->id . ' eliminado porque no tiene productos.', messageType: 'warning');
        } else {
            // Si aún tiene productos, solo notifica que se procesó la devolución
            $this->setMessage(message: 'Se ha procesado la devolución del dinero del Pedido #' . $order->id . '.', messageType: 'success');
        }
    }

    // Vuelve a cargar el componente para actualizar la vista
    $this->mount();
}
```

- setMessage(): Configura un mensaje a mostrar en la vista.

```php
public function setMessage($message, $messageType = 'success'): void
{
    $this->message = $message;
    $this->messageType = $messageType;
    $this->showMessage = true;
}
```

- render(): Devuelve la vista asociada al componente.

```php
public function render(): View
{
    return view(view: 'livewire.order-crud');
}
```

VISTA:

- Formulario para filtrar:
    - Campos de búsqueda: Permite filtrar los pedidos por estado del pedido, nombre del usuario (parcial o completo), y fecha de creación.

```
<form wire:submit.prevent="filter" aria-label="Formulario de filtro de pedidos">
    <div class="row g-3 align-items-end">
        <div class="col-md-3">
            <label for="statusFilter" class="form-label">Estado</label>
            <select id="statusFilter" name="statusFilter" class="form-select" wire:model.defer="statusFilter" title="Filtrar por estado" aria-label="Filtrar por estado">
                <option value="">-- Todos --</option>
                <option value="pendiente">Pendiente</option>
                <option value="pagado">Pagado</option>
                <option value="enviado">Enviado</option>
                <option value="cancelado">Cancelado</option>
                <option value="reembolsado">Reembolsado</option>
            </select>
        </div>
        <div class="col-md-3">
            <label for="userName" class="form-label">Nombre de Usuario</label>
            <input type="text" id="userName" name="userName" class="form-control" wire:model.defer="userName"
                placeholder="Ej: Juan Pérez" title="Filtrar por nombre de usuario" aria-label="Nombre de Usuario">
        </div>
        <div class="col-md-3">
            <label for="startDate" class="form-label">Desde</label>
            <input type="date" id="startDate" name="startDate" class="form-control" wire:model.defer="startDate"
                title="Fecha de inicio" aria-label="Fecha de inicio">
        </div>
        <div class="col-md-3">
            <label for="endDate" class="form-label">Hasta</label>
            <input type="date" id="endDate" name="endDate" class="form-control" wire:model.defer="endDate"
                min="{{ $startDate }}" title="Fecha final" aria-label="Fecha final">
        </div>
```

Usa wire:model.defer para evitar actualizaciones en tiempo real y solo enviar los datos al hacer submit.

- Botones: Tiene 2 botones uno para filtrar, que ejecuta el método filter() del componente. Y otro para borrar filtros, que llama a resetFilters() para limpiar todos los campos y volver a mostrar todos los pedidos.

```
        <div class="col-12 text-end mt-3">
            <button type="submit" class="btn btn-primary me-2" aria-label="Filtrar pedidos">
                <i class="bi bi-search"></i> Filtrar
            </button>
            <button type="button" class="btn btn-secondary" wire:click="resetFilters" aria-label="Borrar filtros">
                <i class="bi bi-x-circle"></i> Borrar Filtros
            </button>
        </div>
    </div>
</form>
```

- Mensajes de estado: Si en el componente se establece un mensaje con setMessage(), se muestra como alerta Bootstrap con la clase correspondiente (success, warning, etc.).

```
@if ($showMessage)
    <div class="alert alert-{{ $messageType === 'success' ? 'success' : ($messageType === 'warning' ? 'warning' : 'danger') }} mb-4" role="alert">
        {{ $message }}
    </div>
@endif
```

- Listado de Pedidos: Cada pedido se muestra como una tarjeta/tabla y cada tarjeta contiene:
  - Cabecera con el id del pedido, el nombre del usuario que ha hecho el pedido, precio total del pedido, su estado.

```
@foreach ($orders as $order)
    <div class="card mb-4 shadow-sm">
        <div class="card-header d-flex flex-wrap justify-content-between align-items-center bg-light">
            <div>
                <strong>Pedido #{{ $order->id }}</strong> - {{ $order->user->name }}
                <span class="ms-3">Total: <strong>€{{ number_format(num: $order->total, decimals: 2) }}</strong></span>
            </div>
            <div>
                <label for="status-{{ $order->id }}" class="me-2 fw-bold">Estado:</label>
                <select id="status-{{ $order->id }}" name="status-{{ $order->id }}" class="form-select form-select-sm d-inline-block w-auto"
                    wire:change="updateStatus({{ $order->id }}, $event.target.value)"
                    title="Cambiar estado del pedido #{{ $order->id }}" aria-label="Estado del pedido {{ $order->id }}">
                    <option value="pendiente" {{ $order->status === 'pendiente' ? 'selected' : '' }}>Pendiente</option>
                    <option value="pagado" {{ $order->status === 'pagado' ? 'selected' : '' }}>Pagado</option>
                    <option value="enviado" {{ $order->status === 'enviado' ? 'selected' : '' }}>Enviado</option>
                    <option value="cancelado" {{ $order->status === 'cancelado' ? 'selected' : '' }}>Cancelado</option>
                    <option value="reembolsado" {{ $order->status === 'reembolsado' ? 'selected' : '' }}>Reembolsado</option>
                </select>
            </div>
        </div>
```

- Cuerpo donde se muestran los productos que contiene el pedido, junto con la talla seleccionada, la cantidad y su precio. Además cada producto de un pedido tiene un botón de eliminarlo del pedido.

```
<div class="card-body">
    @foreach ($order->items as $item)
        <div class="row align-items-center mb-3 pb-3 border-bottom">
            <div class="col-md-3">
                <h6 class="mb-0" title="Nombre del producto">{{ $item->product->name }}</h6>
            </div>
            <div class="col-md-2">
                <span class="text-muted">Talla:</span> {{ $item->size->name }}
            </div>
            <div class="col-md-2">
                <span class="text-muted">Cantidad:</span> {{ $item->quantity }}
            </div>
            <div class="col-md-2">
                <span class="text-muted">Precio:</span> €{{ number_format(num: $item->price, decimals: 2) }}
            </div>
            <div class="col-md-3 text-end">
                <button class="btn btn-outline-danger btn-sm" wire:click="removeItem({{ $item->id }})"
                    title="Eliminar producto del pedido" aria-label="Eliminar producto {{ $item->product->name }}">
                    <i class="bi bi-trash3"></i> Eliminar Producto
                </button>
            </div>
        </div>
    @endforeach
```

- Además si el estado del pedido es pagado o reembolsado, al eliminar un producto del pedido (como si alguien realizara una devolución) se muestra en la vista un botón para devolver el dinero del producto eliminado, y el dinero a devolver.

```
@if (isset($totalreturn[$order->id]))
    <div class="mt-2 d-flex justify-content-between">
        <span><strong>Total a devolver:</strong> €{{ number_format(num: $totalreturn[$order->id], decimals: 2) }}</span>
        <button class="btn btn-warning btn-sm" wire:click="returned({{ $order->id }})"
            title="Devolver dinero del pedido" aria-label="Devolver dinero del pedido {{ $order->id }}">
            <i class="bi bi-coin"></i> Devolver dinero
        </button>
    </div>
@endif
```

1.3.6 - PRODUCTCRUD.PHP

FUNCIONALIDAD: Este componente proporciona un CRUD completo para la gestión de productos en el panel de administración de un ecommerce. Permite crear, editar, listar y eliminar productos, así como gestionar tallas, stocks y múltiples imágenes por producto.

- Variables:
  - **products**: colección de productos cargados desde la base de datos.
  - **product_id, name, description, color, gender, style, category, price, discount:** campos principales del formulario.
  - **images**: imágenes nuevas subidas desde el formulario.
  - **imagesDB**: imágenes ya guardadas en la base de datos asociadas al producto.
  - **newCategory**, newSize: inputs para crear nuevas categorías y tallas.
  - **categories**: array con todas las categorías únicas disponibles.
  - **sizes**: array asociativo [size_id => stock] para el stock por talla.
  - **availableSizes**: tallas filtradas por tipo de producto (ropa/calzado).
  - **allSizes**: todas las tallas disponibles en la tabla sizes.
  - **view**: controla la vista actual (list o form).

```php
public $products, $product_id, $name,
$description, $color, $gender, $style,
$category, $price, $discount;

10 references
public $images = [];
4 references
public $imagesDB = [];

7 references
public $newCategory = '';
7 references
public $newSize = '';
4 references
public $categories = [];
7 references
public $sizes = [];
6 references
public $availableSizes;
3 references
public $allSizes;

5 references
public $view = 'list';
```

- mount(): Inicializa productos, tallas y categorías desde la base de datos. También aplica el filtrado inicial de tallas.

```php
public function mount(): void
{
    $this->products = Product::all(); // Carga todos los productos
    $this->allSizes = Size::all(); // Carga todas las tallas posibles

    $this->filterSizes(); // Filtra tallas según categoría actual (inicial)

    // Obtiene todas las categorías únicas existentes en productos para el filtro o selector
    $this->categories = Product::distinct()->pluck(column: 'category')->toArray();

    $this->availableSizes = Size::all();
}
```

- filter(): Permite buscar productos por ID o nombre. Si no se encuentra coincidencia, muestra todos los productos con un mensaje de error.

```php
public function filter(): void
{
    if (empty($this->product_id) && empty($this->name)) {
        session()->flash(key: 'error', value: 'Introduce algún dato de búsqueda');
        $this->products = Product::all();
        return;
    }

    $query = Product::query();

    // Filtra por ID exacto si se proporciona
    if (!empty($this->product_id)) {
        $query->where(column: 'id', operator: $this->product_id);
    }

    // Filtra por nombre con búsqueda parcial usando LIKE
    if (!empty($this->name)) {
        $query->where(column: 'name', operator: 'like', value: '%' . $this->name . '%');
    }

    $this->products = $query->get();

    // Si no hay resultados, muestra mensaje y recarga todos los productos
    if ($this->products->isEmpty()) {
        session()->flash(key: 'error', value: 'Producto no encontrado');
        $this->products = Product::all();
    }
}
```

- resetFilters(): Limpia los filtros aplicados y recarga todos los productos.

```php
public function resetFilters(): void
{
    $this->name = '';
    $this->product_id = '';
    $this->products = Product::all();
}
```

- resetInputs(): Limpia todos los campos del formulario, incluyendo imágenes y tallas.

```php
public function resetInputs(): void
{
    $this->product_id = $this->name = $this->description = $this->color = $this->gender = $this->style = $this->category = $this->newSize = $this->newCategory = '';
    $this->price = null;
    $this->sizes = [];
    $this->images = [];      // Limpia imágenes cargadas nuevas
    $this->imagesDB = [];    // Limpia imágenes guardadas en BD
}
```

- showCreateForm(): Activa la vista de formulario vacío para crear un nuevo producto.

```php
public function showCreateForm(): void
{
    $this->resetInputs();
    $this->view = 'form';
}
```

- filterSizes(): Filtra tallas disponibles según la categoría (calzado = numéricas, ropa = XS-XL).

```php
public function filterSizes(): void
{
    if (strtolower(string: $this->category) === 'calzado') {
        // Para calzado, filtra solo tallas numéricas
        $this->availableSizes = $this->allSizes->filter(function ($size): bool {
            return is_numeric(value: $size->name);
        });
    } else {
        // Para ropa, filtra tallas estándar XS, S, M, L, XL
        $this->availableSizes = $this->allSizes->filter(function ($size): bool {
            return in_array(needle: strtolower(string: $size->name), haystack: ['xs', 's', 'm', 'l', 'xl']);
        });
    }
}
```

- addNewCategory(): Permite agregar una nueva categoría al listado si no existe. Se limpia el campo después de guardarla.

