

# Proyecto UT4 - Ferretería

DSW - 2DAW



## Integrantes del grupo:

- Raúl Rodríguez Martínez
- Gianfranco Lombardo López
- Iván Vega Hernández

# ÍNDICE:

<b>Introducción:</b> .....	<b>3</b>
<b>Características principales:</b> .....	<b>3</b>
<b>Base de datos (ferreteria):</b> .....	<b>4</b>
<b>Consultas</b> .....	<b>5</b>
<b>Archivo: conexiones.php</b> .....	<b>5</b>
<b>Archivo: consultas.php</b> .....	<b>7</b>
<b>Interfaces</b> .....	<b>9</b>
<b>Archivo: inicioSesion.php</b> .....	<b>9</b>
<b>Archivo: registro.php</b> .....	<b>11</b>
<b>Archivo: catalogo.php</b> .....	<b>13</b>
<b>Archivo: catalogoAdmin.php</b> .....	<b>16</b>
<b>Archivo: agregarProducto.php</b> .....	<b>18</b>
<b>Archivo: modificarProducto.php</b> .....	<b>20</b>
<b>Archivo: paginaIntermedia.php</b> .....	<b>23</b>
<b>Archivo: validaciones.php</b> .....	<b>33</b>
<b>Archivo: barraBusqueda.js</b> .....	<b>37</b>
<b>Archivo: carrito.js</b> .....	<b>40</b>

# **Documentación Completa del Proyecto de Ferretería**

## **Introducción:**

El Proyecto “Ferretería” es una plataforma web pensada para hacer más fácil y accesible la compra y venta de productos de ferretería. La idea es que los clientes puedan navegar por el catálogo, añadir productos a un carrito de compras y que los administradores gestionen el inventario de forma sencilla y eficiente.

El objetivo principal del proyecto es poder administrar una ferretería, ofreciendo a los clientes una forma rápida de encontrar lo que buscan y a los administradores herramientas para mantener todo organizado.

## **Características principales:**

Para los clientes:

- Registro e inicio de sesión.
- Búsqueda rápida de productos.
- Carrito de compras para gestionar sus pedidos.

Para los administradores:

- Opciones para agregar, modificar o eliminar productos.

Seguridad:

- Validación de datos y protección de la información de los usuarios.

Este proyecto está diseñado para optimizar las tareas diarias de una ferretería, simplificando procesos y reduciendo el trabajo manual. En este documento se detallan las funcionalidades y cómo se implementa cada una de ellas en el sistema.

# Base de datos (ferreteria):

Esta contiene las tablas necesarias para gestionar clientes, administradores y productos.

## Tablas principales:

- cliente y administrador:
  - Tienen estructuras similares, la diferencia es que representan roles distintos.
  - Incluyen campos como nombre, email, y pass para gestionar los usuarios.
  - email es único para evitar duplicados.

```
CREATE TABLE cliente (
    id INT AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    pass Varchar(20) not null,
    PRIMARY KEY (id)
);

CREATE TABLE administrador (
    id INT AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    pass Varchar(20) not null,
    PRIMARY KEY (id)
);
```

- productos:
  - Define productos con su nombre, precio, descripción y una columna para almacenar imágenes como binarios largos (LONGBLOB).

```
CREATE TABLE productos (
    id INT AUTO_INCREMENT,
    nombre VARCHAR(50) NOT NULL,
    precio DECIMAL(10, 2) NOT NULL,
    descripcion VARCHAR(200) NOT NULL,
    imagen LONGBLOB,
    PRIMARY KEY (id)
);
```

# Consultas

## Archivo: conexiones.php

Este archivo se encarga de gestionar la conexión a la base de datos MySQL para el proyecto. Contiene dos funciones principales:

### 1. conexion():

Esta función establece la conexión con la BBDD utilizando los datos de configuración:

- \$servername
- \$username
- \$password
- \$dbname

Luego se crea un objeto de conexión usando ***new mysqli***, y por último se verifica si la conexión fue exitosa

```
function conexion()
{
    // Configuración de la conexión
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "ferreteria";

    // Crear la conexión
    $conn = new mysqli($servername, $username, $password, $dbname);

    // Verificar si la conexión fue exitosa
    if ($conn->connect_error) {
        die("<p class='error'>Conexión fallida: " . $conn->connect_error . "</p>");
    } else {
        return $conn;
    }
}
```

## 2. apagar(\$conn):

Esta función recibe como parámetro una conexión activa.

Si recibe un objeto válido cierra la conexión con “`$conn->close()`”, en caso contrario muestra un mensaje indicando que no hay una conexión activa para cerrar.

```
function apagar($conn)
{
    if ($conn) {
        $conn->close(); // Cierra la conexión a la base de datos
    } else {
        echo "<p>No hay una conexión activa para cerrar.</p>";
    }
}
```

## Archivo: consultas.php

Este archivo contiene las funciones necesarias para interactuar con la base de datos. Estas funciones gestionan operaciones como creación, consulta, actualización y eliminación de datos relacionados con usuarios, productos y administradores.

---

### Funciones relacionadas con usuarios:

1. crearUsuario():

Permite registrar un nuevo usuario. Primero se verifica si el correo ya está registrado en la base de datos. Si no lo está, inserta los datos del nuevo usuario en la tabla cliente. Si el correo ya existe, se muestra un mensaje de error y se redirige al formulario de registro.

```
function crearUsuario($nombre, $email, $pass)
```

2. verificarExisteUsuario() y verificarExisteAdministrador():

Estas funciones verifican si un usuario o un administrador con un correo y contraseña específicos existen en la base de datos. Si se encuentra, retornan true; de lo contrario, retornan false.

```
function verificarExisteUsuario($email, $password)
```

```
function verificarExisteAdministrador($email, $password)
```

### Funciones relacionadas con la gestión de los productos:

1. verificarProductoPorNombre():

Comprueba si ya existe un producto con el nombre dado.

```
function verificarProductoPorNombre($nombre)
```

2. crearProducto():

Verifica si el producto ya existe, luego mueve la imagen al servidor y guarda el producto en la base de datos. Si algo sale mal, se muestra un error.

```
function crearProducto($nombre, $descripcion, $precio, $imagen)
```

3. modificarProducto():

Permite cambiar los datos de un producto, incluyendo su imagen. Si se proporciona una nueva imagen, reemplaza la anterior. De lo contrario, utiliza la imagen actual.

```
function modificarProducto($id, $nombre, $descripcion, $precio, $imagen = null)
```

4. eliminarProducto():

Elimina un producto de la base de datos según su ID.

```
function eliminarProducto($id)
```

5. obtenerProductosClientes():

Muestra los productos en un catálogo para que los clientes los vean y los agreguen al carrito.

```
function obtenerProductosClientes()
```

6. obtenerProductosAdmin():

Muestra los productos en un catálogo para los administradores, pero con opciones para modificarlos o eliminarlos.

```
function obtenerProductosAdmin()
```

7. obtenerProductoPorId():

Permite obtener un producto específico usando su ID, y retorna sus datos si lo encuentra.

```
function obtenerProductoPorId($id)
```

# Interfaces

## Archivo: inicioSesion.php

Este archivo genera un formulario para los usuarios de la aplicación, permitiendo tanto a clientes como a administradores iniciar sesión con su correo electrónico y contraseña, y acceder a las funcionalidades correspondientes de cada usuario.

---

### Propósitos del formulario:

- Autenticación de usuarios:

Permite a los usuarios iniciar sesión proporcionando sus credenciales (correo electrónico y contraseña).

- Identificación de roles:

Incluye una opción "**Continuar como administrador**" para que los administradores puedan iniciar sesión. Esta opción se gestiona con un checkbox.

- Redirección de datos:

Al presionar el botón "Iniciar Sesión", los datos se envían al servidor (paginaIntermedia.php) para que el sistema verifique si el usuario existe y si la contraseña es correcta.

- Enlace para registrarse:

Si el cliente no tiene cuenta, puede hacer clic en el enlace que lo llevará a la página de registro.

## Estructura del formulario:

```
<!-- Formulario para que los usuarios inicien sesión -->
<form class="login-form" action="../Verificaciones/paginaIntermedia.php" method="post">
    <h2 class="login-title">Iniciar sesión en Ferretería</h2>

    <!-- Campo oculto que indica la acción que debe realizarse cuando se envía el formulario -->
    <input type="hidden" name="accion" value="verificar_usuario">

    <div class="login">
        <input class="login-input" type="text" name="email" id="email" placeholder="Correo electrónico">
        <input class="login-input" type="password" name="password" id="password" placeholder="Contraseña">

        <!-- Opción para iniciar sesión como administrador -->
        <div class="admin">
            <input class="checkbox" type="checkbox" name="admin" id="admin">
            <label class="admin-check" for="admin">Continuar como administrador</label>
        </div>

        <!-- Botón para enviar el formulario -->
        <button class="login-button" type="submit">Iniciar Sesión</button>

        <!-- Enlace a la página de registro si el usuario no tiene cuenta -->
        <label class="link" onclick="window.location.href='../Interfaces/registro.php'">¿No tienes cuenta? Regístrate ahora</label>
    </div>
</form>
```

## Campos del formulario:

- Campo oculto: Incluye la acción **verificar\_usuario** para que “**páginaIntermedia.php**” identifique la acción.
- Correo electrónico: Captura el email del usuario.
- Contraseña: Sigue la clave del usuario.
- Checkbox: Permite a los administradores identificarse como tal.

## Archivo: registro.php

Este archivo genera un formulario de registro que permite a los usuarios crear una cuenta en la plataforma. Este formulario captura información básica y envía los datos al archivo **paginaIntermedia.php** para su procesamiento.

---

### Propósitos del formulario:

- Registro de nuevos usuarios:

Permite a los clientes registrarse proporcionando datos básicos.

- Validación y envío de datos:

Al hacer clic en el botón "Registrarse", el formulario se envía a la página "**paginaIntermedia.php**" donde el servidor procesa los datos para crear la cuenta.

- Enlace para iniciar sesión:

Si el usuario ya tiene una cuenta, puede hacer clic en un enlace que lo llevará a la página de inicio de sesión.

### Estructura del formulario:

```
<!-- Formulario para registrar a un nuevo usuario -->
<form class="register-form" action="../Verificaciones/paginaIntermedia.php" method="post">
    <h2 class="register-title">Crea una cuenta</h2>

    <!-- Campo oculto que indica la acción que debe realizarse cuando se envía el formulario -->
    <input type="hidden" name="accion" value="registrar_usuario">

    <!-- Contenedor para los campos del formulario -->
    <div class="register">
        <input class="register-input" type="text" name="name" id="name" placeholder="Nombre">
        <input class="register-input" type="text" name="email" id="email" placeholder="Correo electrónico">
        <input class="register-input" type="password" name="password" id="password" placeholder="Contraseña">
        <input class="register-input" type="password" name="confirm_password" id="confirm_password" placeholder="Confirmar contraseña">

        <!-- Botón para enviar el formulario -->
        <button class="register-button" type="submit">Registrarse</button>

        <!-- Enlace a la página de inicio de sesión si el usuario ya tiene cuenta -->
        <label class="link" onclick="window.location.href='../Interfaces/inicioSesion.php'">¿Ya tienes una cuenta? Inicia sesión</label>
    </div>
</form>
```

### Campos del formulario:

- Incluye un campo oculto con el valor **registrar\_usuario** para que el servidor sepa qué acción realizar cuando reciba los datos.
- El formulario solicita al usuario ingresar su *nombre, correo electrónico, contraseña y confirmación de contraseña*.
- El campo "**Confirmar contraseña**" es importante para asegurarse de que el usuario haya escrito la misma contraseña en ambos campos de la contraseña.

## **Archivo: catalogo.php**

Este archivo es la interfaz principal para que los clientes visualicen el catálogo de productos de la tienda. Utiliza funciones de PHP para generar dinámicamente los productos disponibles y ofrece una interacción mejorada con JavaScript para funcionalidades como el carrito de compras y la búsqueda.

---

### **Propósitos del archivo:**

- Mostrar una lista dinámica de productos disponibles en la tienda.
- Facilitar la búsqueda de productos en tiempo real mediante una barra de búsqueda.
- Integrar un carrito de compras interactivo para gestionar los productos seleccionados.
- Permitir que los clientes cierren sesión de forma sencilla.