```php
public function addNewCategory(): void
{
    // Limpia espacios y convierte el texto a formato "Título" (primera letra mayúscula)
    $name = trim(string: mb_convert_case(string: $this->newCategory, mode: MB_CASE_TITLE));

    // Si el nombre no está vacío y no existe ya en el array de categorías
    if ($name && !in_array(needle: $name, haystack: $this->categories)) {
        // Añade la nueva categoría al listado de categorías
        $this->categories[] = $name;

        // Asigna la nueva categoría seleccionada para usar en el formulario
        $this->category = $name;

        // Muestra un mensaje flash para informar al usuario que la categoría fue añadida
        session()->flash(key: 'message', value: 'Nueva categoría añadida.');
    }

    // Limpia el campo input de nueva categoría para que quede vacío
    $this->newCategory = '';
}
```

- addNewSize(): Permite crear una talla si no existe en la base de datos. Se añade automáticamente al array sizes con stock 0.

```php
public function addNewSize(): void
{
    // Limpia espacios y convierte el texto a MAYÚSCULAS para uniformidad
    $name = trim(string: mb_strtoupper(string: $this->newSize));

    // Si el nombre no está vacío
    if ($name) {
        // Busca o crea una nueva talla en la tabla sizes con ese nombre
        $size = Size::firstOrCreate(attributes: ['name' => $name]);

        // Comprueba que la talla no esté ya en el array de tallas disponibles
        if (!collect(value: $this->availableSizes)->contains(key: 'id', operator: $size->id)) {
            // Si no está, la añade a la lista de tallas disponibles
            $this->availableSizes[] = $size;
        }

        // Inicializa el stock de esta talla nueva a 0 en el array sizes (para el formulario)
        $this->sizes[$size->id] = 0;
    }

    // Limpia el campo input de nueva talla para que quede vacío
    $this->newSize = '';
}
```

- removePreviewImage($key): Elimina una imagen cargada en tiempo real del array images, antes de ser subida.

```php
public function removePreviewImage($key): void
{
    if (isset($this->images[$key]) && is_object(value: $this->images[$key])) {
        unset($this->images[$key]);
        // Reindexar el array para evitar huecos y mantenerlo limpio
        $this->images = array_values(array: $this->images);
    }
}
```

- deleteImage($id): Elimina una imagen existente tanto de la base de datos como del disco, y actualiza imagesDB.

```php
public function deleteImage($id): void
{
    $image = Image::find(id: $id);

    if ($image) {
        Storage::disk(name: 'public')->delete(paths: $image->url);
        $image->delete();

        $this->imagesDB = Image::where(column: 'product_id', operator: $this->product_id)->get();
    }
}
```

- saveImages($productId): Este es un método privado que se llama en store y en update. Lo que hace es validar y guardar todas las imágenes nuevas asociadas al

producto, dentro de storage/app/public/products. Además crea un registro en la tabla images para cada imagen.

```php
private function saveImages($productId): void
{

    $validImages = collect(value: $this->images)
        ->filter(callback: fn(TValue $img): bool => $img instanceof \Illuminate\Http\UploadedFile)
        ->values();

    if ($validImages->isEmpty()) {
        return;
    }

    $this->validate(rules: [
        'images.*' => 'image|max:2048',
    ]);

    foreach ($this->images as $image) {
        $path = $image->store('products', 'public');
        Image::create(attributes: [
            'url' => $path,
            'product_id' => $productId,
        ]);
    }

    $this->images = [];
    $this->dispatch(event: 'clearFileInput');

}
```

- edit($id): Carga los datos del producto seleccionado, incluyendo imágenes y tallas con su stock, para edición. Cambia la vista a form.

```php
public function edit($id): void
{

    $product = Product::with(relations: 'images')->findOrFail(id: $id);
    $this->product_id = $product->id;
    $this->fill(values: $product->toArray());

    $this->imagesDB = $product->images;

    $this->sizes = $product->sizes->pluck('pivot.stock', 'id')->toArray();

    $this->availableSizes = Size::all();
    $this->categories = Product::distinct()->pluck(column: 'category')->toArray();
    $this->view = 'form';
}
```

- store(): Valida y guarda un nuevo producto. Si se han ingresado nuevas categorías o tallas, también se guardan. Asocia las tallas con sus respectivos stocks mediante la tabla pivote product_size. Guarda imágenes y reinicia el formulario tras guardar.

```php
public function store(): void
{
    $this->validate(rules: [
        'name' => 'required',
        'price' => 'required|numeric',
        'discount' => 'nullable|numeric|min:0', // nullable para permitir vacío
        'sizes' => [
            'required',
            function ($attribute, $value, $fail): void {
                $hasStock = false;
                foreach ($value as $stock) {
                    if ($stock > 0) {
                        $hasStock = true;
                        break;
                    }
                }
                if (!$hasStock) {
                    $fail('Debes asignar stock a al menos una talla.');
                }
            }
        ],
    ]);

    // Si se ingresó una nueva categoría, guárdala
    if (!empty($this->newCategory)) {
        $this->category = trim(string: $this->newCategory);
    }

    // Si se ingresó una nueva talla, guárdala
    if (!empty($this->newSize)) {
        $size = Size::firstOrCreate(attributes: ['name' => trim(string: $this->newSize)]);
        $this->sizes[$size->id] = 0; // inicializa stock en 0 para esa talla
    }

    $product = Product::create(attributes: $this->validateData());

    // Guardar tallas con su stock
    foreach ($this->sizes as $sizeId => $stock) {
        if ($stock > 0) {
            $product->sizes()->attach(ids: $sizeId, attributes: ['stock' => $stock]);
        }
    }

    // Guardar imágenes
    $this->saveImages(productId: $product->id);

    // Recargar imágenes de BD (opcional si vas a mostrar inmediatamente)
    $this->imagesDB = Image::where(column: 'product_id', operator: $product->id)->get();

    $this->resetInputs();
    session()->flash(key: 'message', value: 'Producto creado correctamente.');
    $this->view = 'list';
}
```

- update(): Valida y actualiza un producto existente. Si se añaden nuevas categorías o tallas, también se guardan. Sincroniza las tallas con su stock mediante la tabla pivote product_size. Guarda nuevas imágenes si las hay, reinicia el formulario y vuelve a la vista de listado.

```php
public function update(): void
{
    $this->validate(rules: [
        'name' => 'required',
        'price' => 'required|numeric',
        'discount' => 'nullable|numeric|min:0', // nullable para permitir vacio
        'sizes' => [
            'required',
            function ($attribute, $value, $fail): void {
                $hasStock = false;
                foreach ($value as $stock) {
                    if ($stock > 0) {
                        $hasStock = true;
                        break;
                    }
                }
                if (!$hasStock) {
                    $fail('Malamente');
                }
            }
        ],
    ]);

    if (!empty($this->newCategory)) {
        $this->category = trim(string: $this->newCategory);
    }

    if (!empty($this->newSize)) {
        $size = Size::firstOrCreate(attributes: ['name' => trim(string: $this->newSize)]);
        $this->sizes[$size->id] = 0;
    }

    $product = Product::findOrFail(id: $this->product_id);
    $product->update($this->validateData());

    $syncData = [];
    foreach ($this->sizes as $sizeId => $stock) {
        if ($stock > 0) {
            $syncData[$sizeId] = ['stock' => $stock];
        }
    }

    $product->sizes()->sync($syncData);

    // Guardar imágenes asociadas SOLO si hay imágenes nuevas
    if (!empty($this->images)) {
        $this->saveImages(productId: $product->id);
    }

    $this->resetInputs();
    session()->flash(key: 'message', value: 'Producto actualizado.');
    $this->view = 'list';
}
```

- cancel(): Cancela la creación o edición de un producto. Limpia todos los campos del formulario y vuelve a la vista de listado.

```php
public function cancel(): void
{
    $this->resetInputs();
    $this->view = 'list';
}
```

- delete($id): Elimina un producto existente de la base de datos según su ID. Muestra un mensaje de confirmación tras la eliminación.

```php
public function delete($id): void
{
    Product::findOrFail(id: $id)->delete();
    session()->flash(key: 'message', value: 'Producto eliminado.');
}
```

- validateData(): Devuelve los datos del formulario ya validados y formateados (como nombre, color, estilo y género en formato título) para ser usados en la creación o actualización del producto.

```php
private function validateData(): array
{
    return [
        'name' => trim(string: mb_convert_case(string: $this->name, mode: MB_CASE_TITLE)),
        'description' => $this->description,
        'color' => trim(string: mb_convert_case(string: $this->color, mode: MB_CASE_TITLE)),
        'gender' => trim(string: mb_convert_case(string: $this->gender, mode: MB_CASE_TITLE)),
        'style' => trim(string: mb_convert_case(string: $this->style, mode: MB_CASE_TITLE)),
        'category' => $this->category,
        'price' => $this->price,
        'discount' => $this->discount,
    ];

}
```

- render(): Devuelve la vista Livewire product-crud.

```php
public function render(): View
{
    return view(view: 'livewire.product-crud');
}
```

VISTA: Se divide en dos secciones principales, controladas por la propiedad $view: list que es la vista de listado de productos con filtrado y acciones (editar, eliminar) y form que es la vista de formulario para crear o actualizar productos, incluyendo la gestión de imágenes

Listado de Productos ($view === 'list'):

- Filtro de Productos: Filtra por nombre o ID usando inputs con wire:model=”name” y wire.mode=”product_id”.

- Mensajes Flash: Muestra mensajes de éxito o error después de acciones (crear, editar, eliminar).

```blade
@if (session()->has(key: 'message'))
    <div class="alert alert-success mb-4" role="alert">
        {{ session(key: 'message') }}
    </div>
@endif
@if (session()->has(key: 'error'))
    <div class="alert alert-danger mb-4" role="alert">
        {{ session(key: 'error') }}
    </div>
@endif
```

- Botón para crear un producto nuevo, y supuesta paginación: La paginación no está implementada por falta de tiempo, pero se podría implementar a futuro para mejorar la vista de los productos.

```
<button wire:click="showCreateForm" class="btn btn-success" aria-label="Crear nuevo producto">
    Crear nuevo producto
</button>
```

```
<!-- NO funciona (No está implementado es de decoración 😉 )-->
<div>
    <label for="show-count" class="visually-hidden">Número de productos a mostrar</label>
    <select id="show-count" class="form-select form-select-sm"
        aria-label="Seleccionar número de productos a mostrar">
        <option value="5">Mostrar 5 primeros productos</option>
        <option value="10">Mostrar 10 primeros productos</option>
        <option value="20">Mostrar 20 primeros productos</option>
        <option value="50">Mostrar 50 primeros productos</option>
    </select>
</div>
```

- Tabla de Productos: Muestra los productos con todas sus atributos, y además dos botones para editar y eliminar. Si no hay productos se muestra un mensaje dentro de la tabla indicando que "No hay Productos".