### **Estructura principal:**

#### 8. Conexión a la base de datos:

En el archivo se incluye un archivo externo que contiene las funciones necesarias para consultar la base de datos. Esto asegura que la función “**obtenerProductosClientes()**” esté disponible para generar dinámicamente la lista de productos:

```
<?php  
  
// Incluimos el archivo que contiene las funciones para consultar la base de datos  
require_once '../Consultas/consultas.php';  
  
?>
```

## 9. Estructura del catálogo:

En la parte superior de la página, tenemos un encabezado con el título "**Catálogo de Productos**", una barra de búsqueda que permite a los clientes buscar productos dentro del catálogo y dos botones, uno para ver el carrito de compras y otro para cerrar sesión.

```
<!-- Encabezado con el nombre del catálogo y las opciones de búsqueda y navegación -->
<header>
  <div class="menu">
    <h1 class="menu-title">Catálogo de Productos</h1>

    <!-- Barra de búsqueda para que los clientes busquen productos -->
    <div class="menu-search">
      <input type="text" placeholder="Buscar productos..." id="searchInput" aria-label="Buscar producto">
    </div>

    <!-- Botones de navegación, uno para ver el carrito y otro para cerrar sesión -->
    <div class="menu-buttons">
      <button type="button" id="view-cart">Ver carrito</button>
      <button type="button" onclick="window.location.href='../Interfaces/inicioSesion.php'">Cerrar sesión</button>
    </div>
  </div>
</header>
```

## 10. Modal del carrito de compras:

Incluye una ventana emergente que se muestra cuando el cliente hace clic en "**Ver carrito**". Este cuenta con la lista de productos añadidos al carrito, el total de la compra, y opciones para cerrar el modal o vaciar el carrito.

```
<!-- Modal para mostrar el carrito de compras -->
<div class="modal" id="cartModal">
  <div class="modal-content">
    <h2>Carrito de Compras</h2>

    <!-- Lista de productos que el usuario ha añadido al carrito -->
    <ul id="cartItems" class="cart-items"></ul>
    <div class="cart-footer">
      <div class="total-price" id="totalPrice">Total: 0€</div>
      <button id="closeModal" class="close-btn">Cerrar</button>
      <button id="emptyCartButton" class="empty-cart">Vaciar Carrito</button>
    </div>
  </div>
</div>
```

## 11. Consulta de Productos:

Se llama a la función “**obtenerProductosClientes()**” para obtener el listado de productos desde la base de datos y mostrarlos en la página:

```
<?php  
|     obtenerProductosClientes();  
?>
```

## 12. Scripts adicionales:

Estos scripts se incluyen para manejar la funcionalidad del carrito y para realizar la búsqueda de productos en el catálogo, respectivamente:

```
<!-- Enlazamos los scripts JavaScript para el carrito y la barra de búsqueda -->  
<script src="../../scripts/carrito.js"></script>  
<script src="../../scripts/barraBusqueda.js"></script>
```

## Archivo: catalogoAdmin.php

Este archivo es similar al catálogo de productos que ven los clientes, pero en este se incluyen funcionalidades para que los administradores gestionen los productos fácilmente (buscar, agregar, modificar y eliminar productos del catálogo).

---

### Propósitos del archivo:

- Ver todos los productos disponibles en el catálogo.
- Buscar productos mediante una barra de búsqueda.
- Agregar nuevos productos al catálogo.
- Modificar productos del catálogo.
- Eliminar productos del catálogo.
- Cerrar sesión de manera segura.

### Estructura principal:

#### 1. Conexión a la base de datos:

El archivo incluye un archivo externo que contiene las funciones necesarias para consultar la base de datos. Esto asegura que la función “**obtenerProductosClientes()**” esté disponible para generar dinámicamente la lista de productos:

```
<?php  
  
// Incluimos el archivo que contiene las funciones para consultar la base de datos  
require_once '../Consultas/consultas.php';  
  
?>
```

#### 2. Estructura del Catálogo para Administradores:

En la parte superior de la página, tenemos un encabezado con el título “**Catálogo de Productos**” y una barra de búsqueda que permite a los administradores buscar productos dentro del catálogo.

También cuenta con una etiqueta (**Eres Admin**) que indica que el usuario está logueado como administrador, y con dos botones, uno para agregar un nuevo producto al catálogo y otro para cerrar sesión.

```
<!-- Encabezado para el catálogo de productos, personalizado para administradores -->
<header class="header-admin">
  <div class="menu-admin">
    <h1 class="menu-admin-title">Catálogo de Productos</h1>

    <!-- Barra de búsqueda para que los administradores busquen productos -->
    <div class="menu-admin-search">
      <input type="text" placeholder="Buscar productos..." id="searchInput" aria-label="Buscar producto">
    </div>

    <!-- Indicador de que el usuario es administrador -->
    <label for="admin">(Eres Admin)</label>

    <!-- Botones de navegación para los administradores, uno para añadir un producto nuevo al catálogo y otro para cerrar sesión -->
    <div class="menu-admin-buttons">
      <button type="button" onclick="window.location.href='../Interfaces/agregarProducto.php'">Agregar producto</button>
      <button type="button" onclick="window.location.href='../Interfaces/inicioSesion.php'">Cerrar sesión</button>
    </div>
  </div>
</header>
```

### 3. Consulta de Productos:

Se llama a la función “**obtenerProductosAdmin()**” para obtener y mostrar los productos en el catálogo, similar a cómo se hace en el catálogo para los clientes. La diferencia es que en este catálogo, la tarjeta de cada producto tendrá las opciones para modificarlo o eliminarlo de la base de datos.

```
<?php
  // Llamamos a la función que obtiene y muestra los productos en el catálogo
  obtenerProductosAdmin();
?>
```

### 4. Script adicional:

Este script se incluye para realizar la búsqueda de productos en el catálogo:

```
<!-- Enlazamos el script JavaScript para la barra de búsqueda -->
<script src="../../scripts/barraBusqueda.js"></script>
```

## Archivo: agregarProducto.php

Este archivo genera un formulario que permite a los administradores añadir nuevos productos al catálogo. Los datos del formulario se envían al archivo **paginaIntermedia.php** para ser procesados.

---

### Propósitos del formulario:

- Agregar nuevos productos:

Facilita a los administradores ingresar datos (nombre, descripción, precio e imagen) de nuevos productos al catálogo de la tienda.

- Gestionar el inventario:

Se les proporciona a los administradores una interfaz sencilla para que mantengan actualizado el inventario.

- Mejorar la experiencia de administración:

Ofrece un formulario simple y efectivo para agregar productos.

- Asegurar la correcta carga de información:

Con campos obligatorios y validaciones, se asegura que los productos se ingresen correctamente (incluyendo imágenes).

### Estructura del formulario

```
<!-- Formulario para agregar un nuevo producto al catálogo -->
<form class="product-form" action="../Verificaciones/paginaIntermedia.php" method="post" enctype="multipart/form-data">
    <div class="product-container">
        <h2 class="product-title">Agregar Producto</h2>

        <!-- Campo oculto para enviar la acción de agregar un producto -->
        <input type="hidden" name="accion" value="agregar_producto">

        <label for="product-name" class="product-label">Nombre del Producto</label>
        <input type="text" id="product-name" name="product_name" class="product-input" placeholder="Nombre del producto">

        <label for="description" class="product-label">Descripción</label>
        <textarea id="description" name="description" class="product-textarea" placeholder="Descripción del producto" rows="4"></textarea>

        <label for="price" class="product-label">Precio</label>
        <input type="number" id="price" name="price" class="product-input" placeholder="Precio" min="0" step="0.01">

        <!-- Etiqueta y campo para cargar una imagen del producto -->
        <label for="image" class="product-label">Imagen del Producto</label>
        <input type="file" id="image" name="image" class="product-file" accept="image/*">

        <!-- Botón para enviar el formulario y agregar el producto -->
        <button type="submit" class="product-button">Agregar Producto</button>
    </div>
</form>
```

- En este formulario se utiliza el atributo **enctype="multipart/form-data"** ya que es necesario para permitir la carga de archivos, como las imágenes del producto.
- El formulario tiene el botón "Aregar Producto" que al clicar en él, los datos se envían al archivo "**paginaIntermedia.php**" para ser procesados y almacenados.

#### **Campos del formulario:**

- Incluye un campo oculto con el valor **agregar\_producto** para que el servidor sepa qué acción realizar cuando reciba los datos.
- El formulario solicita al administrador ingresar el nombre del producto, descripción, precio, y una imagen.
- Para la imagen del producto utilizamos un campo de tipo file que permite al administrador subir una imagen del producto. Solo se aceptan imágenes, por eso usamos el atributo **accept="image/\*"**.

## **Archivo: modificarProducto.php**

Este archivo está diseñado para permitir a los administradores modificar los datos de un producto existente. La información del producto es precargada en el formulario desde la base de datos, y los cambios se envían para ser procesados.

---

### **Propósitos del formulario:**

- Cargar los datos de un producto específico desde la base de datos, utilizando un ID proporcionado, para permitir su modificación.
- Mostrar un formulario pre-rellenado con la información actual del producto (nombre, descripción, precio e imagen).
- Permitir modificar y actualizar los datos del producto, incluyendo la imagen.
- Enviar los cambios al servidor mediante un formulario que procesa los datos en “*páginaIntermedia.php*”.
- Gestionar errores y redirecciones en caso de que el producto no exista o el ID sea inválido.

### **Estructura principal:**

#### 1. Conexión a la base de datos:

En el archivo se incluye dos archivos externos que contienen tanto las funciones necesarias para consultar la base de datos, como la lógica para manejar solicitudes o validaciones adicionales.

```
// Incluir la conexión y las funciones necesarias
require_once '../Consultas/consultas.php';
require_once '../Verificaciones/páginaIntermedia.php';
```

## 2. Verificación del ID del Producto:

En este archivo, primero se verifica usando el método GET, si se ha pasado un id válido a través de la URL. Si el ID es válido, se convierte a número entero para evitar problemas de seguridad.

Luego, se obtiene el producto correspondiente desde la base de datos usando la función “**obtenerProductoPorId(\$id)**”.

Si el producto no se encuentra, se redirige a la página de catálogo de administración con un mensaje de error.

```
// Verificar si se pasó el ID por GET
if (isset($_GET['id']) && is_numeric($_GET['id'])) {
    $id = intval($_GET['id']); // Convertir el ID a un número entero

    // Obtener el producto desde la base de datos
    $producto = obtenerProductoPorId($id);

    if ($producto) {
        // Producto encontrado, ahora rellenamos el formulario con los datos del producto
        $nombre = htmlspecialchars($producto['nombre']);
        $descripcion = htmlspecialchars($producto['descripcion']);
        $precio = htmlspecialchars($producto['precio']);
        $imagen = htmlspecialchars($producto['imagen']);
    } else {
        // Si el producto no se encuentra, redirigir a la página de error
        header("Location: ../Interfaces/catalogoAdmin.php?error=Producto+no+encontrado.");
        exit;
    }
} else {
    // Si no se pasa un ID válido, redirigir a la página de error
    header("Location: ../Interfaces/catalogoAdmin.php?error=ID+no+especificado+o+inválido.");
    exit;
}

?>
```

## 3. Formulario de modificación del producto:

Si el producto se encuentra en la base de datos, se cargan sus datos en el formulario. Esto permite al administrador ver los valores actuales y modificar solamente los que deseé.

Se incluyen campos ocultos para enviar el ID del producto, la acción de “**confirmar\_modificación**”, y la imagen actual, al archivo “**paginaIntermedia.php**”.

```
<!-- Formulario de modificación del producto -->
<form class="product-form" action="../Verificaciones/paginaIntermedia.php" method="post" enctype="multipart/form-data">
    <div class="product-container">
        <h2 class="product-title">Modificar Producto</h2>

        <!-- Campos ocultos para enviar información adicional del producto al servidor -->
        <input type="hidden" name="accion" value="confirmar_modificacion">
        <input type="hidden" name="id" value="= $producto['id'] ?&gt;"&gt; &lt;!-- Pasamos el ID del producto al formulario --&gt;
        &lt;input type="hidden" name="current_image" value="<?= $producto['imagen'] ?&gt;"&gt; &lt;!-- Imagen actual --&gt;</pre
```

#### 4. Manejo de la imagen:

Si el producto tiene una imagen actual, esta se muestra en el formulario, permitiendo que el administrador decida si reemplazarla o no. Si se elige una nueva imagen, se subirá a la base de datos al enviar el formulario.

```
<!-- Muestra la imagen actual si existe -->
<?php if ($producto['imagen']): ?>
    
<?php endif; ?>
```

## **Archivo: paginaIntermedia.php**

Este archivo procesa las diversas acciones relacionadas con usuarios y productos enviadas mediante formularios (método POST). Estas acciones incluyen registrar usuarios, iniciar sesión, agregar, modificar y eliminar productos y validar datos.

Este archivo es muy importante para manejar la lógica entre las interfaces de usuario y las operaciones en la base de datos.

---

### **Propósito del archivo:**

- **Centralizar la lógica:**

Se encarga de manejar todo lo relacionado con los usuarios y los productos desde un solo lugar, evitando duplicar código en otros archivos.

- **Validar datos:**

Antes de hacer cualquier operación, se asegura de que los datos enviados desde los formularios sean correctos, como el formato del correo o que el precio sea un número válido.

- **Gestión de usuarios:**

Maneja el registro de nuevos usuarios, la verificación al iniciar sesión y el acceso tanto para usuarios normales como administradores.

- **Manejo de productos:**

Controla las operaciones principales del catálogo, como agregar, modificar o eliminar productos.

- **Redirección y mensajes:**

Luego de procesar las acciones, redirige a las páginas correspondientes con mensajes claros de éxito o error.

- **Conexión con la base de datos:**

Sirve para conectar los formularios del usuario con las funciones que interactúan directamente con la base de datos.

## Estructura principal:

### 1. Conexión a la base de datos:

En el archivo se incluyen los archivos necesarios que contienen tanto las funciones necesarias para consultar la base de datos, como las validaciones de los datos y la conexión a la base de datos.

```
// Incluir archivos necesarios para consultas, conexiones y validaciones
require_once '../Consultas/consultas.php';
require_once '../Consultas/conexiones.php';
require_once 'validaciones.php';
```

### 2. Recepción de datos:

El archivo se asegura de procesar solamente las solicitudes enviadas mediante el método POST, lo cual es esencial para la seguridad y control.

```
// Verificar si la solicitud es POST para procesar acciones específicas
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
```

### 3. Identificación de la acción:

Se utiliza el campo “**acción**” enviado en el formulario para determinar qué proceso se debe ejecutar.

```
$accion = $_POST['accion'] ?? null; // Obtener la acción enviada por el formulario

// Procesar la acción según su valor
switch ($accion) {
    case 'registrar_usuario':
```

### 4. Procesamiento de cada acción:

Para cada acción, el archivo realiza las siguientes tareas:

- Validación de datos: Se asegura de que los datos enviados sean correctos antes de continuar.
- Llamada a funciones específicas: Utiliza funciones predefinidas para realizar tareas como crear usuarios o productos.
- Redirección: Después de procesar la acción, redirige al usuario a la página adecuada con mensajes claros de éxito o error.

```

// Procesar la acción según su valor
switch ($accion) {
    case 'registrar_usuario':
        // Obtener los datos del usuario
        $nombre = $_POST['name'] ?? null;
        $email = $_POST['email'] ?? null;
        $password = $_POST['password'] ?? null;
        $confirmPassword = $_POST['confirm_password'] ?? null;
        $errores = [];

        // Validar el nombre del usuario
        if (validarDato('string', $nombre) !== true) {
            $errores[] = "El nombre es inválido.";
        }

        // Validar el correo electrónico
        if (validarDato('email', $email) !== true) {
            $errores[] = "El correo es inválido."; // Agrega el mensaje de error devuelto por la validación
        }

        // Validar la contraseña
        if (validarDato('string', $password) !== true) {
            $errores[] = "La contraseña es inválida.";
        }

        // Valido que las contraseñas sean iguales
        if ($password != $confirmPassword) {
            $errores[] = "Las contraseñas no coinciden";
        }

        // Si hay errores, redirigir de vuelta con los errores
        if (!empty($errores)) {
            // Redirigir a la página de registro con los errores en la URL
            header("Location: ../Interfaces/registro.php?errores=" . urlencode(implode(", ", $errores)));
            exit;
        }

        // Crear el usuario
        $usuarioCreado = crearUsuario($nombre, $email, $password);
        exit;
}

```

## 5. Manejo de Casos por Defecto:

Si no se reconoce la acción, redirige a una página de error para evitar comportamientos inesperados.

```

default:
    // Si no se reconoce la acción, redirigir a una página de error
    header("Location: error.php");
    break;

```

## 6. Estructura de Validaciones y Mensajes:

En cada acción, si hay errores en los datos, se redirige al usuario al formulario correspondiente con los mensajes de error.

```

// Si hay errores, redirigir de vuelta con los errores
if (!empty($errores)) {
    // Redirigir a la página de registro con los errores en la URL
    header("Location: ../Interfaces/registro.php?errores=" . urlencode(implode(", ", $errores)));
    exit;
}

```

### Casos del switch:

- 'registrar\_usuario':

**Propósito:** Procesar el registro de nuevos usuarios.

#### Captura de datos del formulario:

- Recoge name, email, password y `confirm_password()` del formulario enviado por el usuario.
- Inicializa un array \$errores para almacenar posibles errores.

#### Validación de datos:

- Verifica que el nombre, el correo electrónico y la contraseña sean válidos mediante la función `validarDato()`.
- Asegura que las contraseñas coincidan.

#### Manejo de errores:

- Si hay errores, los concatena y redirige al formulario de registro con los mensajes.

#### Creación del usuario:

- Si todo es correcto, llama a `crearUsuario()` para registrarlo en la base de datos.

```

case 'registrar_usuario':
    // Obtener los datos del usuario
    $nombre = $_POST['name'] ?? null;
    $email = $_POST['email'] ?? null;
    $password = $_POST['password'] ?? null;
    $confirmPassword = $_POST['confirm_password'] ?? null;
    $errores = [];

    // Validar el nombre del usuario
    if (validarDatos('string', $nombre) !== true) {
        $errores[] = "El nombre es inválido.";
    }

    // Validar el correo electrónico
    if (validarDatos('email', $email) !== true) {
        $errores[] = "El correo es inválido." // Agrega el mensaje de error devuelto por la validación
    }

    // Validar la contraseña
    if (validarDatos('string', $password) !== true) {
        $errores[] = "La contraseña es inválida.";
    }

    // Valido que las contraseñas sean iguales
    if ($password != $confirmPassword) {
        $errores[] = "Las contraseñas no coinciden";
    }

    // Si hay errores, redirigir de vuelta con los errores
    if (!empty($errores)) {
        // Redirigir a la página de registro con los errores en la URL
        header("Location: ../Interfaces/registro.php?errores=" . urlencode(implode(", ", $errores)));
        exit;
    }

    // Crear el usuario
    $usuarioCreado = crearUsuario($nombre, $email, $password);
    exit;

```

- 'verificar\_usuario':

#### Propósito:

- Validar las credenciales del usuario o administrador e iniciar sesión.

#### Captura de datos del formulario:

- Recoge email, password y si el checkbox de admin está activado.

#### Verificación de usuario normal o administrador:

- Si el checkbox no está activado, verifica al usuario con `verificarExisteUsuario()`.
- Si está activado, verifica al administrador con `verificarExisteAdministrador()`.

### Redirección:

- Usuarios normales se redirigen al catálogo general.
- Administradores se redirigen al catálogo administrativo.
- Si las credenciales son inválidas, redirige al formulario de inicio de sesión con un mensaje de error.

```
case 'verificar_usuario':
    // Captura Los datos del formulario
    $email = $_POST['email'] ?? null;
    $password = $_POST['password'] ?? null;
    $checkbox = isset($_POST['admin']);

    // Validar si el correo existe
    if ($checkbox !== true) {
        if (verificarExisteUsuario($email, $password)) {
            // Aquí puedes verificar la contraseña o redirigir a otra página
            header("Location: ../Interfaces/catalogo.php");
            exit();
        } else {
            // Si el usuario no existe
            header("Location: ../Interfaces/inicioSesion.php?error=Credenciales+inválidas");
        }
    } elseif ($checkbox == true) {
        if (verificarExisteAdministrador($email, $password)) {
            // Aquí puedes verificar la contraseña o redirigir a otra página
            header("Location: ../Interfaces/catalogoAdmin.php");
            exit();
        } else {
            // Si el usuario no existe
            header("Location: ../Interfaces/inicioSesion.php?error=Credenciales+inválidas");
        }
    }
    exit;
```

- 'agregar\_producto':

### Propósito:

- Agregar un nuevo producto al sistema.

### Captura de datos del formulario:

- Obtiene el nombre, descripción, precio y archivo de imagen del formulario.

### Validaciones:

- Verifica que el nombre, la descripción y el precio sean válidos.
- Valida que la imagen sea correcta y se haya cargado correctamente.

### **Manejo de errores:**

- Si hay errores, redirige al formulario de agregar producto con mensajes detallados.

### **Creación del producto:**

- Si los datos son válidos, llama a **crearProducto()** para almacenarlo en la base de datos.

```
case 'agregar_producto':
    // Validar los campos del producto
    $nombre = $_POST['product_name'] ?? null;
    $descripcion = $_POST['description'] ?? null;
    $precio = $_POST['price'] ?? null;
    $imagen = $_FILES['image'] ?? null;
    $errores = [];

    // Validación del nombre del producto
    if (validarDato('string', $nombre) !== true) {
        $errores[] = "El nombre del producto es inválido.";
    }

    // Validación de la descripción
    if (validarDato('string', $descripcion) !== true) {
        $errores[] = "La descripción del producto es inválida.";
    }

    // Validación del precio
    if (validarDato('numero', $precio) !== true) {
        $errores[] = "El precio debe ser un número positivo.";
    }

    // Validación de la imagen (solo si se ha subido una)
    if ($imagen && !validarImagen($imagen)) {
        $errores[] = "La imagen es inválida o no se subió correctamente.";
    }

    // Si hay errores, redirigir de vuelta al formulario con los errores
    if (!empty($errores)) {
        // Redirigir a agregarProducto.php con los errores en la URL
        header("Location: ../Interfaces/agregarProducto.php?errores=" . urlencode(implode(", ", $errores)));
        exit; // Asegurarse de que el script no continúe
    }

    // Crear el producto si no hay errores
    $productoCreado = crearProducto($nombre, $descripcion, $precio, $imagen);
    exit;
```

- 'eliminar\_producto':

#### Propósito:

- Eliminar un producto existente del catálogo.

#### Captura del ID del producto:

- Obtiene el ID enviado por el formulario.

#### Validación del ID:

- Verifica que el ID sea un número válido.

#### Eliminación del producto:

- Llama a **eliminarProducto()** para eliminarlo de la base de datos.

#### Redirección:

- Si la eliminación es exitosa, redirige al catálogo del administrador con un mensaje de éxito.
- Si falla, redirige con un mensaje de error.

```
case 'eliminar_producto':
    // Obtener el ID del producto
    $id = $_POST['id'] ?? null;

    // Verificar si el ID es válido
    if (!validarData('numero', $id)) {
        header("Location: ../Interfaces/catalogoAdmin.php?error=ID+de+producto+inválido.");
        exit;
    }

    // Eliminar el producto usando la función que creamos
    $productoEliminado = eliminarProducto($id);

    if ($productoEliminado) {
        // Redirigir al catálogo con un mensaje de éxito
        header("Location: ../Interfaces/catalogoAdmin.php?exito=Producto+eliminado+con+éxito.");
        exit;
    } else {
        // Redirigir con mensaje de error si no se pudo eliminar el producto
        header("Location: ../Interfaces/catalogoAdmin.php?error=Hubo+un+error+al+eliminar+el+producto.");
        exit;
    }
}
```

- 'modificar\_producto':

#### Propósito:

- Redirigir al formulario de modificación de un producto existente.

#### Captura del ID del producto:

- Obtiene el ID enviado por el formulario.

#### Redirección:

- Redirige al formulario de modificación de la interfaz “modificarProducto.php” con el ID en la URL.
- Si no se envió un ID válido, redirige al catálogo administrativo con un mensaje de error.

```
case 'modificar_producto':
    if (isset($_POST['id'])) {
        $id = intval($_POST['id']); // Asegúrate de convertir el ID a entero

        // Redirige a la interfaz de modificación con el ID del producto
        header("Location: ../Interfaces/modificarProducto.php?id=$id");
        exit;
    } else {
        header("Location: ../Interfaces/catalogoAdmin.php?error=ID+no+especificado+para+modificación.");
        exit;
    }
    break;
```

- 'confirmar\_modificacion':

#### Propósito:

- Guardar los cambios en un producto existente.

#### Captura de datos:

- Obtiene el ID, nombre, descripción, precio, y la imagen actual del producto o la imagen nueva (si existe).

#### Validaciones:

- Verifica que el nombre, la descripción y el precio sean válidos.
- Valida que la nueva imagen sea correcta.

#### Manejo de errores:

- Si hay errores, redirige al formulario de modificación con mensajes de error.

## **Modificación del producto:**

- Si no se subió una nueva imagen, mantiene la actual.
- Llama a **modificarProducto()** para guardar los cambios en la base de datos.

## **Redirección:**

- Si la modificación es exitosa, redirige al catálogo administrativo con un mensaje de éxito.
- Si falla, redirige al formulario de modificación con un mensaje de error.

```
case 'confirmar_modificacion':
    if (isset($_POST['id'])) {
        $id = intval($_POST['id']); // Asegúrate de convertir el ID a entero
        $nombre = $_POST['product_name'] ?? null;
        $descripcion = $_POST['description'] ?? null;
        $precio = $_POST['price'] ?? null;
        $imagen = $_FILES['image'] ?? null;

        // Obtener la imagen actual si no se envió una nueva
        $imagenActual = $_POST['current_image'] ?? null;

        // Validar los datos del producto
        $errores = [];

        if (validarDatos('string', $nombre) !== true) {
            $errores[] = "El nombre del producto es inválido.";
        }

        if (validarDatos('string', $descripcion) !== true) {
            $errores[] = "La descripción del producto es inválida.";
        }

        if (validarDatos('numero', $precio) !== true) {
            $errores[] = "El precio debe ser un número positivo.";
        }

        // Validación de la imagen (solo si se ha subido una nueva)
        if ($imagen && !validarImagen($imagen)) {
            $errores[] = "La imagen es inválida o no se subió correctamente.";
        }

        // Si hay errores, redirigir de vuelta al formulario con los errores
        if (!empty($errores)) {
            // Redirigir a modificarProducto.php con los errores en la URL
            header("Location: ../Interfaces/modificarProducto.php?id=$id&errores=". urlencode(implode(", ", $errores)));
            exit; // Asegurarse de que el script no continúa
        }

        // Si no hay errores, procesar la actualización del producto
        if ($imagen && $imagen['error'] == 0) {
            // Se proporciona una nueva imagen
            $productoModificado = modificarProducto($id, $nombre, $descripcion, $precio, $imagen);
        } else {
            // No se proporciona una nueva imagen, se mantiene la actual
            $productoModificado = modificarProducto($id, $nombre, $descripcion, $precio, $imagenActual);
        }

        // Verificar si el producto fue modificado exitosamente
        if ($productoModificado) {
            // Redirigir a la página de éxito (catalogoAdmin.php en este caso)
            header("Location: ../Interfaces/catalogoAdmin.php?exito=Producto+modificado+exitosamente.");
        } else {
            // Si algo falló en la modificación, redirigir de vuelta a la página de modificar producto con un mensaje de error
            header("Location: ../Interfaces/modificarProducto.php?id=$id&error=Hubo+un+error+al+modificar+el+producto.");
        }
        exit; // Asegurarse de que el script no continúa
    } else {
        header("Location: ../Interfaces/catalogoAdmin.php?error=ID+no+especificado+para+modificación.");
        exit;
    }
}
break;
```

## Archivo: validaciones.php

El archivo validaciones.php proporciona funciones para validar datos y archivos cargados en la aplicación. Estas funciones aseguran que los datos ingresados cumplan con los formatos requeridos antes de ser procesados, protegiendo la integridad del sistema y la base de datos.