```
<table class="table table-bordered table-striped w-100 align-items-center" role="table"
    aria-describedby="productos-heading">
    <thead class="thead-light">
        <tr>
            <th scope="col">ID</th>
            <th scope="col">Nombre</th>
            <th scope="col">Descripción</th>
            <th scope="col">Características</th>
            <th scope="col">Tallas</th>
            <th scope="col">Stock</th>
            <th scope="col">Rebaja</th>
            <th scope="col">Precio</th>
            <th scope="col">Acciones</th>
        </tr>
    </thead>
```

```
    <tbody>
        @forelse ($products as $product)
            <tr>
                <td>{{ $product->id }}</td>
                <td>{{ $product->name }}</td>
                <td>{{ $product->description }}</td>
                <td>
                    <p>Categoría: {{ $product->category }}</p>
                    <p>Color: {{ $product->color }}</p>
                    <p>Género: {{ $product->gender }}</p>
                    <p>Estilo: {{ $product->style }}</p>
                </td>
                <td>
                    @foreach ($product->sizes as $size)
                        <div>{{ $size->name }}</div>
                    @endforeach
                </td>
                <td>
                    @foreach ($product->sizes as $size)
                        <div>{{ $size->pivot->stock }}</div>
                    @endforeach
                </td>
```

```
    <tbody>
        @forelse ($products as $product)
            <tr>
                <td>{{ $product->id }}</td>
                <td>{{ $product->name }}</td>
                <td>{{ $product->description }}</td>
                <td>
                    <p>Categoría: {{ $product->category }}</p>
                    <p>Color: {{ $product->color }}</p>
                    <p>Género: {{ $product->gender }}</p>
                    <p>Estilo: {{ $product->style }}</p>
                </td>
                <td>
                    @foreach ($product->sizes as $size)
                        <div>{{ $size->name }}</div>
                    @endforeach
                </td>
                <td>
                    @foreach ($product->sizes as $size)
                        <div>{{ $size->pivot->stock }}</div>
                    @endforeach
                </td>
                @if ($product->discount <= 0)
                    <td>No tiene rebaja</td>
                @else
                    <td>{{ $product->discount }}%</td>
                @endif
                <td>
                    @if ($product->discount > 0)
                        <span style="text-decoration: line-through; "
                            aria-label="Precio original">
                            {{ number_format(num: $product->price, decimals: 2) }}€
                        </span>
                        <span class="fw-bold text-danger" aria-label="Precio con descuento">
                            {{ number_format(num: $product->price - ($product->price * $product->discount) / 100, decimals: 2) }}€
                        </span>
                    @else
                        {{ number_format(num: $product->price, decimals: 2) }}€
                    @endif
                </td>
```

Al pulsar el botón de eliminar, debería salir un alert preguntando si está seguro de realizar esa acción puesto que no se puede revertir.

```
                <td>
                    <button wire:click="edit({{ $product->id }})" class="btn btn-warning btn-sm mt-2"
                        aria-label="Editar producto {{ $product->name }}" title="Editar producto">
                        <i class="bi bi-pencil" aria-hidden="true"></i>
                    </button>
                    <button wire:click="delete({{ $product->id }})" class="btn btn-danger btn-sm mt-2"
                        onclick="confirm('¿Estás seguro de eliminar este producto?') || event.stopImmediatePropagation()"
                        aria-label="Eliminar producto {{ $product->name }}" title="Eliminar producto">
                        <i class="bi bi-trash" aria-hidden="true"></i>
                    </button>
                </td>
            </tr>
        @empty
            <tr>
                <td colspan="9" class="text-center">No hay Productos</td>
            </tr>
        @endforelse
    </tbody>
</table>
```

Formulario de Creación/Edición ($view !== 'list'):

Encabezado dinámico: Dependiendo de qué botón se pulse, el de editar o crear, el encabezado tendrá escrito Actualizar o Crear.

```
<h2 class="mb-5 my-4 text-center" role="heading" aria-level="2">
    {{ $product_id ? 'Actualizar' : 'Crear' }} Productos
</h2>
```

Nota: Esta comprobación también se hace en otros elementos como el botón para guardar el producto o actualizarlo, o para poner algún campo obligatorio o no (ej: el de las img)

- Gestión de Imágenes: Las imágenes guardadas en la base de datos se muestran en cuadrícula con botón para ver en grande (usa un modal), y para eliminarlas.

```
<label for="images" class="form-label">Imágenes</label>
{{-- Imágenes guardadas en BD --}}
@if (count(value: $imagesDB) > 0)
    <div class="col-md-6 d-flex flex-wrap gap-3 mb-3" role="list"
        aria-label="Imágenes guardadas">
        @foreach ($imagesDB as $index => $image)
            <div style="position: relative; width: 150px; height: 150px;" role="listitem">
                <img src="{{ asset(path: 'storage/' . $image->url) }}"
                    alt="Imagen del producto {{ $index + 1 }}" class="rounded w-100 h-100"
                    style="object-fit: cover;">

                <button type="button"
                    class="btn btn-sm btn-dark position-absolute top-0 end-0 m-1"
                    style="background-color: rgba(0, 0, 0, 0.5); border: none;"
                    data-bs-toggle="modal" data-bs-target="#imageModal{{ $index }}"
                    title="Ver imagen en tamaño grande"
                    aria-label="Ver imagen {{ $index + 1 }} en tamaño grande">
                    <i class="bi bi-fullscreen text-light"></i>
                </button>

                <button type="button"
                    class="btn btn-sm btn-danger position-absolute bottom-0 end-0 m-1"
                    style="background-color: rgba(0, 0, 0, 0.5); border: none;"
                    wire:click="deleteImage({{ $image->id }})" title="Eliminar imagen"
                    aria-label="Eliminar imagen {{ $index + 1 }}">
                    <i class="bi bi-trash text-light"></i>
                </button>
            </div>

            {{-- Modal de imagen grande --}}
            <div class="modal fade" id="imageModal{{ $index }}" tabindex="-1"
                aria-labelledby="imageModalLabel{{ $index }}" aria-hidden="true">
                <div class="modal-dialog modal-dialog-centered modal-lg">
                    <div class="modal-content">
                        <div class="modal-body p-0">
                            <img src="{{ asset(path: 'storage/' . $image->url) }}"
                                alt="Imagen ampliada del producto {{ $index + 1 }}"
                                class="img-fluid w-100">
                        </div>
                    </div>
                </div>
            </div>
        @endforeach
    </div>
@endif
```

- Input de nuevas imágenes: Se pueden subir más de una imagen siempre y cuando al elegir las imágenes en el explorador se seleccionen más de una. Cuando se crea un nuevo producto el campo es obligatorio y tiene que haber al menos 1 imagen subida, pero al editar deja de ser obligatorio.

```
<input type="file" wire:model="images" id="images" class="form-control form-control-sm"
    multiple {{ !$product_id ? 'required' : '' }} aria-describedby="imagesHelp"
    aria-label="Subir nuevas imágenes del producto">

<small id="imagesHelp" class="form-text ">Puedes subir una o más imágenes {{ !$product_id ? '. Campo obligatorio' : '' }}</small>
```

- Previsualización de imágenes seleccionadas: Usa temporaryUrl() para mostrar previews, y tienen un botón para eliminarlas que llama al método del componente “removePreviewImage({{ $key }})”.

```
@if (count(value: $images) > 0)
    <div class="d-flex flex-wrap gap-3 mt-3" role="list"
        aria-label="Vista previa de nuevas imágenes seleccionadas">
        @foreach ($images as $key => $tempImage)
            @if (is_object(value: $tempImage) && method_exists(object_or_class: $tempImage, method: 'temporaryUrl'))
                <div style="position: relative; width: 150px; height: 150px;" role="listitem">
                    <img src="{{ $tempImage->temporaryUrl() }}"
                        alt="Vista previa de imagen {{ $key + 1 }}"
                        class="rounded w-100 h-100" style="object-fit: cover;">
                    <button type="button"
                        class="btn btn-sm btn-danger position-absolute bottom-0 end-0 m-1"
                        wire:click="removePreviewImage({{ $key }})"
                        title="Quitar imagen"
                        aria-label="Quitar imagen previa {{ $key + 1 }}">
                        <i class="bi bi-trash text-light"></i>
                    </button>
                </div>
            @endif
        @endforeach
    </div>
@endif
```

- Otros Campos del Formulario: Hay campos para añadir todos los atributos de producto:
    - Nombre: Campo obligatorio para el nombre del producto.

```
<div class="col-md-6">
    <label for="name" class="form-label">Nombre</label>
    <input type="text" id="name" wire:model="name" class="form-control form-control-sm"
        placeholder="Nombre" required aria-required="true" aria-describedby="nameHelp">
    <small id="nameHelp" class="form-text ">Nombre del producto. Campo obligatorio</small>
</div>
```

    - Descripción: Breve resumen del producto, también obligatorio.

```
<div class="col-md-6">
    <label for="description" class="form-label">Descripción</label>
    <input type="text" id="description" wire:model="description"
        class="form-control form-control-sm" placeholder="Descripción" required aria-required="true"
        aria-describedby="descriptionHelp">
    <small id="descriptionHelp" class="form-text ">Descripción breve del producto. Campo obligatorio</small>
</div>
```

- Color: Campo obligatorio que indica el color principal del producto.

```
<div class="col-md-6">
    <label for="color" class="form-label">Color</label>
    <input type="text" id="color" wire:model="color" class="form-control form-control-sm"
        placeholder="Color" required aria-required="true" aria-describedby="colorHelp">
    <small id="colorHelp" class="form-text ">Color principal del producto. Campo obligatorio</small>
</div>
```

- Género: Es un campo obligatorio, donde se indica el género objetivo del producto (por ejemplo, hombre, mujer, unisex).

```
<div class="col-md-6">
    <label for="gender" class="form-label">Género</label>
    <input type="text" id="gender" wire:model="gender" class="form-control form-control-sm"
        placeholder="Género" required aria-required="true" aria-describedby="genderHelp">
    <small id="genderHelp" class="form-text ">Género para el que está pensado el
        producto. Campo obligatorio</small>
</div>
```

- Estilo: Campo obligatorio que indica el estilo o tipo de diseño (casual, deportivo, etc.)

```
<div class="col-md-6">
    <label for="style" class="form-label">Estilo</label>
    <input type="text" id="style" wire:model="style" class="form-control form-control-sm"
        placeholder="Estilo" required aria-required="true" aria-describedby="styleHelp">
    <small id="styleHelp" class="form-text ">Estilo del producto. Campo obligatorio</small>
</div>
```

- Precio: Campo numérico obligatorio con decimales, que representa el precio del producto.

```
<div class="col-md-6 row mt-3">
    <div class="col-md-6">
        <label for="price" class="form-label">Precio</label>
        <input type="number" id="price" wire:model="price" class="form-control form-control-sm"
            placeholder="Precio" step="0.01" required aria-required="true"
            aria-describedby="priceHelp">
        <small id="priceHelp" class="form-text ">Precio en moneda local. Campo obligatorio</small>
    </div>
```

- Descuento: Campo opcional para indicar un descuento en porcentaje (entre 0 y 90).

```
    <div class="col-md-6">
        <label for="discount" class="form-label">Descuento</label>
        <input type="number" id="discount" wire:model="discount"
            class="form-control form-control-sm" placeholder="Descuento" step="5"
            max="90" min="0" aria-describedby="discountHelp">
        <small id="discountHelp" class="form-text ">Porcentaje de descuento (0-90%)</small>
    </div>
</div>
```

- Selección y Creación de Categorías: Para añadir la categoría del producto se utiliza un campo <select> que carga dinámicamente las categorías existentes desde la base de datos.

```
<div class="col-md-6">
    <label for="category" class="form-label">Categoría</label>
    <select id="category" wire:model="category" wire:change="filterSizes"
        class="form-select form-select-sm" required aria-required="true"
        aria-describedby="categoryHelp">
        <option value="">-- Selecciona una categoría --</option>
        @foreach ($categories as $cat)
            <option value="{{ $cat }}">{{ $cat }}</option>
        @endforeach
    </select>
    <small id="categoryHelp" class="form-text ">Categoría del producto. Campo obligatorio</small>
</div>
```

Por eso, se proporciona un input adicional para permitir la creación de una nueva categoría directamente desde el formulario, ya sea porque no existe ninguna categoría registrada, o porque se desea añadir una nueva categoría sin abandonar el formulario actual.