---

### Propósito del archivo

- Validación de datos:  
Verificar que los datos ingresados por los usuarios (como correos, números y contraseñas) sean válidos.
- Validación de archivos:  
Asegurar que los archivos subidos (imágenes) cumplan con los requisitos de tipo y tamaño.

### Funciones principales:

1. **validarDatos()**: Esta función, mediante un switch, permite validar diferentes tipos de datos como teléfonos, correos electrónicos, contraseñas, números, URLs, fechas, etc, con reglas específicas para cada tipo.

#### Parámetros que se le pasan a esta función:

- **\$tipo**: El tipo de dato a validar (ej. teléfono, email, string, numero, etc.).
- **\$dato**: El dato que se desea validar.

#### Validaciones:

- **Teléfono**: Acepta números de 9 o 10 dígitos.

```
case 'telefono':  
    // Validación de teléfono (9 o 10 dígitos)  
    if (preg_match('/^\d{9,10}$/', $dato)) {  
        return true;  
    } else {  
        return "Formato de número de teléfono incorrecto.";  
    }
```

- **Correo electrónico:** Sanitiza y valida el formato del correo electrónico.

```
// Sanitizar el correo antes de validar
$dato = filter_var($dato, FILTER_SANITIZE_EMAIL);

// Validar formato del correo electrónico
if (filter_var($dato, FILTER_VALIDATE_EMAIL)) {
    return true;
} else {
    return "Formato de correo electrónico incorrecto.";
}
```

- **Texto (string):** Elimina etiquetas HTML y espacios vacíos, y verifica que el campo no esté vacío.

```
// Elimina etiquetas HTML y espacios en blanco
$dato = trim(strip_tags($dato));
if (!empty($dato)) {
    return true;
} else {
    return "El campo de texto no puede estar vacío o solo contener espacios.";
}
```

- **Número (número):** Acepta solo números positivos.

```
// Comprueba si es un número y si es positivo
if (is_numeric($dato) && $dato >= 0) {
    return true;
} else {
    return "Debe ser un número positivo.";
}
```

- **Contraseña (password):** Requiere al menos 8 caracteres, una letra y un número.

```
// Validación de contraseña (mínimo 8 caracteres, al menos 1 letra y 1 número)
if (preg_match('/^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/', $dato)) {
    return true;
} else {
    return "La contraseña debe tener al menos 8 caracteres, incluyendo una letra y un número.";
}
```

- **URL (url)**: Se valida si tiene el formato correcto de una URL.

```
// Validación de URL
if (filter_var($dato, FILTER_VALIDATE_URL)) {
    return true;
} else {
    return "URL no válida.";
}
```

- **Fecha (fecha)**: Se valida si el dato tiene un formato de fecha válido, específicamente en el formato (YYYY-MM-DD).  
Devuelve true si el dato es válido. En caso contrario devuelve un mensaje de error descriptivo.

```
// Validación de formato de fecha (YYYY-MM-DD)
$fechaValida = DateTime::createFromFormat('Y-m-d', $dato);
if ($fechaValida && $fechaValida->format('Y-m-d') === $dato) {
    return true;
} else {
    return "Formato de fecha incorrecto. Use YYYY-MM-DD.";
}
```

## 2. ***validarImagen()***:

Esta función valida las imágenes subidas, comprobando que el tipo de archivo y el tamaño sean correctos para evitar errores al guardar las imágenes en el sistema.

Parámetro que se le pasa a esta función:

- **\$imagen**: Información del archivo subido (`$_FILES['nombre']`).

Validaciones:

- **Errores de carga**: Se verifica si hubo algún error al subir el archivo. Si el valor es distinto de 0, significa que hubo un problema durante la carga del archivo

```
// Verificar si se ha subido un archivo
if ($imagen['error'] !== 0) {
    return false; // Error en la carga
}
```

- **Tipo de archivo:** Se verifica el tipo de archivo de la imagen subida. Sólo se aceptan imágenes en formatos JPEG, PNG y GIF.

```
// Verificar tipo de archivo
$tipoImagen = mime_content_type($imagen['tmp_name']);
if (!in_array($tipoImagen, ['image/jpeg', 'image/png', 'image/gif'])) {
    return false; // Si no es una imagen válida
}
```

- **Tamaño de archivo:** Se verifica si el tamaño del archivo es mayor que el límite permitido. Devuelve true si la imagen es válida o, en caso contrario, un mensaje de error.

```
// Verificar tamaño (por ejemplo, no mayor a 2MB)
if ($imagen['size'] > 2 * 1024 * 1024) { // 2MB
    return false; // Imagen demasiado grande
}
```

## Archivo: barraBusqueda.js

Este script implementa la funcionalidad de búsqueda en tiempo real al catálogo de productos. Permite a los usuarios filtrar los productos visibles en función del texto ingresado en el campo de búsqueda.

---

### Propósito del archivo:

- Filtrar productos:

Detectar coincidencias entre el texto ingresado en el campo de búsqueda y los nombres de los productos disponibles.

- Mejorar la experiencia del usuario:

Mostrar resultados en tiempo real sin necesidad de recargar la página.

- Indicar resultados vacíos:

Mostrar un mensaje si no se encuentran productos que coincidan con el término de búsqueda.

---

### Análisis del código:

1. Configuración inicial:

Una vez que el contenido de la página ha cargado completamente (**mediante DOMContentLoaded**), el código obtiene el campo de entrada de búsqueda (**searchInput**), todas las tarjetas de productos (**productCards**) y el contenedor del catálogo (**catalogContainer**).

```
document.addEventListener("DOMContentLoaded", () => [  
    // Obtiene el input de búsqueda, las tarjetas de productos y el contenedor del catálogo  
    const searchInput = document.getElementById("searchInput");  
    const productCards = document.querySelectorAll(".product-card");  
    const catalogContainer = document.querySelector(".catalog-container");
```

## 2. Creación del mensaje si no se encuentran productos:

Se crea un mensaje que indica "No han encontrado productos con ese nombre." Este mensaje está oculto inicialmente, y se agregará al contenedor del catálogo.

```
// Crear el mensaje que se mostrará si no hay productos que coincidan con la búsqueda
let noResultsMessage = document.createElement("p");
noResultsMessage.id = "noResultsMessage";
noResultsMessage.textContent = "No han encontrado productos con ese nombre.";
noResultsMessage.style.display = "none"; // Ocultar por defecto
catalogContainer.appendChild(noResultsMessage); // Añadir el mensaje al contenedor del catálogo
```

- input EventListener:

Detecta cambios en el campo de búsqueda y ejecuta la lógica cada vez que el usuario escribe algo.

## 3. Búsqueda en tiempo real:

Cada vez que el usuario escribe en el campo de búsqueda:

- Convierte el texto ingresado a minúsculas para realizar una búsqueda insensible a mayúsculas.
- El código recorre cada tarjeta de producto y compara el nombre del producto con el término de búsqueda. Si el nombre del producto incluye el término de búsqueda, se muestra la tarjeta. Si no, se oculta.

```
// Detectar cuando el usuario escribe algo en el campo de búsqueda
searchInput.addEventListener("input", () => {
    const searchTerm = searchInput.value.toLowerCase();
    let matches = 0; // Contador de coincidencias

    // Recorrer todas las tarjetas de producto
    productCards.forEach(card => {
        const productName = card.getAttribute("data-name").toLowerCase();

        // Comprobar si el nombre del producto incluye el término de búsqueda
        if (productName.includes(searchTerm)) {
            card.style.display = "flex"; // Muestra el producto
            matches++; // Incrementar el contador de coincidencias
        } else {
            card.style.display = "none"; // Si no hay coincidencia, ocultar el producto
        }
    });
});
```

4. Mostrar u ocultar mensaje si no hay coincidencias:

Si no se encuentran productos que coincidan con la búsqueda, se muestra el mensaje de "**No se encontraron productos**". Si se encuentran productos, el mensaje se oculta.

```
// Mostrar mensaje si no hay coincidencias
if (matches === 0) {
    noResultsMessage.style.display = "block",
    noResultsMessage.style.color = "white",
    noResultsMessage.style.fontSize = "20px",
    noResultsMessage.style.margin = "20% auto";
} else {
    noResultsMessage.style.display = "none";
}
```

## Archivo: carrito.js

Este archivo implementa la lógica para gestionar el carrito de compras en el frontend, ofreciendo una experiencia interactiva para los usuarios. Incluye funcionalidades para añadir productos al carrito, visualizar el contenido, actualizar cantidades y eliminar productos.

---

### Propósito del archivo:

- Gestión de productos:  
Permite a los usuarios añadir, visualizar y eliminar productos del carrito.
- Visualización del carrito:  
Muestra un modal con el listado de productos seleccionados, incluyendo su precio, cantidad y total.
- Vaciar el carrito:  
Ofrece la opción de vaciar todo el carrito, con una confirmación previa.

### Estructura principal:

#### 1. Configuración inicial:

Al cargar la página (DOMContentLoaded), se configura el carrito como un array vacío. Se obtienen los elementos HTML del carrito, el modal, y los botones correspondientes.

```
document.addEventListener('DOMContentLoaded', () => {
  let carrito = []; // Usamos let porque el carrito puede cambiar a lo largo del script
  const cartModal = document.getElementById('cartModal'); // Modal del carrito
  const cartItems = document.getElementById('cartItems'); // Contenedor de los elementos del carrito
  const totalPriceElement = document.getElementById('totalPrice'); // Elemento donde se mostrará el total
  const emptyCartButton = document.getElementById('emptyCartButton'); // Botón para vaciar el carrito
```

## 2. Mostrar y cerrar el carrito:

- Abrir modal:

Al hacer clic en el botón "**Ver carrito**", este se muestra en un modal. Luego la función **actualizarCarrito()** se ejecuta para llenar el modal con los productos y su precio total.

- Cerrar modal:

Al hacer clic en el botón de cierre del modal "**Cerrar**", el carrito se oculta.

```
// Mostrar modal del carrito
document.getElementById('view-cart')?.addEventListener('click', () => {
  cartModal.style.display = 'flex';
  actualizarCarrito();
});

// Cerrar modal del carrito
document.getElementById('closeModal')?.addEventListener('click', () => {
  cartModal.style.display = 'none';
})
```

## 3. Añadir productos al carrito:

Cuando el usuario hace clic en el botón de añadir al carrito, este:

- Recoge datos del producto:

Obtiene el nombre, precio e imagen desde los atributos de la tarjeta del producto.