```
<div class="col-md-6">
    <label for="newCategory" class="form-label">Otra categoría:</label>
    <div class="input-group input-group-sm">
        <input type="text" id="newCategory" wire:model="newCategory" class="form-control"
            placeholder="Nueva Categoría" aria-describedby="newCategoryHelp">
        <button class="btn btn-outline-dark" type="button" wire:click="addNewCategory"
            aria-label="Añadir nueva categoría">
            Añadir
        </button>
    </div>
    <small id="newCategoryHelp" class="form-text ">Escribe una nueva categoría y presiona
        Añadir</small>
</div>
```

El botón "Añadir" asociado permite guardar la nueva categoría en la base de datos e incluirla inmediatamente en el listado del <select>.

- Asignación de Tallas y Stock: Al igual que con las categorías, se incluye un campo adicional que permite registrar una nueva talla. Las tallas disponibles se cargan desde la base de datos y se muestran junto a campos numéricos para añadir individualmente su stock. El sistema exige que al menos una talla tenga stock asignado para poder guardar el producto.

```
<h5 class="my-4" role="heading" aria-level="3">Asignar Stock por Talla</h5>
@error('sizes')
    <div class="text-danger" role="alert">{{ '*Debes asignar stock a al menos una talla.' }}</div>
@enderror
<small id="sizeHelp" class="form-text mb-1">Es OBLIGATORIO que al menos una talla tenga Stock</small>
<div class="d-flex flex-wrap gap-3" role="group" aria-labelledby="sizesLabel">
    @foreach ($availableSizes as $size)
        <div class="d-flex align-items-center mb-2">
            <label for="size-{{ $size->id }}" class="form-label mb-0 me-2"
                style="min-width: fit-content;" id="sizesLabel">{{ $size->name }}</label>
            <input type="number" id="size-{{ $size->id }}"
                wire:model="sizes.{{ $size->id }}" class="form-control form-control-sm me-4"
                placeholder="Stock" min="0" style="width: 80px;"
                aria-describedby="Campo de stock para la talla {{ $size->name }}">
        </div>
    @endforeach
    <div class="d-flex align-items-center mb-2">
        <label for="newSize" class="form-label mb-0 me-2" style="min-width: fit-content;">Otra
            talla:</label>
        <div class="input-group input-group-sm">
            <input type="text" id="newSize" wire:model="newSize"
                class="form-control form-control-sm" placeholder="Ej: XS, M, XL..."
                aria-describedby="newSizeHelp">
            <button class="btn btn-outline-dark" type="button" wire:click="addNewSize"
                aria-label="Añadir nueva talla">
                Añadir
            </button>
        </div>
    </div>
</div>
```

- Botones: El formulario cuenta con los botones para Crear/Actualizar que llaman a su correspondiente método del componente, y un botón para cancelar la accesión y volver para atrás.

```
<div class="col-12 d-flex gap-2">
    <button type="submit" class="btn btn-primary btn-sm"
        aria-label="{{ $product_id ? 'Actualizar producto' : 'Crear producto' }}">
        {{ $product_id ? 'Actualizar' : 'Crear' }}
    </button>
    <button type="button" wire:click="cancel" class="btn btn-secondary btn-sm"
        aria-label="Cancelar acción">Cancelar</button>
</div>
```

1.3.7 - PRODUCTSHOW.PHP

FUNCIONALIDAD: Mostrar la información detallada de un producto seleccionado, incluyendo sus imágenes, descripción, precio (con o sin descuento) y un botón para añadirlo al carrito.

- Variables:
    - **product**: Producto que será mostrado en la vista. Se carga al montar el componente, incluyendo sus imágenes relacionadas. `public $product;`

- mount($id): Carga desde la base de datos el producto correspondiente, incluyendo la relación con sus imágenes (with('images')).

```
public function mount($id): void
{
    // Busca el producto por ID.
    $this->product = Product::with(relations: 'images')->findOrFail(id: $id);

}
```

- render(): Devuelve la vista asociada al componente (livewire.product-show) para su renderizado en el navegador.

```
public function render(): View
{
    // Devuelve la vista 'livewire.product-show' con los datos del producto
    return view(view: 'livewire.product-show');
}
```

VISTA:

- Botón de volver atrás: Usa url()->previous() para regresar a la página anterior.

```
<a href="{{ url()->previous() }}" class="btn btn-secondary mb-3" style="width: 100px;" title="Volver atrás"
    aria-label="Volver a la página anterior">
    <i class="bi bi-arrow-left"></i> Volver
</a>
```

- Imágenes del producto: Se muestra de formas diferentes dependiendo de si hay una o más imágenes.
    - Si hay solo una imagen → se muestra en formato img individual.

```
@if ($product && $product->images->count() > 0)
    @if ($product->images->count() === 1)
        {{-- Solo una imagen --}}
        <img src="{{ asset(path: 'storage/' . $product->images->first()->url) }}"
            alt="Imagen de {{ $product->name }} - {{ $product->category }}, para {{ $product->gender }}"
            aria-label="Imagen del producto {{ $product->name }}"
            title="Imagen del producto {{ $product->name }}" class="img-fluid rounded shadow-sm" />
```

    - Si hay múltiples imágenes → se utiliza un carrusel de Bootstrap personalizado con controles de navegación (flechas izquierda y derecha).

```
@else
    {{-- Carrusel Bootstrap --}}
    <div id="carouselProductImages{{ $product->id }}" class="carousel slide" data-bs-ride="carousel">
        <div class="carousel-inner rounded shadow-sm">
            @foreach ($product->images as $index => $image)
                <div class="carousel-item {{ $index === 0 ? 'active' : '' }}">
                    <img src="{{ asset(path: 'storage/' . $image->url) }}" class="d-block w-100"
                        alt="Imagen {{ $index + 1 }} de {{ $product->name }}"
                        aria-label="Imagen {{ $index + 1 }} del producto {{ $product->name }}"
                        title="Imagen {{ $index + 1 }} del producto {{ $product->name }}">
                </div>
            @endforeach
        </div>

        <!-- Flecha izquierda -->
        <button class="carousel-control-prev" type="button"
            style="background-color: rgba(255, 255, 255, 0.7);"
            data-bs-target="#carouselProductImages{{ $product->id }}" data-bs-slide="prev">
            <span class="carousel-control-prev-icon" style="filter: invert(1);"
                aria-hidden="true"></span>
        </button>

        <!-- Flecha derecha -->
        <button class="carousel-control-next" type="button"
            style="background-color: rgba(255, 255, 255, 0.7);"
            data-bs-target="#carouselProductImages{{ $product->id }}" data-bs-slide="next">
            <span class="carousel-control-next-icon" style="filter: invert(1);"
                aria-hidden="true"></span>
        </button>
    </div>
@endif
```

    - Si no hay imágenes se muestra un mensaje (aunque no debería ocurrir).

```
@else
    <p>No hay imágenes disponibles para este producto.</p>
@endif
```

- Detalles del producto: Se muestra el nombre del producto, su categoría, y género, la descripción, el precio (si hay descuento se muestra el precio original tachado y el nuevo precio en rojo y destacado), y se llama al livewire “<livewire:add-to-cart>”.

```blade
<div class="col-md-8">
    <h3>{{ $product->name }}</h3>
    <p class="text-muted">Categoría: {{ $product->category }} / {{ $product->gender }}</p>

    <!-- Descripción larga -->
    <p>{{ $product->description }}</p>

    <!-- Precio -->
    @if ($product->discount > 0)
        <p>
            <span
                class="text-muted text-decoration-line-through">{{ number_format(num: $product->price, decimals: 2) }}€</span>
            <span
                class="fw-bold fs-5 text-danger ms-2">{{ number_format(num: $product->price - ($product->price * $product->discount) / 100, decimals: 2) }}€</span>
        </p>
    @else
        <p class="fw-bold fs-5">{{ number_format(num: $product->price, decimals: 2) }}€</p>
    @endif

    <livewire:add-to-cart :product="$product" />
</div>
```

1.3.8 - SEEMYORDERS.PHP

FUNCIONALIDAD: Mostrar todos los pedidos realizados por el usuario autenticado, incluyendo los productos, tallas, cantidades, precios y estado del pedido.

- Variables:
    - **orders**: Contiene la colección de pedidos pertenecientes al usuario actual, con todos los datos cargados.

```php
public $orders;
```

- mount(): Obtiene todos los pedidos del usuario autenticado usando Auth::user()->orders(), y utiliza with('items.product', 'items.size') para cargar de forma anticipada los productos y las tallas asociadas a cada ítem del pedido, optimizando las consultas a la base de datos.

```php
public function mount(): void
{
    $this->orders = Auth::user()
        ->orders()
        // Cargamos en la misma consulta los productos y tallas de cada item del pedido para optimizar
        ->with(relations: 'items.product', callback: 'items.size')
        ->get();
}
```

- render(): Devuelve la vista livewire.see-my-orders, que se encargará de mostrar todos los pedidos del usuario.

```php
public function render(): View
{
    // Devuelve la vista 'livewire.see-my-orders' que mostrará los pedidos
    return view(view: 'livewire.see-my-orders');
}
```

VISTA:

- Mostrar los pedidos: Se comprueba si el usuario tiene pedidos ($orders->count()).

- Si tiene, se itera sobre cada uno de ellos, y se muestra los detalles más relevantes como el id del pedido, la fecha, el estado, precio y los productos con su talla, precio individual y cantidad.

```
@if ($orders->count())
    @foreach ($orders as $order)
        <div class="card mb-3">
            <div class="card-header">
                <strong>Pedido #{{ $order->id }}</strong> |
                Fecha: {{ $order->created_at->format('d/m/Y') }} |
                Estado: <span class="badge bg-info">{{ $order->status }}</span>
            </div>
            <div class="card-body">
                @foreach ($order->items as $item)
                    <div class="row mb-2">
                        <div class="col-md-2">
                            <img src="{{ asset(path: 'storage/' . $item->product->image) }}"
                                alt="{{ $item->product->name }}" class="img-fluid">
                        </div>
                        <div class="col-md-10">
                            <h5>{{ $item->product->name }}</h5>
                            <p>{{ $item->product->description }}</p>
                            <p>
                                <strong>Cantidad:</strong> {{ $item->quantity }} |
                                <strong>Precio:</strong> ${{ $item->price }} |
                                <strong>Talla:</strong> {{ $item->size->name ?? 'N/A' }}
                            </p>
                        </div>
                    </div>
                    <hr>
                @endforeach
                <div class="text-end">
                    <strong>Total:</strong> ${{ $order->total }}
                </div>
            </div>
        </div>
    @endforeach
```

- Y si no tiene pedidos, se muestra un mensaje informativo con un alert de bootstrap.

```
@else
    <div class="alert alert-info">
        Aún no tienes pedidos realizados.
    </div>
@endif
```

1.3.9 - SHOWCART.PHP

FUNCIONALIDAD: Este componente gestiona la visualización, selección, modificación y eliminación de productos del carrito de compras del usuario autenticado. También permite proceder con los productos seleccionados para la creación de un pedido.

- Variables:
  - **cartItems**: Lista de items del carrito del usuario con relación a productos e imágenes.
  - **totalPrice**: Precio total de los productos seleccionados.
  - **selectedItems**: IDs de los productos seleccionados para la compra.
  - **itemToDelete**: ID del producto marcado para eliminación (si se usa en un modal de confirmación).
  - **message**: Mensaje de estado mostrado al usuario.
  - **messageType**: Tipo de mensaje (success, error, warning).
  - **showMessage**: Booleano para controlar la visibilidad del mensaje.