- Añade o actualiza el producto en el carrito:

Si el producto ya está en el carrito, incrementa la cantidad. Si no, lo añade como un nuevo ítem.

```

// Añadir productos al carrito al hacer clic en el botón de "Añadir al carrito"
document.querySelectorAll('.product-button').forEach(button => {
    button.addEventListener('click', (e) => {

        // Se obtiene información del producto
        const card = e.target.closest('.product-card');
        const name = card.getAttribute('data-name');
        const price = parseFloat(card.getAttribute('data-price'));
        const img = card.getAttribute('data-img');

        // Comprobar si los datos del producto son válidos
        if (!name || isNaN(price) || !img) {
            console.warn('Error: Datos del producto incompletos.');
            return;
        }

        // Comprobar si el producto ya está en el carrito
        const itemIndex = carrito.findIndex(item => item.name === name);
        if (itemIndex > -1) {
            // Si el producto ya está en el carrito, aumentar la cantidad
            carrito[itemIndex].quantity += 1;
        } else {
            // Si el producto no está en el carrito, agregarlo con cantidad 1
            carrito.push({ name, price, img, quantity: 1 });
        }

        // Actualizar el carrito cada vez que se agrega un producto
        actualizarCarrito();
    });
});

```

#### 4. Vaciar el carrito:

Si el usuario hace clic en el botón “**vaciar carrito**”, se le pregunta si está seguro de querer vaciarlo. Si el carrito no está vacío, se elimina todo su contenido y se actualiza la vista.

```

// Vaciar el carrito si el usuario confirma la acción
emptyCartButton?.addEventListener('click', () => {
    if (carrito.length > 0) {
        if (confirm('¿Estás seguro que deseas vaciar el carrito?')) {
            carrito = []; // Vaciar el carrito
            actualizarCarrito();
        }
    } else {
        alert('El carrito ya está vacío.'); // Si el carrito ya está vacío, mostrar mensaje
    }
});

```

## 5. Actualizar el carrito:

La función `actualizarCarrito()` limpia el contenedor del carrito, luego recorre los productos del carrito para mostrarlos con su nombre, imagen, precio y cantidad. Si el carrito está vacío, se muestra un mensaje. Esta función también recalcula el precio total y lo muestra en la interfaz.

```
// Función para actualizar el carrito (mostrar los productos y el total)
function actualizarCarrito() {
    cartItems.innerHTML = ''; // Limpiar la lista de productos actual
    let total = 0;

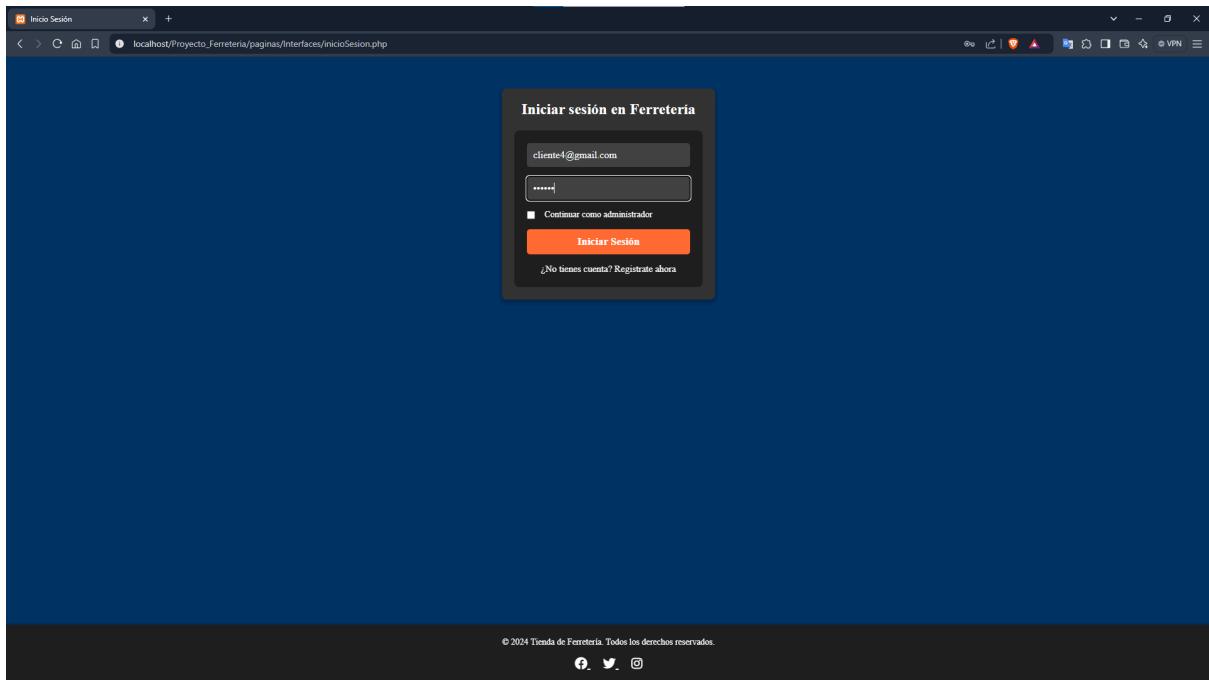
    if (carrito.length === 0) {
        // Si el carrito está vacío, muestra un mensaje
        const emptyMessage = document.createElement('p');
        emptyMessage.textContent = 'El carrito está vacío.';
        emptyMessage.style.TextAlign = 'center';
        cartItems.appendChild(emptyMessage);
    } else {
        // Se recorren los productos en el carrito y los muestra
        carrito.forEach(item => {
            const li = document.createElement('li');
            li.innerHTML =
                `![${item.name}](${item.img})
                <span>${item.name} - ${item.price.toFixed(2)}€ x ${item.quantity}</span>
                <button class="remove-item" data-name="${item.name}">Eliminar</button>`;
            cartItems.appendChild(li);
            total += item.price * item.quantity; // Calcula el total
        });

        // Reasignar eventos a los botones de eliminar
        cartItems.querySelectorAll('.remove-item').forEach(button => {
            button.addEventListener('click', (e) => {
                const name = e.target.getAttribute('data-name');
                carrito = carrito.filter(item => item.name !== name); // Eliminar producto
                actualizarCarrito(); // Actualizar la vista
            });
        });
    }

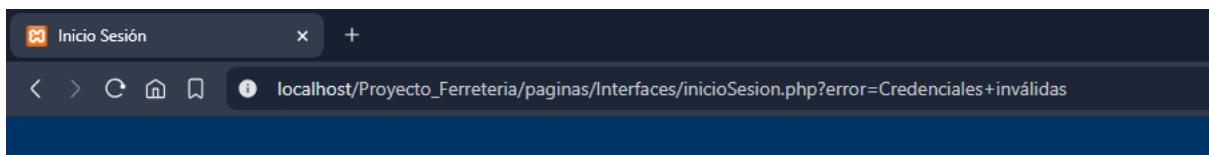
    // Mostrar el precio total del carrito
    totalPriceElement.textContent = `Total: ${total.toFixed(2)}€`;
}
```

## Página en funcionamiento

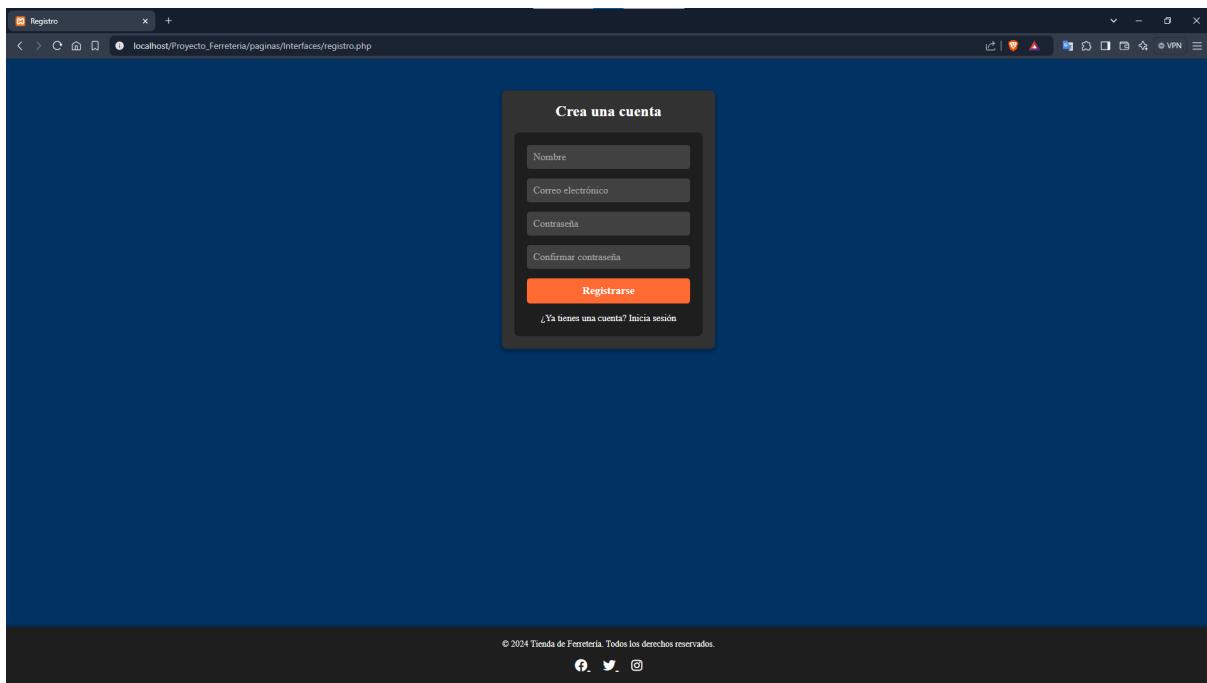
Intentaremos iniciar sesión con un usuario que aún no se ha registrado:



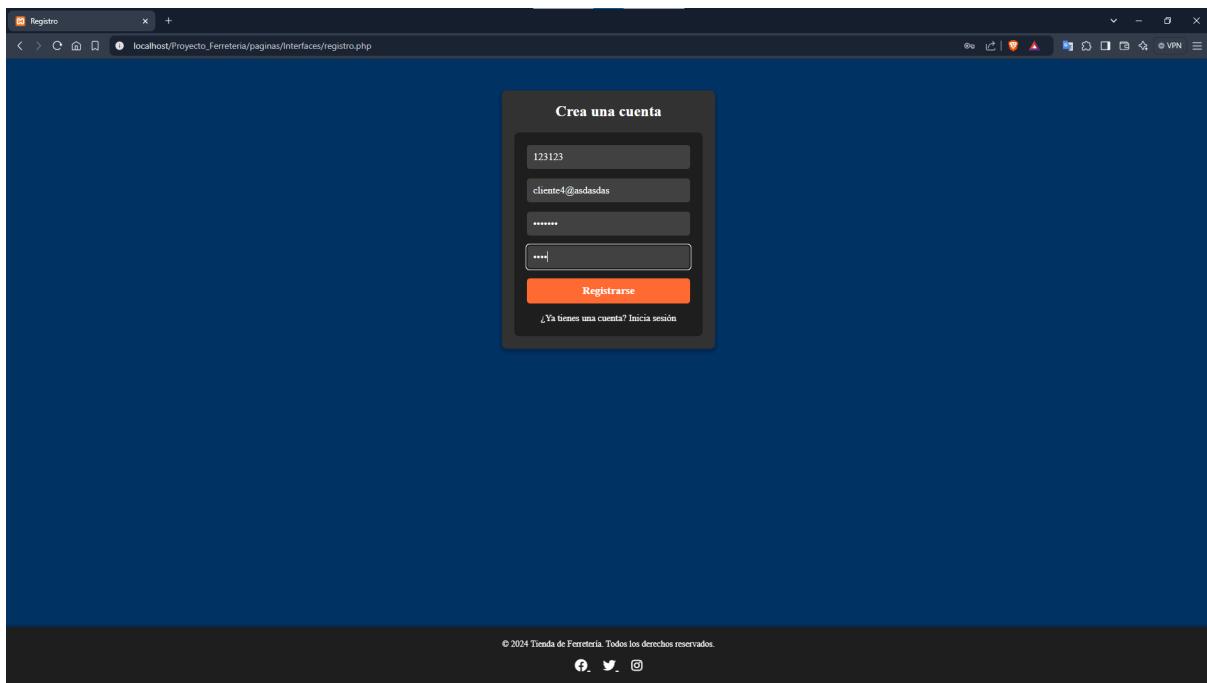
Al darle a iniciar sesión nos redirige a la misma página de login pero marcándonos el error en la url.



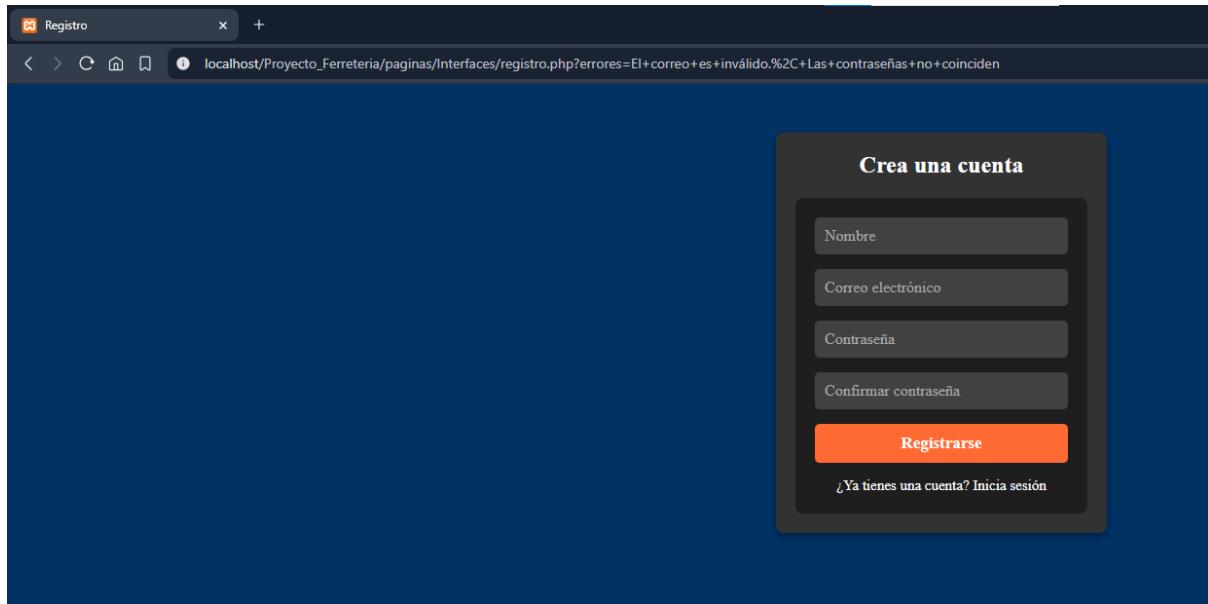
Le damos al enlace "¿No tienes cuenta? Regístrate ahora" y nos redirige a la interfaz del registro