```php
public $cartItems;
1 reference
public $totalPrice = 0;
7 references
public $selectedItems = [];
0 references
public $itemToDelete = null;
1 reference
public $message = null;
1 reference
public $messageType = 'success';
1 reference
public $showMessage = false;
```

- mount(): Se ejecuta al cargar el componente. Obtiene o crea el carrito del usuario autenticado, carga los items del carrito con productos e imágenes asociadas, y si el carrito está vacío, muestra un mensaje de advertencia. También llama al método de "updateTotal()".

```php
public function mount(): void
{
    $user = auth()->user();

    $cart = $user->cart;

    if (!$cart) {
        // Si el usuario no tiene carrito, se crea uno vacío
        $cart = $user->cart()->create();
    }

    // Carga los items del carrito con la relación de producto e imágenes
    $this->cartItems = $cart->items()->with('product.images')->get();

    // Si el carrito está vacío, mostramos mensaje de advertencia
    if ($this->cartItems->isEmpty()) {
        $this->setMessage(message: 'Todavía no hay productos en el carrito', messageType: 'warning');
    }

    // Calcula el total de productos seleccionados (inicialmente ninguno)
    $this->updateTotal();
}
```

- itemSelection($itemId): Guarda o saca del array de $selectedItems las tarjetas que estén o no seleccionadas, y actualiza el total tras cada cambio llamando a la función updateTotal().

```php
public function itemSelection($itemId): void
{
    // Si el item ya estaba seleccionado, lo quitamos
    if (in_array(needle: $itemId, haystack: $this->selectedItems)) {
        $this->selectedItems = array_filter(array: $this->selectedItems, callback: function ($id) use ($itemId): bool {
            return $id != $itemId;
        });
    } else {
        // Si no estaba seleccionado, lo añadimos
        $this->selectedItems[] = $itemId;
    }

    // Actualizamos el total teniendo en cuenta los productos seleccionados
    $this->updateTotal();
}
```

- updateTotal(): Recalcula el precio total según los productos seleccionados, y aplica descuentos si los hay.

```php
public function updateTotal(): void
{
    $total = 0; // Inicializamos el total en 0

    foreach ($this->cartItems as $item) {
        // Verificamos si el ítem está entre los seleccionados
        if (in_array(needle: $item->id, haystack: $this->selectedItems)) {

            // Precio base del producto (sin descuento)
            $price = $item->product->price;

            // Si el producto tiene un descuento, se aplica
            if ($item->product->discount && $item->product->discount > 0) {
                $discount = $item->product->discount;
                $price = $price - ($price * $discount / 100); // Precio con descuento aplicado
            }

            // Sumamos al total la cantidad multiplicada por el precio (con o sin descuento)
            $total += $item->quantity * $price;
        }
    }

    // Asignamos el total calculado a la propiedad del componente
    $this->totalPrice = $total;
}
```

- confirmDeletion($itemId): Elimina el producto del carrito, refresca la lista de items y el total y muestra un mensaje confirmando la eliminación del producto del carrito.

```php
public function confirmDeletion($itemId): void
{
    $cartItem = auth()->user()->cart->items()->find($itemId);
    if ($cartItem) {
        $cartItem->delete();
        // Actualizamos la lista de items del carrito y el total
        $this->cartItems = auth()->user()->cart->items;
        $this->updateTotal();
        $this->setMessage(message: 'Producto eliminado del carrito', messageType: 'success');
    }

    // Si el carrito queda vacío, mostramos mensaje de advertencia
    if ($this->cartItems->isEmpty()) {
        $this->setMessage(message: 'Todavía no hay productos en el carrito', messageType: 'warning');
    }
}
```

- updateCartItem($itemId, $quantity, $size): Actualiza la cantidad y talla de un item del carrito, refresca el total, y muestra mensaje de confirmación o error.

```php
public function updateCartItem($itemId, $quantity, $size): void
{
    $cartItem = auth()->user()->cart->items()->find($itemId);
    if ($cartItem) {
        $cartItem->quantity = $quantity;
        $cartItem->size_id = $size;
        $cartItem->save();

        // Refrescamos los items y el total para reflejar los cambios
        $this->cartItems = auth()->user()->cart->items;
        $this->updateTotal();
        $this->setMessage(message: 'Producto actualizado', messageType: 'success');
    } else {
        $this->setMessage(message: 'No se ha podido actualizar el producto', messageType: 'error');
    }
}
```

- setMessage($message, $messageType = 'success'): Establece el mensaje de estado que se mostrará en la vista.

```php
public function setMessage($message, $messageType = 'success'): void
{
    $this->message = $message;
    $this->messageType = $messageType;
    $this->showMessage = true;
}
```

- makeOrder(): Verifica si hay productos seleccionados, guarda en sesión los IDs seleccionados, y redirige a la ruta user.orderConfirmation.

```php
public function makeOrder(): mixed|RedirectResponse
{
    if (empty($this->selectedItems)) {
        $this->setMessage(message: 'Debes seleccionar al menos un producto', messageType: 'warning');
        return;
    }

    // Guardamos en sesión los IDs de los productos seleccionados para usar en la confirmación
    session()->put(key: 'selectedItems', value: $this->selectedItems);

    // Redirigimos a la ruta de confirmación de pedido
    return redirect()->route(route: 'user.orderConfirmation');
}
```

- render(): Devuelve la vista livewire.show-cart.

```php
public function render(): View
{
    return view(view: 'livewire.show-cart');
}
```

VISTA:

- Mensajes de estado:

```
@if ($message)
    <div class="alert alert-{{ $messageType == 'success' ? 'success' : ($messageType == 'warning' ? 'warning' : 'danger') }} mb-4"
        role="alert" aria-live="assertive" aria-atomic="true" title="Mensaje de estado">
        {{ $message }}
    </div>
@endif
```

- Mostrar los productos: Se itera sobre $cartItems, y se van mostrando cada producto que está guardado en el carrito, mostrandose de forma visual como una tarjeta que contiene:
    - Un checkbox para seleccionar el producto.

```
<div class="row" role="list" aria-label="Lista de productos en el carrito">
    @foreach ($cartItems as $item)
        <div class="col-12 mb-3" role="listitem">
            <div class="card p-3" aria-label="Producto {{ $item->product->name }}">
                <div class="row align-items-center">
                    <!-- Check box -->
                    <div class="col-auto">
                        <!-- Checkbox -->
                        <input type="checkbox" wire:click="itemSelection({{ $item->id }})"
                            {{ in_array(needle: $item->id, haystack: $selectedItems) ? 'checked' : '' }}
                            aria-label="Seleccionar producto {{ $item->product->name }}"
                            title="Seleccionar producto {{ $item->product->name }}">
                    </div>
```

    - Una imagen del producto.

```
<!-- Imagen del producto -->
<div class=" col-3 d-flex justify-content-center">
    <img src="{{ asset(path: 'storage/' . $item->product->images->first()->url) }}"
        alt="Imagen de {{ $item->product->name }} - {{ $item->product->category }}, para {{ $item->product->gender }}"
        aria-label="Imagen del producto {{ $item->product->name }}"
        title="Imagen del producto {{ $item->product->name }}"
        class="img-fluid rounded shadow-sm" style="max-height: 250px;">
</div>
```

    - El nombre y descripción.

```
<div class="col" role="region"
    aria-label="Detalles del producto {{ $item->product->name }}">
    <h3 class="h5">{{ $item->product->name }}</h5>
        <p class="text-muted">{{ $item->product->description }}</p>
```

    - Talla y cantidad seleccionada.

```blade
<div class="row">
    <div class="col-md-6">
        <div class="form-group">
            <label for="size-{{ $item->id }}"
                title="Etiqueta para talla del producto {{ $item->product->name }}">
                Talla
            </label>
            <select wire:model="size" class="form-control"
                id="size-{{ $item->id }}"
                wire:change="updateCartItem({{ $item->id }}, {{ $item->quantity }}, $event.target.value)"
                aria-label="Seleccionar talla para {{ $item->product->name }}"
                title="Seleccionar talla para {{ $item->product->name }}">
                @foreach ($item->product->sizes as $size)
                    <option value="{{ $size->id }}"
                        {{ $size->id == $item->size_id ? 'selected' : '' }}>
                        {{ $size->name }}
                    </option>
                @endforeach
            </select>
        </div>
    </div>
    <div class="col-md-6">
        <div class="form-group">
            <label for="quantity-{{ $item->id }}"
                title="Etiqueta para cantidad del producto {{ $item->product->name }}">
                Cantidad
            </label>
            <input type="number" wire:model="quantity" min="1"
                class="form-control" id="quantity-{{ $item->id }}"
                value="{{ $item->quantity }}"
                wire:change="updateCartItem({{ $item->id }}, $event.target.value, {{ $item->size_id }})"
                aria-label="Cantidad para {{ $item->product->name }}"
                title="Cantidad para {{ $item->product->name }}">
        </div>
    </div>
</div>
```

- Precio con o sin descuento.

```blade
<div class="row">
    <div class="col-md-6">
        <p class="fs-6 fw-semibold"
            aria-label="Precio unidad del producto {{ $item->product->name }}">
            Precio Unidad:
            @if ($item->product->discount)
                <span class="text-muted text-decoration-line-through"
                    aria-label="Precio original tachado">
                    {{ number_format(num: $item->product->price, decimals: 2) }} €
                </span>
                <span class="text-danger fw-bold fs-5"
                    aria-label="Precio con descuento">
                    {{ number_format(num: $item->product->price * (1 - $item->product->discount / 100), decimals: 2) }}
                    €
                </span>
            @else
                {{ number_format(num: $item->product->price, decimals: 2) }} €
            @endif
        </p>
    </div>
```

- Precio total del producto.

```
<div class="col-md-6">
    <p class="fs-6 fw-semibold"
        aria-label="Precio total del producto {{ $item->product->name }}">
        Precio Total:
        {{ number_format(num: $item->product->price * (1 - $item->product->discount / 100) * $item->quantity, decimals: 2) }}
        €
    </p>
</div>
```

- Botones para ver detalles del producto o para eliminar del carrito.

```
<div class="col-auto d-flex justify-content-around flex-column" role="group"
    aria-label="Acciones para el producto {{ $item->product->name }}">
    <a href="{{ route(name: 'productos.show', parameters: $item->product->id) }}"
        class="btn btn-outline-primary mb-2" aria-label="Ver Producto" title="Ver Producto">
        <i class="bi bi-eye" aria-hidden="true"></i> Ver Producto
    </a>
    <a wire:click="confirmDeletion({{ $item->id }})" class="btn btn-danger mt-2"
        role="button" tabindex="0"
        aria-label="Eliminar producto {{ $item->product->name }} del carrito"
        title="Eliminar producto {{ $item->product->name }} del carrito">
        <i class="bi bi-cart-dash" aria-hidden="true"></i> Eliminar
    </a>
</div>
```

- Conteo de productos seleccionados y su precio final:

```
<p class="text-right me-5 m-0" aria-label="Cantidad de productos seleccionados">
    <strong>Cantidad de productos seleccionados: {{ count(value: $selectedItems) }}</strong>
</p>
<p class="text-right me-5 m-0" aria-label="Precio total de productos seleccionados">
    <strong>Total: {{ number_format(num: $totalPrice, decimals: 2) }} €</strong>
</p>
```

- Botón para realizar el pedido:

```
<button class="btn btn-primary me-3" wire:click="makeOrder" aria-label="Realizar pedido"
    title="Realizar pedido">
    Realizar pedido
</button>
```

1.3.10 - SHOWSTOREPROD.PHP

FUNCIONALIDAD: Este componente permite filtrar productos en la tienda donde se muestran todos los productos que hay en la base de datos. Presenta un formulario de búsqueda avanzada y dinámico, y muestra los resultados directamente sin recargar la página, gracias al uso de Livewire.

- Variables:
  - **gender**: género del producto.
  - **categories**: categorías seleccionadas.
  - **color**: color del producto.
  - **style**: estilo del producto.
  - **priceMin**: precio mínimo.
  - **priceMax**: precio máximo.
  - **sizes**: tallas seleccionadas.
  - **onlyOffers**: si se deben mostrar sólo productos con descuento.
  - **prodFiltrado**: resultados filtrados.
  - **availableSizes**: todas las tallas disponibles.
  - **gendersList, categoriesList, colorsList, stylesList:** valores únicos de productos para generar filtros dinámicos.