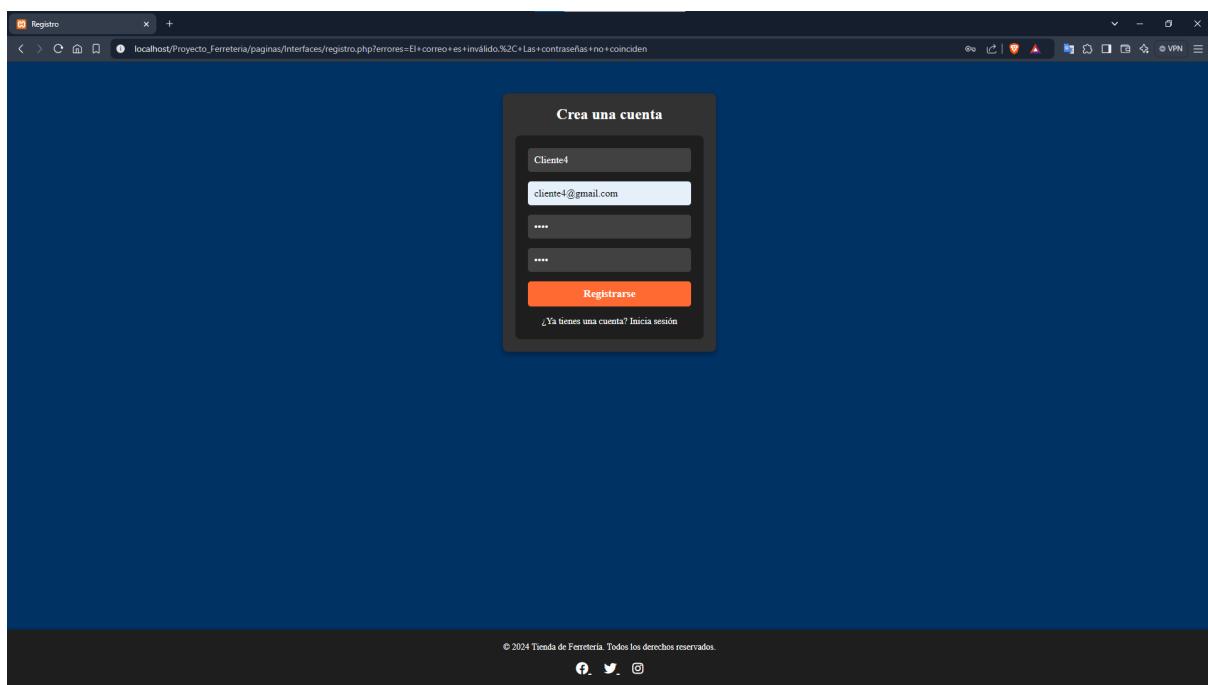
Aquí ingresamos todos los datos correspondientes. En esta ocasión vamos a ingresar mal los datos para verificar que funcionan las validaciones:



Le damos al botón “Registrarse” y como podemos ver, en la url nos aparecen los errores. En este caso no nos aparece ningún error al introducir números en el campo “Nombre de usuario” ya que luego, para el inicio de sesión, solo necesitaremos el correo electrónico y la contraseña.



Ahora si ingresamos bien los datos y le damos a “Registrarse”:



Ahora podemos ver como nos redirige al inicio de sesión con el mensaje de éxito en la url de esta página. También vemos que se añadió correctamente en la base de datos:

The top screenshot shows a browser window titled "Inicio Sesión" with the URL "localhost/Proyecto\_Ferreteria/paginas/interfaces/registro.php?exito=Usuario+registrado+con+éxito.". The page displays a login form for "Iniciar sesión en Ferretería" with fields for "Correo electrónico" and "Contraseña", a checkbox for "Continuar como administrador", and an orange "Iniciar Sesión" button. Below the form is a link "¿No tienes cuenta? Regístrate ahora". The bottom screenshot is a MySQL database table with columns "id", "nombre", "email", and "pass". It contains four rows: Cliente1 (id 2), Cliente2 (id 3), Cliente4 (id 6), and a new row with id \* and email "cliente4@gmail.com".

	id	nombre	email	pass
▶	2	Cliente1	cliente@gmail.com	1234
▶	3	Cliente2	cliente2@gmail.com	1234
*	6	Cliente4	cliente4@gmail.com	1234
*	NULL	NULL	NULL	cliente4@gmail.com

Procedemos a iniciar sesión:

The screenshot shows the same login page as before, but with the "correo electrónico" field populated with "cliente4@gmail.com" and the "contraseña" field partially visible as "\*\*\*\*". All other elements of the form and footer are identical to the previous screenshot.

Podemos ver como nos redirige al catálogo de productos y se cargan algunos productos anteriormente creados:

The screenshot shows a web browser window titled "Catalogo de Productos". The main content area displays two product cards. The first card is for "Clavos de Cabeza 40mm (Paquete de 100)" with a price of 10.49 € and a "Agregar al carrito" button. The second card is for "Tornillo Torx De Cabeza Avellanada" with a price of 15.64 € and a "Agregar al carrito" button. At the top right of the page are "Ver carrito" and "Cerrar sesión" buttons.

Le damos a “Ver carrito” y obviamente este va a estar vacío.

The screenshot shows the same "Catalogo de Productos" page. A modal window titled "Carrito de Compras" is open in the center. It displays the message "El carrito está vacío." and "Total: 0.00€". There are "Cerrar" and "Vesar Carrito" buttons at the bottom of the modal. The rest of the page content is identical to the previous screenshot.

Le agregaremos algún producto (Tornillo Torx) y posteriormente lo eliminaremos del carrito.

Catálogo de Productos

Clavos de Cabeza 40mm (Paquete de 100)

Estos clavos Closet son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad.

Precio: 10.49 €

Agregar al carrito

Tornillo Torx De Cabeza Avellanada

¡Gran fuerza y buena resistencia a la corrosión para que los sujetadores duren! Los tornillos pueden usarse ampliamente en todo tipo de maquinaria de precisión, instrumentos clasificados, electrodo

Precio: 15.64 €

Agregar al carrito

© 2024 Tienda de Ferretería. Todos los derechos reservados.

f t i

Catálogo de Productos

Clavos de Cabeza 40mm (Paquete de 100)

Estos clavos Closet son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad.

Precio: 10.49 €

Agregar al carrito

Tornillo Torx De Cabeza Avellanada

¡Gran fuerza y buena resistencia a la corrosión para que los sujetadores duren! Los tornillos pueden usarse ampliamente en todo tipo de maquinaria de precisión, instrumentos clasificados, electrodo

Precio: 15.64 €

Agregar al carrito

Carrito de Compras

Tornillo Torx De Cabeza Avellanada - 15.64€ x 1

Total: 15.64€

Eliminar

Cerrar Ver carrito

© 2024 Tienda de Ferretería. Todos los derechos reservados.

f t i

**Catálogo de Productos**

**Clavos de Cabeza 40mm (Paquete de 100)**

Estos clavos Clout son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad.

Precio: 10.49 €

**Agregar al carrito**

**Tornillo Torx De Cabeza Avellanada**

¡Gran fuerza y buena resistencia a la corrosión para que los sujetadores duren! Los tornillos pueden usarse ampliamente en todo tipo de maquinaria de precisión, instrumentos clasificados, electrodo

Precio: 15.64 €

**Agregar al carrito**

**Carrito de Compras**

El carrito está vacío.

Total: 0.00€

**Cerrar** **Vaciar Carrito**

Ahora agregaremos varios productos y utilizaremos el botón “Vaciar Carrito”. Como se podrá observar, en la parte superior de la pantalla nos sale una alerta y le damos a aceptar para proceder a vaciar el carrito.

**Catálogo de Productos**

**Clavos de Cabeza 40mm (Paquete de 100)**

Estos clavos Clout son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad.

Precio: 10.49 €

**Agregar al carrito**

**Tornillo Torx De Cabeza Avellanada**

¡Gran fuerza y buena resistencia a la corrosión para que los sujetadores duren! Los tornillos pueden usarse ampliamente en todo tipo de maquinaria de precisión, instrumentos clasificados, electrodo

Precio: 15.64 €

**Agregar al carrito**

**localhost dice**

¿Estás seguro que deseas vaciar el carrito?

**Aceptar** **Cancelar**

**Carrito de Compras**

Clavos de Cabeza 40mm (Paquete de 100) - 10.49€ x 3

**Eliminar**

Tornillo Torx De Cabeza Avellanada - 15.64€ x 1

**Eliminar**

Total: 47.11€

**Cerrar** **Vaciar Carrito**

Catálogo de Productos

localhost/Proyecto\_Ferreteria/paginas/interfaces/catalogo.php

Catálogo de Productos

Buscar productos...

Ver carrito Cerrar sesión

**Clavos de Cabeza 40mm  
(Paquete de 100)**

Estos clavos Clout son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad.

Precio: 10.49 €

Agregar al carrito

**Tornillo Torx De Cabeza  
Avellanada**

¡Gran fuerza y buena resistencia a la corrosión para que los sujetadores duren! Los tornillos pueden usarse ampliamente en todo tipo de maquinaria de precisión, instrumentos clasificados, electrodo

Precio: 15.64 €

Agregar al carrito

Carrito de Compras

El carrito está vacío.

Total: 0.00€

Cerrar Vaciar Carrito

© 2024 Tienda de Ferretería. Todos los derechos reservados.

f t i

Ahora usaremos la barra de búsqueda:

Catálogo de Productos

localhost/Proyecto\_Ferreteria/paginas/interfaces/catalogo.php

Catálogo de Productos

Ciudad

Ver carrito Cerrar sesión

**Clavos de Cabeza 40mm  
(Paquete de 100)**

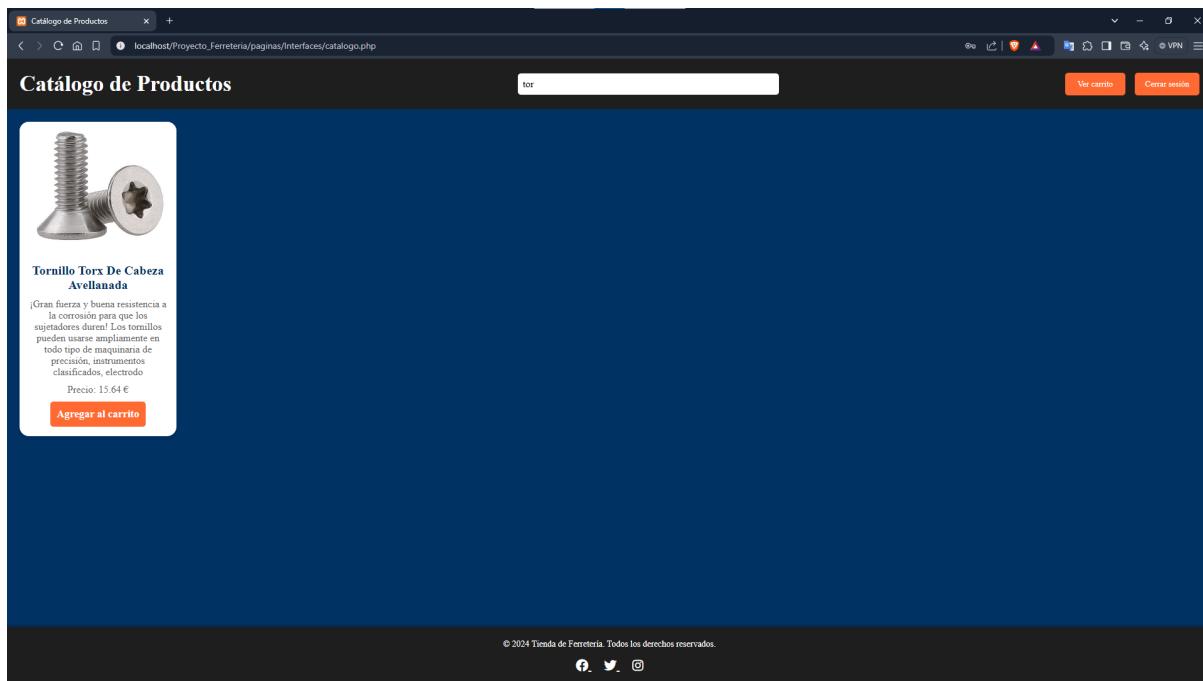
Estos clavos Clout son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad.