```php
public $gender;
4 references
public $categories = [];
4 references
public $color;
4 references
public $priceMin;
4 references
public $priceMax;
4 references
public $sizes = [];
1 reference
public $availableSizes;
4 references
public $style;

// Listas para mostrar opciones de filtros dinámicos
1 reference
public $gendersList = [];
1 reference
public $categoriesList = [];
1 reference
public $colorsList = [];
1 reference
public $stylesList = [];

0 references
public $orderBy;

3 references
public $onlyOffers = false;

6 references
public $prodFiltrado;
```

- mount(): Inicializa las listas dinámicas para los filtros (géneros, categorías, colores, estilos) a partir de los valores únicos en la base de datos, y carga todas las tallas desde la tabla sizes.

```php
public function mount(): void
{
    // Obtenemos valores distintos de cada atributo para los filtros
    $this->gendersList = Product::distinct()->pluck(column: 'gender')->toArray();
    $this->categoriesList = Product::distinct()->pluck(column: 'category')->toArray();
    $this->colorsList = Product::distinct()->pluck(column: 'color')->toArray();
    $this->stylesList = Product::distinct()->pluck(column: 'style')->toArray();

    // Todas las tallas disponibles (tabla Size)
    $this->availableSizes = Size::all();
}
```

- filter(): Construye dinámicamente una consulta aplicando los filtros seleccionados por el usuario.

  Se puede filtrar por género, categorías, color, estilo, tallas, precio (mínimo y máximo), y descuento.

```php
public function filter(): void
{
    $query = Product::query();

    $this->prodFiltrado = null;

    // Aplicamos filtro por género
    if (!empty($this->gender)) {
        $query->where(column: 'gender', operator: $this->gender);
    }

    // Filtro por categorías (pueden ser múltiples)
    if (!empty($this->categories)) {
        $query->whereIn(column: 'category', values: $this->categories);
    }
```

```php
// Filtro por color
if (!empty($this->color)) {
    $query->where(column: 'color', operator: $this->color);
}

// Filtro por estilo
if (!empty($this->style)) {
    $query->where(column: 'style', operator: $this->style);
}

// Precio mínimo
if (!empty($this->priceMin)) {
    $query->where(column: 'price', operator: '>=', value: $this->priceMin);
}

// Precio máximo
if (!empty($this->priceMax)) {
    $query->where(column: 'price', operator: '<=', value: $this->priceMax);
}

// Filtro por tallas (relación many-to-many)
if (!empty($this->sizes)) {
    $query->whereHas(relation: 'sizes', callback: function (Builder<TRelatedModel> $q)… {
        $q->whereIn(column: 'name', values: $this->sizes);
    });
}

// Mostrar solo productos en oferta
if ($this->onlyOffers) {
    $query->where(column: 'discount', operator: '>', value: 0);
}
```

Además, se comprueba si no se ha aplicado ningún filtro y lanza un mensaje con session()->flash() para avisar al usuario, o si no hay ningún producto con las características seleccionadas.

```php
// Si no hay filtros aplicados, mostramos error y limpiamos resultados
if (
    empty($this->gender) && empty($this->categories) && empty($this->color) && empty($this->style) &&
    empty($this->priceMin) && empty($this->priceMax) && empty($this->sizes) && !$this->onlyOffers
) {
    session()->flash(key: 'error', value: 'Introduce algún dato para filtrar');
    $this->prodFiltrado = collect(); // Mejor usar colección vacía que array
    return;
}

// Ejecutamos la consulta y guardamos los productos filtrados
$this->prodFiltrado = $query->with(relations: 'images')->get();

// Si no hay resultados, mostramos mensaje de error
if ($this->prodFiltrado->isEmpty()) {
    session()->flash(key: 'error', value: 'No se ha encontrado ningún producto con esas características.');
}
```

- resetFilters(): Limpia todos los campos de filtro y reinicia el listado.

```php
public function resetFilters(): void
{
    $this->gender = null;
    $this->categories = [];
    $this->color = null;
    $this->priceMin = null;
    $this->priceMax = null;
    $this->sizes = [];
    $this->style = null;
    $this->onlyOffers = false;
    $this->prodFiltrado = null;
}
```

- render(): Devuelve la vista livewire.show-cart, pasando los productos filtrados si existen, o todos los productos por defecto.

```php
public function render(): View
{
    $products = $this->prodFiltrado ?? Product::with(relations: 'images')->get();
    return view(view: 'livewire.show-store-prod', data: compact(var_name: 'products'));
}
```

VISTA:

- Formulario de filtros (en <aside>): Exceptuando los filtros de precio y descuento, el resto de los filtros se generan dinámicamente a partir de los valores existentes en la base de datos de productos. Es decir, se recorren los registros almacenados y se extraen los valores únicos de cada campo para escribir los filtros correspondientes que son:
    - Género (radio)

```blade
<!-- Sidebar con formulario -->
<aside class="col-md-3 mb-4">
    <form class="p-5 rounded border bg-light" wire:submit.prevent="filter"
        aria-label="Formulario de filtro de productos">

        <!-- Género -->
        <div class="mb-4">
            <h3 class="h5">Género</h3 class="h5">
            @foreach ($gendersList as $genero)
                <div class="form-check">
                    <input class="form-check-input" type="radio" wire:model="gender"
                        id="gender-{{ $genero }}" name="gender" value="{{ $genero }}"
                        aria-label="Género {{ $genero }}" title="Género {{ $genero }}">
                    <label class="form-check-label" for="gender-{{ $genero }}">{{ ucfirst(string: $genero) }}</label>
                </div>
            @endforeach
        </div>
```

    - Categorías (checkboxes múltiples)

```blade
<div class="mb-4">
    <h3 class="h5">Categoría</h3 class="h5">
    <div class="row">
        @foreach ($categoriesList as $categoria)
            <div class="col-6 col-md-4 col-lg-3 me-5 pe-5">
                <div class="form-check">
                    <input class="form-check-input" type="checkbox" wire:model="categories"
                        id="cat-{{ $categoria }}" name="categories[]" value="{{ $categoria }}"
                        aria-label="Categoría {{ $categoria }}" title="Categoría {{ $categoria }}">
                    <label class="form-check-label"
                        for="cat-{{ $categoria }}">{{ ucfirst(string: $categoria) }}</label>
                </div>
            </div>
        @endforeach
    </div>
</div>
```

- Color (select)

```
<div class="mb-4">
    <h3 class="h5">Color</h3 class="h5">
    <select class="form-select" wire:model="color" name="color" id="color"
        aria-label="Filtrar por color" title="Selecciona un color">
        <option value="">Todos</option>
        @foreach ($colorsList as $colorItem)
            <option value="{{ $colorItem }}">{{ ucfirst(string: $colorItem) }}</option>
        @endforeach
    </select>
</div>
```

- Talla (checkboxes múltiples)

```
<div class="mb-4">
    <h3 class="h5">Talla</h3 class="h5">
    <div class="row">
        @foreach ($availableSizes as $size)
            <div class="col-6 col-md-4 col-lg-3 me-3">
                <div class="form-check">
                    <input class="form-check-input" type="checkbox" wire:model="sizes"
                        id="size-{{ $size->name }}" name="sizes[]" value="{{ $size->name }}"
                        aria-label="Talla {{ strtoupper(string: $size->name) }}"
                        title="Talla {{ strtoupper(string: $size->name) }}">
                    <label class="form-check-label"
                        for="size-{{ $size->name }}">{{ strtoupper(string: $size->name) }}</label>
                </div>
            </div>
        @endforeach
    </div>
</div>
```

- Estilo (radio)

```
<div class="mb-4">
    <h3 class="h5">Estilo</h3 class="h5">
    @foreach ($stylesList as $styleItem)
        <div class="form-check">
            <input class="form-check-input" type="radio" wire:model="style"
                id="style-{{ $styleItem }}" name="style" value="{{ $styleItem }}"
                aria-label="Estilo {{ $styleItem }}" title="Estilo {{ $styleItem }}">
            <label class="form-check-label"
                for="style-{{ $styleItem }}">{{ ucfirst(string: $styleItem) }}</label>
        </div>
    @endforeach
</div>
```

- Precio mínimo y máximo (inputs numéricos)

```
<div class="mb-4">
    <h3 class="h5">Precio</h3 class="h5">
    <input type="number" wire:model="priceMin" class="form-control" placeholder="Precio mínimo"
        id="priceMin" name="priceMin" min="0" step="0.01" aria-label="Precio mínimo"
        title="Precio mínimo">
    <input type="number" wire:model="priceMax" class="form-control mt-2" placeholder="Precio máximo"
        id="priceMax" name="priceMax" min="0" step="0.01" aria-label="Precio máximo"
        title="Precio máximo">
</div>
```

- Checkbox para mostrar solo ofertas

```
<div class="mb-4">
    <h3 class="h5">Ofertas</h3 class="h5">
    <div class="form-check">
        <input class="form-check-input" type="checkbox" wire:model="onlyOffers" id="onlyOffers"
            name="onlyOffers" aria-label="Solo productos en oferta"
            title="Mostrar solo productos en oferta">
        <label class="form-check-label" for="onlyOffers">Mostrar solo productos en oferta</label>
    </div>
</div>
```

- Botones de "Filtrar" y "Borrar filtros"

```
<div class="d-grid gap-2">
    <button type="submit" class="btn btn-primary" aria-label="Filtrar" title="Filtrar">Filtrar</button>
    <button type="reset" class="btn btn-outline-dark"
        onmouseover="this.classList.replace('btn-outline-dark','btn-outline-secondary')"
        onmouseout="this.classList.replace('btn-outline-secondary','btn-outline-dark')"
        wire:click="resetFilters" aria-label="Borrar filtros" title="Borrar filtros">
        Borrar filtros
    </button>

</div>
```

- Resultados de productos (en <section>): Muestra los productos (filtrados o todos por defecto) en tarjetas de Bootstrap, cada una con:
  - Imagen del producto

```
<section class="col-md-9">
    <div class="row">
        <!-- Fila de productos -->
        @foreach ($prodFiltrado ?? $products as $product)
            <div class="col-md-6 col-lg-4 mb-4">
                <div class="card h-100 shadow-sm">
                    <img src="{{ asset(path: 'storage/' . $product->images->first()->url) }}"  class="card-img-top"
                        alt="Imagen de {{ $product->name }}">
```

  - Nombre y descripción

```
<div class="card-body d-flex flex-column">
    <h3 class="h5" class="card-title">{{ $product->name }}</h3 class="h5">
    <p class="card-text">{{ $product->description }}</p>
```

  - Precio, con precio original tachado si hay descuento

```
@if ($product->discount > 0)
    <p>
        <span class="text-muted text-decoration-line-through">
            {{ number_format(num: $product->price, decimals: 2) }}€
        </span>
        <span class="fw-bold fs-5 text-danger ms-2">
            {{ number_format(num: $product->price - ($product->price * $product->discount) / 100, decimals: 2) }}€
        </span>
    </p>
@else
    <p class="fw-bold fs-5">{{ number_format(num: $product->price, decimals: 2) }}€</p>
@endif
```

- Llama al livewire Addtocart.php, para añadir ese producto al carrito.

```
<livewire:add-to-cart :product="$product" :key="'add-to-cart-' . $product->id . '-' . $loop->index" />
```

- Botón para ver los detalles del producto: redirige a user.product.

```
<a href="{{ route(name: 'productos.show', parameters: $product->id) }}"
    class="btn btn-outline-primary w-100 mt-3"
    aria-label="Ver detalles del producto {{ $product->name }}"
    title="Ver detalles del producto {{ $product->name }}">Ver detalles del
    Producto</a>
```

1.3.11 – USERCRUD.PHP

FUNCIONALIDAD: Gestión completa de usuarios del sistema (crear, leer, actualizar, eliminar) con filtros dinámicos por nombre y rol. También permite alternar entre vista de lista y formulario de edición/creación.

- Variables:
  - **users, user_id, name, email, role:** datos del usuario.
  - **filterName**: Texto a filtrar por nombre.
  - **filterRole**: Filtro por tipo de rol.
  - **roles**: Mapea los roles con sus nombres para los select.
  - **view**: Determina qué vista se muestra (list o form).

```
public $users, $user_id, $name, $email, $role;
3 references
public $filterName = '';
3 references
public $filterRole = '';

// Roles definidos para mostrar en un select (clave => valor)
0 references
public $roles = [
    '0' => 'Usuario',
    '1' => 'Administrador',
];

// Controla la vista actual, puede ser 'list' o 'form'
5 references
public $view = 'list';
```

- mount(): Carga inicial de todos los usuarios al iniciar el componente, llamando a la función “loadUsers()”.

```
public function mount(): void
{
    $this->loadUsers();
}
```

- filter(): Filtra usuarios por nombre (LIKE) y por rol (exacto). Si no hay resultado manda un mensaje de error.

```
public function filter(): void
{
    $query = User::query();

    if (!empty($this->filterName)) {
        // Busca usuarios cuyo nombre contenga el texto del filtro (consulta LIKE)
        $query->where(column: 'name', operator: 'like', value: '%' . $this->filterName . '%');
    }

    if ($this->filterRole !== '') {
        // Filtra por rol exacto
        $query->where(column: 'role', operator: $this->filterRole);
    }

    // Ejecuta la consulta y guarda los resultados en $this->users
    $this->users = $query->get();

    if ($this->users->isEmpty()) {
        // Si no hay resultados, muestra mensaje de error y recarga todos usuarios
        session()->flash(key: 'error', value: 'No se encontraron usuarios con esos filtros.');
        $this->loadUsers();
    }
}
```

- resetFilters(): Limpia todos los filtros y recarga todos los usuarios.

```php
public function resetFilters(): void
{
    $this->filterName = '';
    $this->filterRole = '';
    $this->loadUsers();
}
```

- resetInputs(): Limpia los campos del formulario.

```php
public function resetInputs(): void
{
    $this->user_id = null;
    $this->name = '';
    $this->email = '';
    $this->role = '';
}
```

- showCreateForm(): Cambia la vista a formulario para crear un nuevo usuario.

```php
public function showCreateForm(): void
{
    $this->resetInputs();
    $this->view = 'form';
}
```

- edit($id): Carga un usuario específico por ID para edición.

```php
public function edit($id): void
{
    $user = User::findOrFail(id: $id);
    $this->user_id = $user->id;
    $this->fill(values: $user->toArray());
    $this->view = 'form';
}
```

- store(): Valida y guarda un nuevo usuario en la base de datos.

```php
public function store(): void
{
    $this->validate(rules: [
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email',
        'role' => 'required|in:0,1',
    ]);

    User::create(attributes: $this->validateData());

    session()->flash(key: 'message', value: 'Usuario creado correctamente.');
    $this->resetInputs();
    $this->view = 'list';
    $this->loadUsers();
}
```

- update(): Valida y actualiza un usuario existente.

```php
public function update(): void
{
    $this->validate(rules: [
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email,' . $this->user_id,
        'role' => 'required|in:0,1',
    ]);

    $user = User::findOrFail(id: $this->user_id);
    $user->update($this->validateData());

    session()->flash(key: 'message', value: 'Usuario actualizado correctamente.');
    $this->resetInputs();
    $this->view = 'list';
    $this->loadUsers();
}
```

- cancel(): Cancela la creación o edición y regresa a la vista de lista.

```php
public function cancel(): void
{
    $this->resetInputs();
    $this->view = 'list';
}
```

- delete($id): Elimina un usuario por su ID.

```php
public function delete($id): void
{
    User::findOrFail(id: $id)->delete();
    session()->flash(key: 'message', value: 'Usuario eliminado correctamente.');
    $this->loadUsers();
}
```

- loadUsers(): Método privado reutilizable para cargar todos los usuarios.

```php
private function loadUsers(): void
{
    $this->users = User::all();
}
```

- validateData(): Método privado que retorna los datos validados para crear/editar.

```php
private function validateData(): array
{
    return [
        'name' => $this->name,
        'email' => $this->email,
        'role' => $this->role,
    ];
}
```

- render(): Devuelve la vista user-crud.blade.php.

```php
public function render(): View
{
    return view(view: 'livewire.user-crud');
}
```

VISTA:

Vista list:

- Filtros por nombre y rol con botones para aplicar y borrar filtros.

```blade
<div class="card mb-4">
    <div class="card-header bg-light">
        <strong>Filtrar Usuarios</strong>
    </div>
    <div class="card-body">
        <form wire:submit.prevent="filter" class="row g-3 align-items-end">
            <div class="col-md">
                <label for="filterName" class="form-label">Nombre</label>
                <input type="text" id="filterName" name="filterName" wire:model.defer="filterName" class="form-control" placeholder="Ej: María">
            </div>

            <div class="col-md">
                <label for="filterRole" class="form-label">Rol</label>
                <select id="filterRole" name="filterRole" wire:model.defer="filterRole" class="form-select">
                    <option value="">-- Todos --</option>
                    @foreach ($roles as $key => $label)
                        <option value="{{ $key }}">{{ $label }}</option>
                    @endforeach
                </select>
            </div>

            <div class="col-auto">
                <button type="submit" class="btn btn-primary">
                    <i class="bi bi-search"></i> Filtrar
                </button>
            </div>

            <div class="col-auto">
                <button type="button" wire:click="resetFilters" class="btn btn-secondary">
                    <i class="bi bi-x-circle"></i> Borrar Filtros
                </button>
            </div>
        </form>
    </div>
</div>
```

- Mensajes de estado:

```blade
@if (session()->has(key: 'message'))
    <div class="alert alert-success">{{ session(key: 'message') }}</div>
@endif

@if (session()->has(key: 'error'))
    <div class="alert alert-danger">{{ session(key: 'error') }}</div>
@endif
```

- Tabla responsiva con los campos id, nombre, email, rol ("Administrador" o "Usuario"), y los botones para editar o eliminar. Al pulsar el botón de eliminar debería salir un alert preguntando si está seguro de realizar esa acción.

```blade
<table class="table table-bordered table-striped w-100 align-items-center">
    <thead class="table-light">
        <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Email</th>
            <th>Rol</th>
            <th>Acciones</th>
        </tr>
    </thead>
```

```
    <tbody>
        @forelse($users as $user)
            <tr>
                <td>{{ $user->id }}</td>
                <td>{{ $user->name }}</td>
                <td>{{ $user->email }}</td>
                <td>{{ $user->role == 1 ? 'Administrador' : 'Usuario' }}</td>
                <td>
                    <button wire:click="edit({{ $user->id }})" class="btn btn-sm btn-warning">
                        <i class="bi bi-pencil"></i> Editar
                    </button>
                    <button wire:click="delete({{ $user->id }})" class="btn btn-sm btn-danger"
                        onclick="confirm('¿Estás seguro de eliminar este usuario?') || event.stopImmediatePropagation()">
                        <i class="bi bi-trash"></i> Eliminar
                    </button>
                </td>
            </tr>
        @empty
            <tr>
                <td colspan="5" class="text-center">No hay usuarios</td>
            </tr>
        @endforelse
    </tbody>
</table>
```

Vista form:

- Formulario para editar: Contiene los campos nombre, email y rol, siendo el rol un select donde solo hay dos opciones o usuario o administrador.

```
<h2 class="mb-5 my-4 text-center">Editar Usuario</h2>

<div class="card p-3">
    <form wire:submit.prevent="update">
        <div class="mb-3">
            <label for="name" class="form-label">Nombre:</label>
            <input type="text" id="name" name="name" wire:model.defer="name" class="form-control">
            @error('name')
                <span class="text-danger">{{ $errors->first('name') }}</span>
            @enderror
        </div>

        <div class="mb-3">
            <label for="email" class="form-label">Email:</label>
            <input type="email" id="email" name="email" wire:model.defer="email" class="form-control">
            @error('email')
                <span class="text-danger">{{ $errors->first('email') }}</span>
            @enderror
        </div>

        <div class="mb-3">
            <label for="role" class="form-label">Rol:</label>
            <select id="role" name="role" wire:model.defer="role" class="form-select">
                <option value="">Selecciona un rol</option>
                <option value="0">Usuario</option>
                <option value="1">Administrador</option>
            </select>
            @error('role')
                <span class="text-danger">{{ $errors->first('role') }}</span>
            @enderror
        </div>
```

- Botones: para guardar los cambios o cancelar y volver a la lista.