Precio: 10.49 €

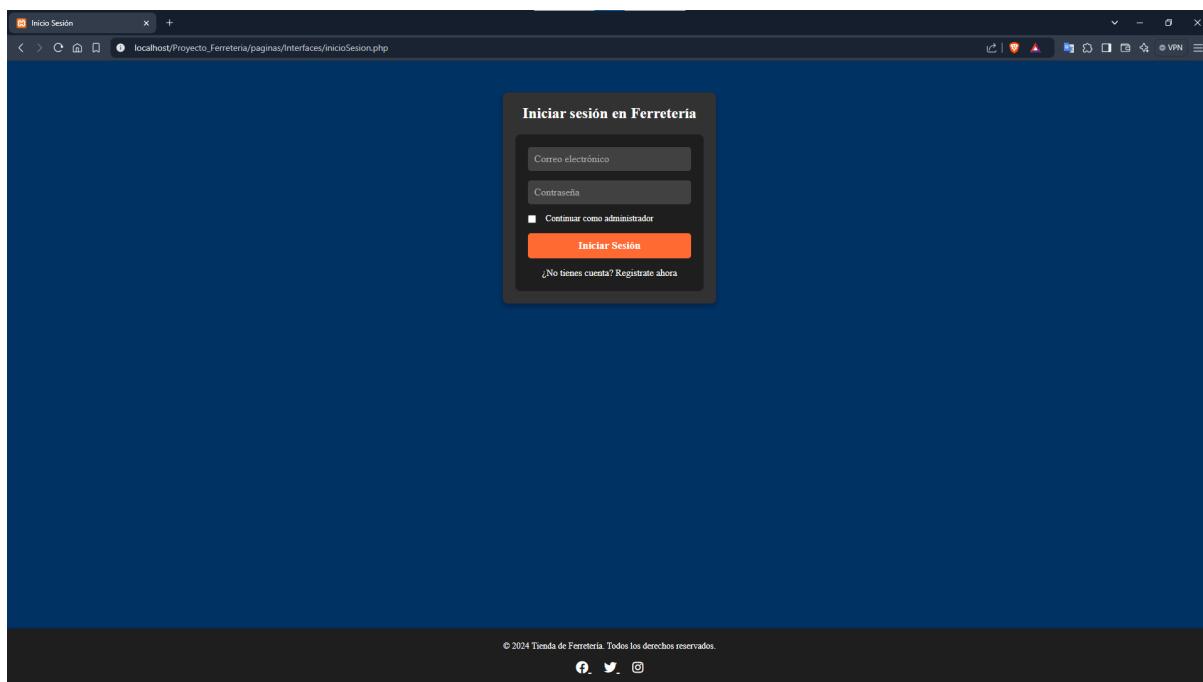
Agregar al carrito

© 2024 Tienda de Ferretería. Todos los derechos reservados.

f t i



Por último cerramos sesión. Lógicamente, esto nos redirige al login.

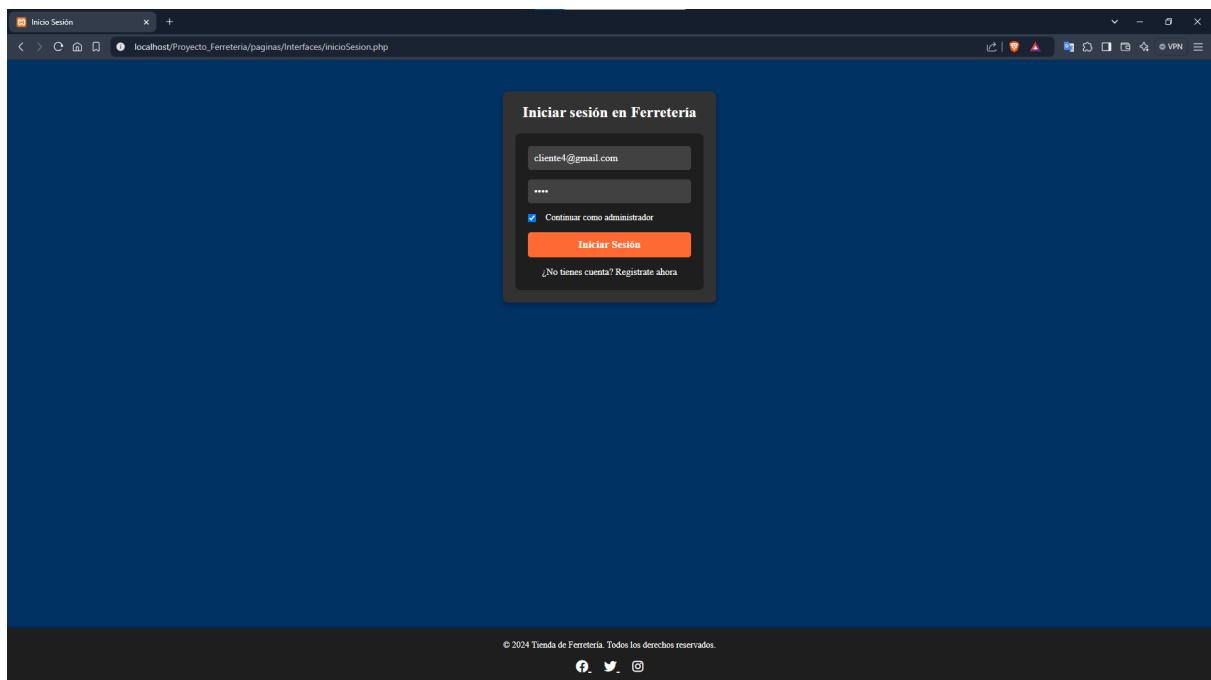


Anteriormente, en la base de datos hemos creado 3 admins, ya que no hemos hecho una página para el registro de los administradores.

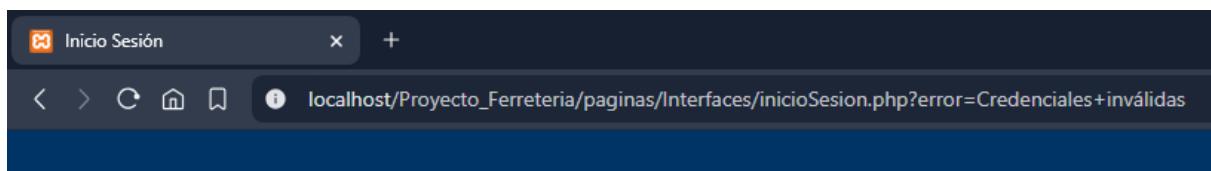
```
INSERT INTO administrador (nombre, email, pass) VALUES ('Gian', 'gian@gmail.com', 1234);
INSERT INTO administrador (nombre, email, pass) VALUES ('Raul', 'raul@gmail.com', 1234);
INSERT INTO administrador (nombre, email, pass) VALUES ('Ivan', 'ivan@gmail.com', 1234);
```

Intentaremos iniciar sesión con el usuario creado anteriormente y marcamos la casilla “Continuar como administrador”:

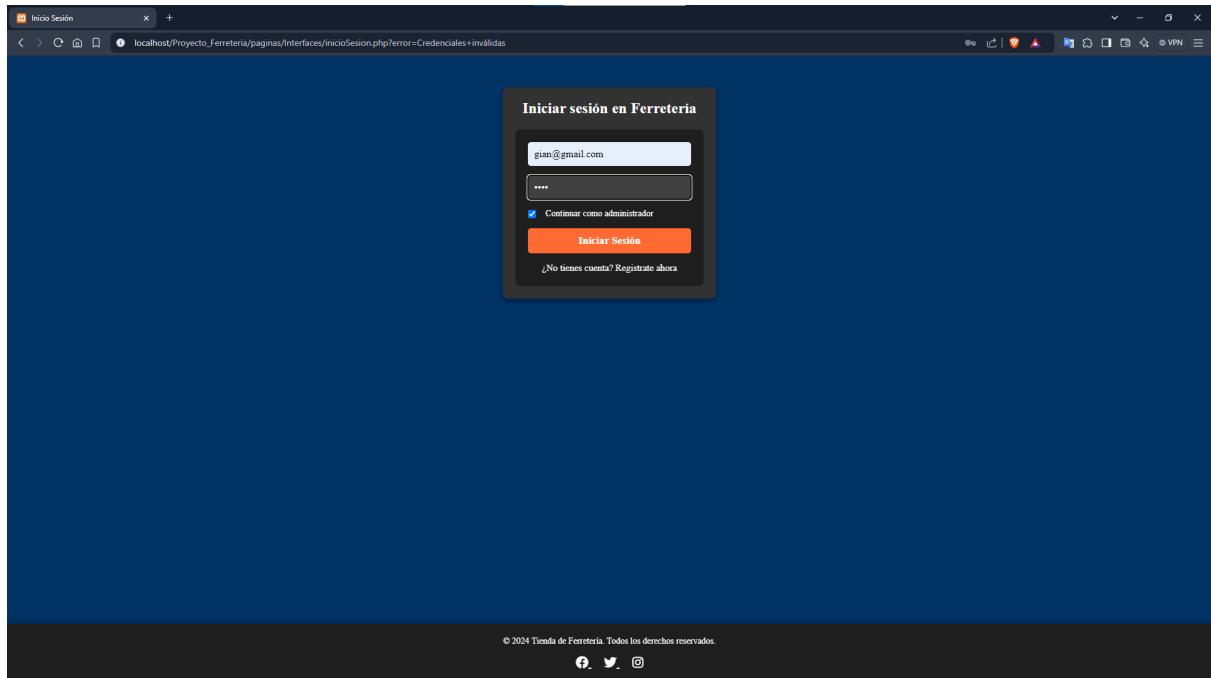
- email: “cliente4@gmail.com”
- contraseña: 1234



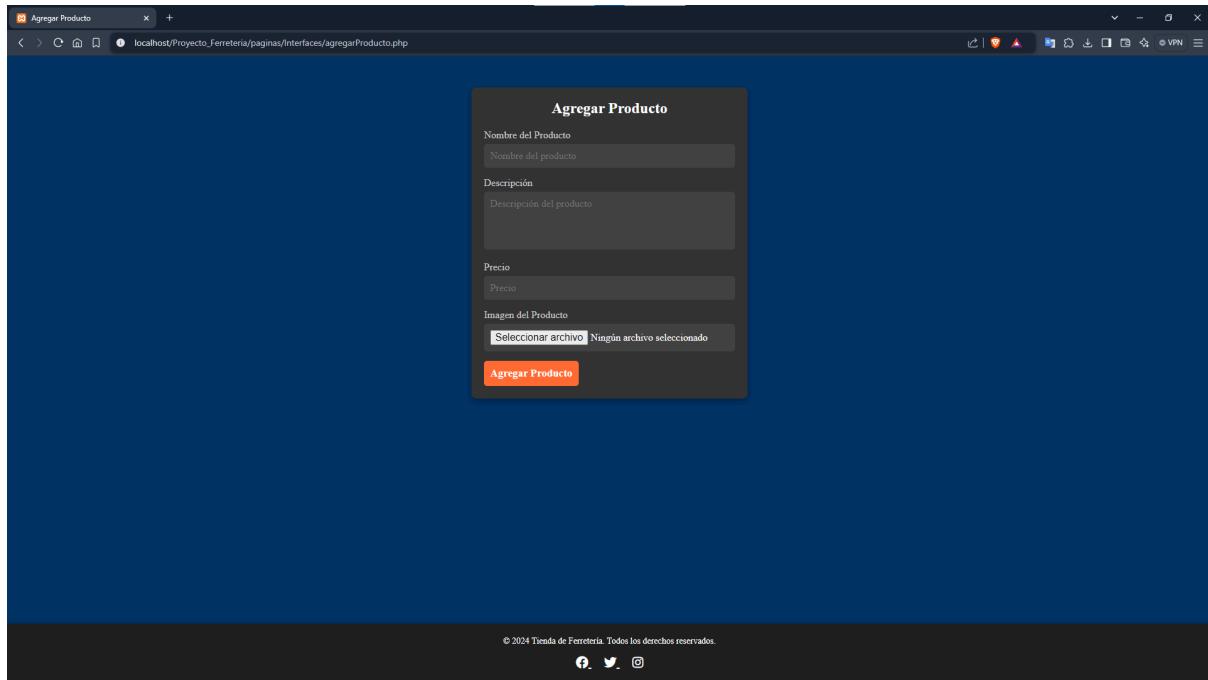
Como se podrá observar nos sale el error en la URL:



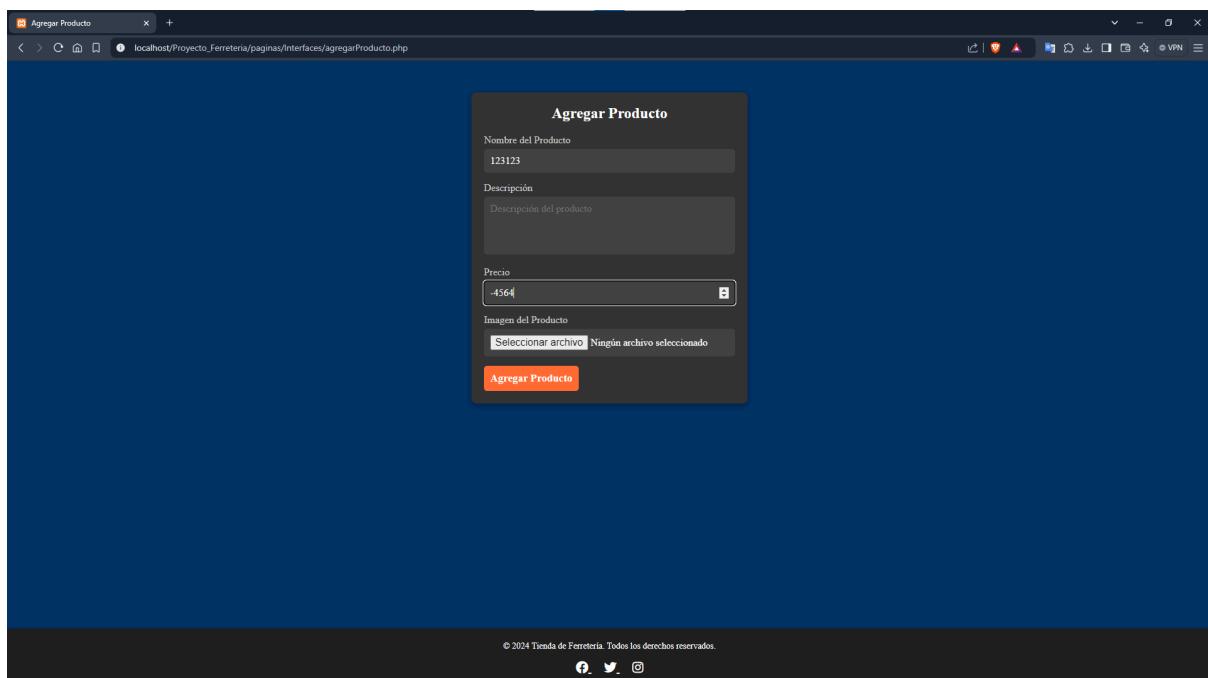
Ahora ingresamos las credenciales correctamente e iniciamos sesión como admin. Esto nos redirige al catálogo de productos con las funciones correspondientes para el administrador:



Ahora agregaremos un nuevo producto al catálogo. Para eso le damos al botón “Agregar producto”. Esto nos redirige a la página correspondiente:



Aquí ingresamos, nuevamente, **mal** todos los datos para verificar que también funcionan estas validaciones. Luego le damos al botón “Agregar Producto” y veremos los errores correspondientes en la URL:



Agregar Producto

localhost/Proyecto\_Ferreteria/paginas/Interfaces/agregarProducto.php?errores=La+descripción+del+producto+es+inválida.%2C+El+precio+debe+ser+un+número+positivo.%2C+La+imagen+es+inválida+o+...