```
        <button type="submit" class="btn btn-primary">Guardar Cambios</button>
        <button type="button" wire:click="cancel" class="btn btn-secondary ms-2">Cancelar</button>
    </form>
</div>
```

1.4 - GESTIÓN DE USUARIOS Y ROLES

Roles definidos (ej: admin = 1, usuario = 0)

Middleware personalizado para control de acceso

Redirecciones post-login según el rol

Autenticación y verificación

1.5 - SISTEMA DE CORREO ELECTRÓNICO

Cómo funciona el carrito

Cómo se realiza un pedido

Sistema de descuentos/ofertas (si lo usaste)

Gestión de stock

1.6 - LÓGICA DE NEGOCIO

Qué correos se envían (ej. confirmación de pedido, registro, etc.)

Cómo están configurados (archivo .env, servicios como Mailtrap, Gmail, etc.)

Qué clases de notificación o Mailable usaste

Vista del correo (si usaste resources/views/emails)

PRUEBAS

Para asegurar el correcto funcionamiento del ecommerce, se plantea realizar diferentes tipos de pruebas, tanto funcionales como de accesibilidad.

1.1 - FUNCIONALES

1.1.1 - UNITARIAS

Se tiene previsto realizar **pruebas unitarias**, enfocadas en validar el comportamiento de funciones o componentes individuales del sistema. Esto incluye validaciones sobre lógica de negocio.

Ejemplos de pruebas unitarias:

- Comprobar que el cálculo del total del carrito es correcto según los productos añadidos.
- Validar que el stock se actualiza correctamente después de realizar un pedido.
- Verificar que se genera correctamente el resumen de un pedido por usuario.

Estas pruebas son clave para detectar errores tempranos y asegurar que cada componente individual del sistema se comporta como se espera.

1.1.2 - DE INTERFAZ (END-TO-END)

Además, basándome en la experiencia adquirida en el entorno laboral, se realizarán **pruebas automatizadas de interfaz.** Estas pruebas simulan la experiencia de un usuario real que interactúa con la aplicación desde el navegador.

Se utilizarán herramientas como **Cypress + Cucumber** o **Selenium + Cucumber**, que permiten definir escenarios en lenguaje natural (Gherkin) y automatizar su ejecución.

Ejemplos de pruebas end-to-end:

- Simular el comportamiento al pulsar botones y comprobar la navegación o acción realizada.
- Rellenar formularios con datos válidos e inválidos y verificar las respuestas del sistema.
- Comprobar que no se puede enviar un formulario vacío y que se muestran los errores correctamente.
- Verificar que cuando un formulario se completa correctamente, se procesa sin errores.

Estas pruebas ayudan a validar los principales flujos del ecommerce desde la perspectiva del usuario final.

1.2 - DE ACCESIBILIDAD

El proyecto también incluirá **pruebas de accesibilidad web**, siguiendo como referencia la norma **UNE-EN 301549:2022**, que establece los criterios para que un sitio web sea accesible para todas las personas, incluyendo aquellas con discapacidades.

La evaluación se centrará en los puntos **9 (Web), 11 (Software)** y **12 (Servicios de Apoyo)**, ya que el sitio no tendrá integradas tecnologías específicas de accesibilidad (punto 5), no tendrá opciones de voz (punto 6), ni vídeos (punto 7), y no incorporará documentos descargables (punto 10), y por lo tanto no hará falta que se examinen.

1.2.1 - HERRAMIENTAS

Las pruebas se realizarán principalmente mediante herramientas y extensiones de Chrome como:

- **Lighthouse**
- **Siteimprove**
- **taba11y**

- **Color Contrast Analyzer**
- **Live CSS**
- **HeadingsMap**
- **Screen Reader**

Además, aquellas comprobaciones no cubiertas por las extensiones se realizarán de forma manual.

Para facilitar el proceso, se seguirá un **manual de accesibilidad en desarrollo**, basado en lo aprendido dentro del equipo de trabajo y facilitado por una compañera de empresa.

1.2.2 - DOCUMENTACIÓN

Todo el proceso de pruebas quedará documentado en varios formatos:

- Un **documento Excel** que recogerá los resultados de accesibilidad con su puntuación.
- Dos documentos **.txt**:
    - Uno técnico, orientado al **programador**, con los fallos detectados y posibles soluciones.
    - Otro destinado a la **declaración de accesibilidad**, con una redacción más formal y orientada a informar al usuario final de las limitaciones detectadas.
- Carpetas con **capturas de pantalla** e **informes** generados por las herramientas utilizadas.

Esta documentación permitirá tener trazabilidad del trabajo realizado, además de servir como guía de mejora para futuras versiones del sitio.

1.2.3 - PUNTOS A ANALIZAR

Punto 9:

9.1.1. Alternativas de texto

9.1.1.1 Contenido no textual

9.1.2 Medios basados en el tiempo

9.1.2.1 Solo audio y solo vídeo (grabado) (Condicional)

9.1.2.2 Subtítulos (grabados) (Condicional)

9.1.2.3 Audiodescripción o Medio Alternativo (grabado) (Condicional)

9.1.2.5 Audiodescripción (grabada) (Condicional)

9.1.3 Adaptable

9.1.3.1 Información y relaciones (Condicional)

9.1.3.2 Secuencia significativa (Condicional)

9.1.3.3 Características sensoriales

9.1.3.4 Orientación

9.1.3.5 Identificación del propósito de la entrada

9.1.4 Distinguible

9.1.4.1 Uso del Color

9.1.4.2 Control de audio

9.1.4.3 Contraste (mínimo) (Condicional)

9.1.4.4 Cambio tamaño de texto

9.1.4.5 Imágenes de texto

9.1.4.10 Reajuste de texto

9.1.4.11 Contraste no textual

9.1.4.12 Espaciado de texto

9.1.4.13 Contenido señalado con el puntero o que tiene el foco (Condicional)

9.2.1 Teclado accesible

9.2.1.1 Teclado

9.2.1.2 Sin trampas con el foco del teclado

9.2.1.4 Atajos del teclado

9.2.2 Tiempo suficiente

9.4.1. Compatible

9.4.1.1. Procesamiento

9.4.1.2 Nombre, función, valor

9.4.1.3 Mensajes de estado

9.6. Requisitos de conformidad de las Pautas WCAG

Punto 11:

11.7 Preferencias de usuario

11.8.1 Tecnología de gestión de contenidos (Condicional)

11.8.2 Creación de contenidos accesibles (Condicional)

11.8.3 Preservación de la información de accesibilidad durante las transformaciones (Condicional)

11.8.4 Servicio de reparación (Condicional)

11.8.5 Plantillas (Condicional)

Punto 12:

12.1 Documentación del producto

12.1.1 Características de accesibilidad y compatibilidad

12.1.2 Documentación accesible

12.2 Servicios de soporte

12.2.2 Información sobre las características de accesibilidad y compatibilidad

12.2.3 Comunicación efectiva

12.2.4 Documentación accesible

COSTES/PRESUPUESTO

El principal coste ha sido el tiempo dedicado a la planificación, diseño, desarrollo y pruebas del ecommerce. Gracias a utilizar herramientas y frameworks de código

abierto como Laravel y Livewire, no ha sido necesario invertir en licencias de software, lo que ha reducido considerablemente los costes.

El entorno de trabajo se ha montado con XAMPP, que ofrece Apache, PHP y MySQL de forma gratuita, facilitando el desarrollo local sin requerir gastos adicionales en infraestructura.

Los recursos utilizados han sido principalmente mi equipo personal y acceso a internet, sin contratar servicios externos ni herramientas de pago.

En caso de llevar el proyecto a producción, se deberá considerar la contratación de un hosting y registro de dominio. Estos servicios, dependiendo del proveedor y las características contratadas, tendrían un coste mensual aproximado entre 5 y 10 euros para un plan básico adecuado al tamaño y funcionalidad del ecommerce. Además, se podría necesitar inversión en certificados SSL y mantenimiento continuo.

En conclusión, el proyecto ha tenido un coste inicial bajo o nulo desde el punto de vista económico, concentrándose la inversión principalmente en el tiempo y esfuerzo del desarrollo. Los costes futuros estarán relacionados con la puesta en marcha y operación en un entorno real.

CONCLUSIONES

BIBLIOGRAFÍA

GLOSARIO

ANEXOS

1.1 - PROTOTIPO DE LA WEB

Como parte del proceso de diseño previo al desarrollo, se ha creado un prototipo de baja fidelidad utilizando la herramienta **Balsamiq Mockups**. Este prototipo tiene como objetivo representar de forma visual la estructura general y la disposición de

los elementos clave de la tienda online, antes de su implementación final en Laravel + Livewire.

El diseño busca ofrecer una experiencia de usuario intuitiva, clara y accesible tanto para usuarios comunes como para administradores del sistema. Incluye pantallas como la página principal, listado de productos, detalle del producto, carrito de compras, formulario de registro/login, y panel de administración.

El prototipo completo se adjunta en el archivo **Prototipo_Ecommerce.pdf**.

Este documento ha servido de base para tomar decisiones durante el desarrollo, validar ideas y anticipar posibles mejoras en la interfaz antes de escribir el código.

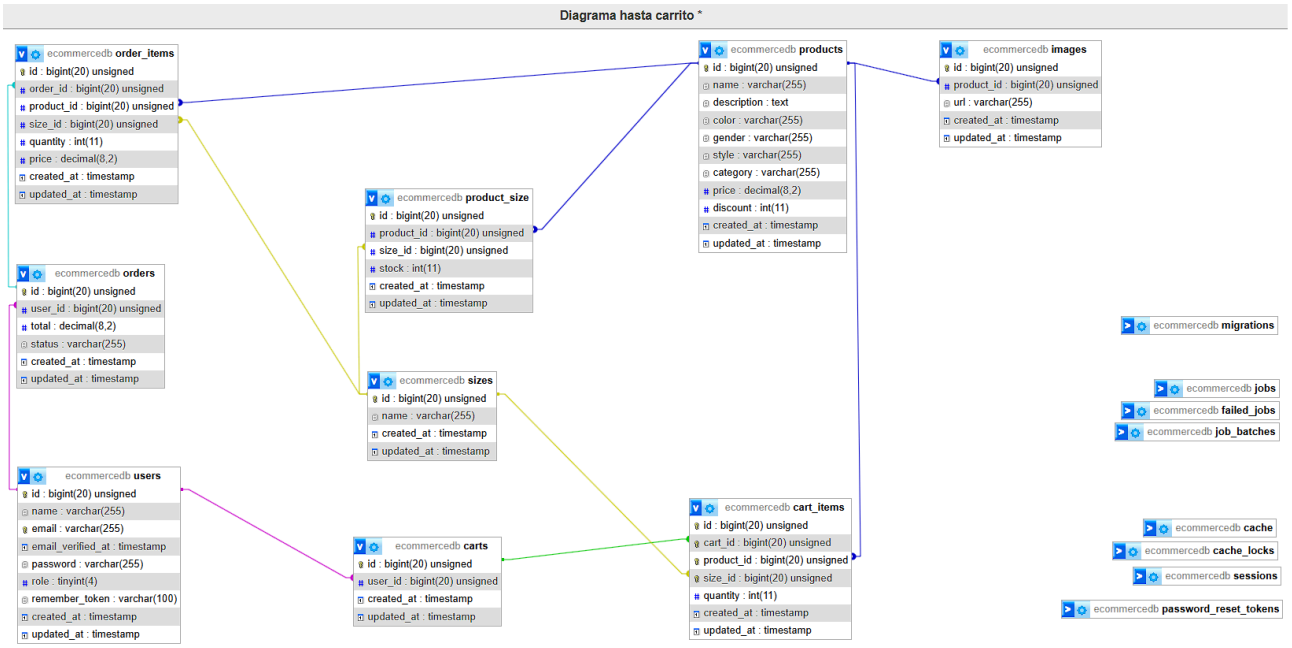
1.2 - BASE DE DATOS

Al crear un nuevo proyecto con Laravel, se generan automáticamente algunas tablas por defecto relacionadas con la autenticación y la gestión de usuarios. Sin embargo, para el desarrollo del ecommerce se han diseñado e implementado tablas adicionales que responden a las necesidades específicas del sistema, como la gestión de productos, tallas, imágenes, carrito, pedidos y sus relaciones.

A continuación, se presenta el modelo de base de datos utilizado, incluyendo las tablas principales y sus relaciones.

1.2.1 - DIAGRAMA RELACIONAL

El siguiente diagrama muestra la estructura relacional de la base de datos utilizada en el proyecto. En él se pueden observar las tablas principales del sistema, así como las relaciones entre ellas (uno a muchos y muchos a muchos).



Este esquema facilita la comprensión de la lógica de datos detrás del ecommerce, permitiendo identificar de forma clara cómo se vinculan los productos con sus imágenes, tallas disponibles, carritos de usuario, pedidos realizados, etc.

1.2.2 - GRAFO

El grafo de relaciones representa visualmente las conexiones entre las tablas de la base de datos desde una perspectiva más simplificada. Este enfoque se centra en mostrar cómo se enlazan los elementos del sistema, destacando las dependencias clave:

- users (**id**, name, email, email_verified_at, password, role, remember_token, created_at, updated_at)
- products (**id**, name, description, color, gender, style, category, price, discount, created_at, updated_at)
- images (**id**, product_id, url, created_at, updated_at)
- product_size (**id**, product_id, size_id, stock, created_at, updated_at)

- sizes (id, name, created_at, updated_at)
- carts (id, users_id, created_at, updated_at)
- cart_items (**id**, cart_id, product_id, size_id,quantity, created_at, updated_at)
- orders (id, users_id, total, status, created_at, updated_at)
- order_items (**id**, order_id, product_id, size_id, quantity, price, created_at, updated_at)

Este recurso es útil tanto para el desarrollo como para futuras tareas de mantenimiento, ya que ofrece una visión rápida y global de la estructura del proyecto.