**Agregar Producto**

Nombre del Producto

Descripción

Precio

Imagen del Producto  
 Ningún archivo seleccionado

**Agregar Producto**

Si le intentamos poner un nombre que ya existe en la base de datos también nos dará error, ya que este no puede estar duplicado.

Agregar Producto

localhost/Proyecto\_Ferreteria/paginas/Interfaces/agregarProducto.php

**Agregar Producto**

Nombre del Producto

Descripción

Precio

Imagen del Producto  
 Tornillo Torx.jpg

**Agregar Producto**

© 2024 Tienda de Ferretería. Todos los derechos reservados.  
Facebook Twitter Instagram

Agregar Producto

localhost/Proyecto\_Ferreteria/paginas/Interfaces/agregarProducto.php?error=El+producto+con+el+nombre+Tornillo+Torx+De+Cabeza+Avellanada+ya+existe.

Ahora los ingresamos correctamente y le damos al botón. Una vez creado el producto, nos redirige al catálogo nuevamente. En la BBDD podremos observar el nuevo producto agregado.

	<b>id</b>	<b>nombre</b>	<b>precio</b>	<b>descripcion</b>	<b>imagen</b>
▶	1	Clavos de Cabeza 40mm (Paquete de 100)	10.49	Estos clavos Clout son de acero de aleación zinc...	BLOB
	2	Tornillo Torx De Cabeza Avellanada	15.64	¡Gran fuerza y buena resistencia a la corrosión ...	BLOB
*	8	Martillo degarra inoxidable con mango de fibra de	23.82	El mejor material y duradero: los martillos de ga...	BLOB
	NULL	NULL	NULL	NULL	NULL

Ahora al producto recién creado vamos a eliminarlo clicando en el botón “Eliminar”, lógicamente.

No se llega a observar bien pero, al pasar el ratón por encima del producto, la tarjeta del mismo se eleva un poco y el botón se oscurece.

Le damos a eliminar:

Catálogo de Productos

Buscar productos...

(Eres Admin)

Agregar producto

Cerrar sesión

Clavos de Cabeza 40mm (Paquete de 100)

Tornillo Torx De Cabeza Avellanada

Martillo degarra inoxidable con mango de fibra de carbono

© 2024 Tienda de Ferretería. Todos los derechos reservados.

Catálogo de Productos

Buscar productos...

Clavos de Cabeza 40mm (Paquete de 100)

Tornillo Torx De Cabeza Avellanada

Como se puede observar el producto se ha eliminado, y en la URL nos aparece el mensaje de éxito. Si actualizamos la BBDD ya no estará disponible ese producto:

	id	nombre	precio	descripcion	imagen
▶	1	Clavos de Cabeza 40mm (Paquete de 100)	10.49	Estos clavos Clout son de acero de aleación zinc...	BLOB
◀	2	Tornillo Torx De Cabeza Avellanada	15.64	¡Gran fuerza y buena resistencia a la corrosión ...	BLOB
*	NULL	NULL	NULL	NULL	NULL

Por último, y con esto terminamos, modificaremos un producto del catálogo. Le damos al botón “Modificar” (Tornillo Torx) y nos redirige a la página correspondiente, donde se cargarán los datos del producto seleccionado.

A este producto, le cambiaremos el nombre y la imagen, y le damos a confirmar:

Catálogo de Productos

localhost/Proyecto\_Ferreteria/paginas/Interfaces/catalogoAdmin.php?exito=Producto+eliminado+con+éxito.

(Eres Admin)

Agregar producto Cerrar sesión

Clavos de Cabeza 40mm (Paquete de 100)

Estos clavos Clout son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad

Precio: 10.49 €

Eliminar Modificar

Tornillo Torx De Cabeza Avellanada

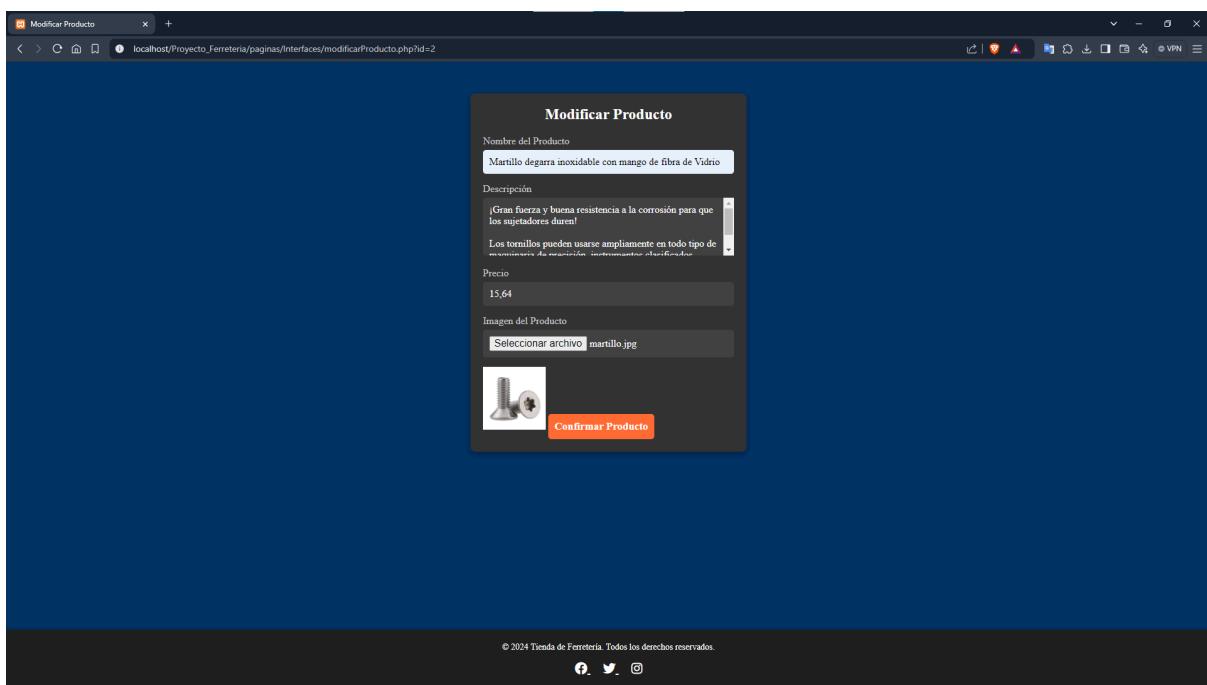
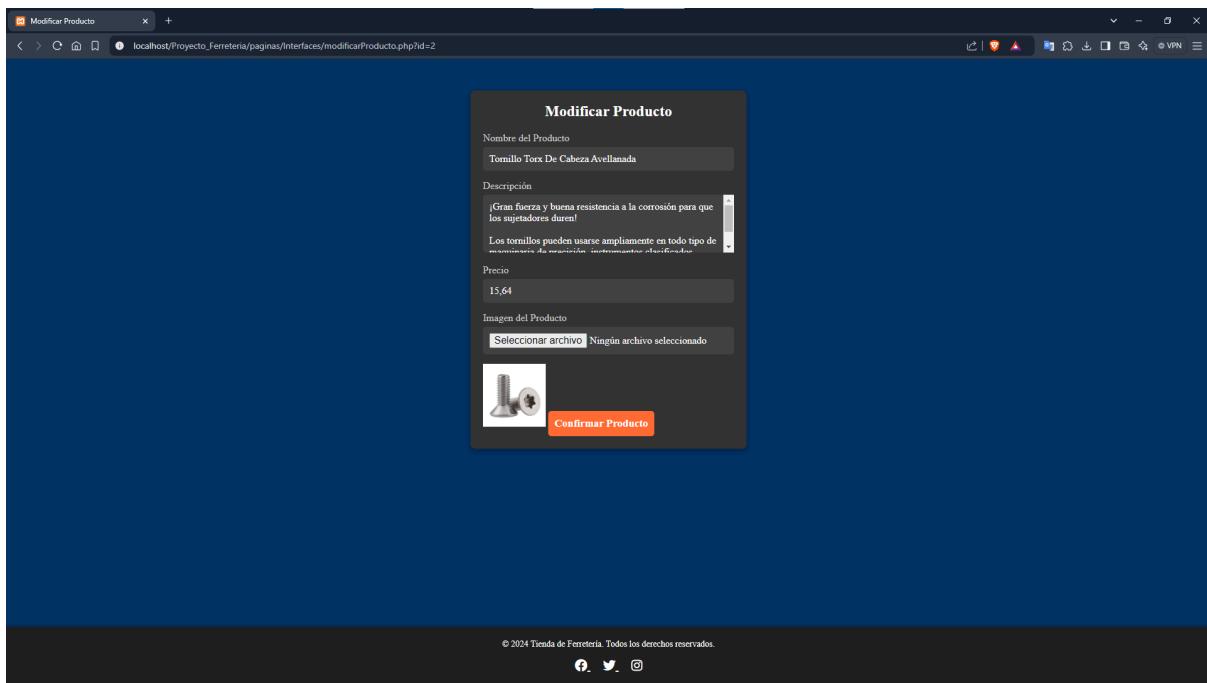
¡Gran fuerza y buena resistencia a la corrosión para que los sujetadores duren! Los tornillos pueden usarse ampliamente en todo tipo de maquinaria de precisión, instrumentos clasificados, electrodo durabilidad

Precio: 15.64 €

Eliminar Modificar

© 2024 Tienda de Ferretería. Todos los derechos reservados.

f t i



Como podemos observar, en la URL nos aparece el mensaje de éxito y en el catálogo ya está actualizado el producto. Si vamos a la BBDD veremos que el ID del producto es el mismo, sólo cambió su nombre e imagen:

Catálogo de Productos

localhost/Proyecto\_Ferreteria/paginas/Interfaces/catalogoAdmin.php?exito=Producto+modificado+exitosamente.

**Clavos de Cabeza 40mm (Paquete de 100)**

Clavos de Cabeza 40mm (Paquete de 100)

Estos clavos Clout son de acero de aleación zincado, que es una de las formas más altas de acero resistente a la corrosión, lo que hace que duren más que otros herrajes. Añade protección y durabilidad

Precio: 10.49 €

**Martillo degarra inoxidable con mango de fibra de**

Martillo degarra inoxidable con mango de fibra de

¡Gran fuerza y buena resistencia a la corrosión para que los sujetadores duren! Los tornillos pueden usarse ampliamente en todo tipo de maquinaria de precisión, instrumentos clasificados, electrodo

Precio: 15.64 €

**Eliminar**   **Modificar**

**Eliminar**   **Modificar**

	id	nombre	precio	descripcion	imagen
▶	1	Clavos de Cabeza 40mm (Paquete de 100)	10.49	Estos clavos Clout son de acero de aleación zinc...	BLOB
▶	2	Tornillo Torx De Cabeza Avellanada	15.64	¡Gran fuerza y buena resistencia a la corrosión ...	BLOB
*	NULL	NULL	NULL	NULL	NULL

	id	nombre	precio	descripcion	imagen
▶	1	Clavos de Cabeza 40mm (Paquete de 100)	10.49	Estos clavos Clout son de acero de aleación zinc...	BLOB
▶	2	Martillo degarra inoxidable con mango de fibra de	15.64	¡Gran fuerza y buena resistencia a la corrosión ...	BLOB
*	NULL	NULL	NULL	NULL	NULL

FIN 😊