



DOCUMENTACIÓN TÉCNICA

Contenido

1.	Introducción.....	2
2.	Integrantes del proyecto y funciones.	2
3.	Objetivos.....	3
4.	Requisitos del sistema	4
5.	Tecnologías utilizadas	4
6.	Diseño del sistema	5
7.	Funcionamiento del sistema	8
	• Carrito de compras.....	8
	• Gestión de stock	9
	• Gestión de admin y usuarios	10
	• Pedidos	11
	• Interfaz HTML + CSS.....	13
	• Validación del formulario (SkyNET/Backend/Productos/Scripts/validar_formulario.js)	16
	• Búsqueda con XPath	17
	• Exportación de productos en XML.....	17
	• Backups automáticos.....	17
8.	Algunas de la pruebas	18
9.	Mejoras y Escalabilidad.....	20
10.	Anexos.	20

1. Introducción

Este proyecto implica la creación de una página web orientada a la comercialización de productos tecnológicos. El objetivo principal es proporcionar un lugar donde los usuarios puedan consultar, comparar y comprar diversos productos tecnológicos como smartphones, tablets, portátiles, consolas y ordenadores ya montados. La página se diseñará para ser sencilla de usar y proporcionar toda la información necesaria sobre cada producto.

La meta del proyecto es satisfacer la demanda de una página confiable y que sea simple la compra de tecnología en línea. Además, aspira a ofrecer una opción asequible para aquellos usuarios que requieren productos tecnológicos.

Está orientada a un público bastante diverso: desde personas que simplemente buscan tecnología para uso personal hasta usuarios más sofisticados que requieren productos más particulares.

En el [anexo](#) se incluye un enlace al documento de **análisis funcional** donde se detalla como queríamos que funcionara la tienda.

2. Integrantes del proyecto y funciones.

La base de datos de SkyNET fue realizada en conjunto con mis compañeros, dando le un enfoque simple que completamente funcional.

- **Brahim Hanaoui Karbab (Parte Frontend y Estilo).**
 - Planificación inicial del proyecto y marketing.
 - Funcionamiento:
 - Formulario soporte.
 - Página inicio.
 - Página individual del producto.
 - Búsqueda de imágenes de productos.
 - CSS:
 - Diseños de todas las páginas, la elección del CSS junto a los colores y la disposición de cada parte de la página.
 - Implementación:
 - Redes sociales.
- **José García Martínez-Abarca (Full Frontend).**
 - Documentación:
 - Análisis funcional (ANALISIS_FUNCIONAL.pdf)
 - Estructura de los puntos a realizar (Estructura de los puntos a realizar.pdf).
 - Funcionamiento:
 - Carrito de compra.
 - Checkout.
 - login / signup.
 - Página perfil usuario.
 - página principal.
 - CSS:
 - Integración del estilo CSS proporcionado por Ibra.
 - Implementación:
 - Implementación de GitHub en VS Code.

- **Antonio Rodríguez Blaya (Full Backend).**
 - Documentación:
 - Manual usuario cliente (Manual usuario(Usuarios).pdf).
 - Manual usuario gestión (Manual usuario(gestión tienda).pdf).
 - Manual Instalación (Manual instalación.pdf).
 - Manual Backup base de datos (BACKUP AUTOMATICO BASE DE DATOS EN WINDOWS.pdf).
 - Funcionamiento:
 - Formulario de productos.
 - Validaciones formulario.
 - Listado de productos
 - Búsqueda en XML.
 - Exportación Xpath.
 - Listado historial de ventas.
 - Listado de incidencias / preguntas.
 - Implementaciones:
 - Creación del proyecto en GitHub.
 - Creación del Trello.
 - Copia de Seguridad de base de datos en Windows (Backup).
 - CSS:
 - Integración del estilo CSS proporcionado por Ibra.

3. Objetivos

Objetivo principal:

Crear un sitio web eficaz y atractivo enfocado en la comercialización de productos tecnológicos, que facilite a los usuarios la navegación, la comprensión de los detalles de los productos y la realización de compras de forma segura y eficaz.

Objetivos específicos:

- Carrito de compras, gestión de stock, pedidos y usuarios.
- Interfaz amigable con HTML + CSS y validación de formularios.
- Búsqueda de productos con XPath en XML.
- Backups automáticos de la base de datos.
- Exportación de productos en XML.

4. Requisitos del sistema

- **Hardware y mínimo recomendado.**

Usuarios.

- Tener conexión a internet.
- Tener un navegador web.
- Ordenador, teléfono o Tablet.

Para desarrollo

- Tener instalado el XAMP, VS Code y el proyecto. (en el anexo hay un enlace a un tutorial paso a paso de cómo instalar el proyecto para poder trabajar en él).

Componentes	Requisitos Mínimos	Requisitos Recomendados
Procesador (CPU).	Intel Core i3 / AMD Ryzen 3	Intel Core i5 / AMD Ryzen 5 (4 núcleos)
Memoria RAM	8 GB	16 GB o más (para multitarea)
Almacenamiento	SSD de 256 GB	SSD de 512GB o más (más rápido para BD)
Sistema Operativo	Windows 10 / 11	Windows 11

- **Software necesario.**

- Navegador web (Preferiblemente Google Chrome).
- Servidor web local XAMPP para usar la página web.
- PHP, como lenguaje de programación del lado del servidor.
- Navegador con soporte de HTML, CSS, JavaScript.
- Para gestionar base de datos (MySQL).
- Para estructura de datos y pruebas: uso de XML y XPath.

5. Tecnologías utilizadas

- **Lenguajes** → HTML, CSS, JavaScript, PHP. → Backend y Frontend.
- **XML** → Para exportación y búsqueda con Xpath → Backend.
- **Base de datos** → MySQL → Backend.
- **Entorno de desarrollo** → Visual Studio Code → Backend y Frontend.
- **Herramientas de control de versiones** → GitHub y Trello.

6. Diseño del sistema

○ Estructuras de las carpetas del proyecto.

SkyNET



○ Diseño de la interfaz

• Backend

Página principal (index) donde podemos acceder a diversas funciones de gestión de los productos como añadir productos, listado de productos, historial de ventas e Listado incidencias / preguntas.

En el [anexo](#) se incluye enlace al **manual usuario (gestión tienda)** donde se explica con más detalle cada función.



Ilustración 1. index del backend -- imagen de elaboración nuestra.

• Frontend

Página principal (Tienda) de SkyNET donde se puede ver el catálogo de productos, filtros, barra de navegación, icono de iniciar sesión o registrarse con nosotros, icono del carrito.

En el [anexo](#) se incluye enlace al **manual de gestión de usuario (Usuario)** donde se explica con más detalle cada función.

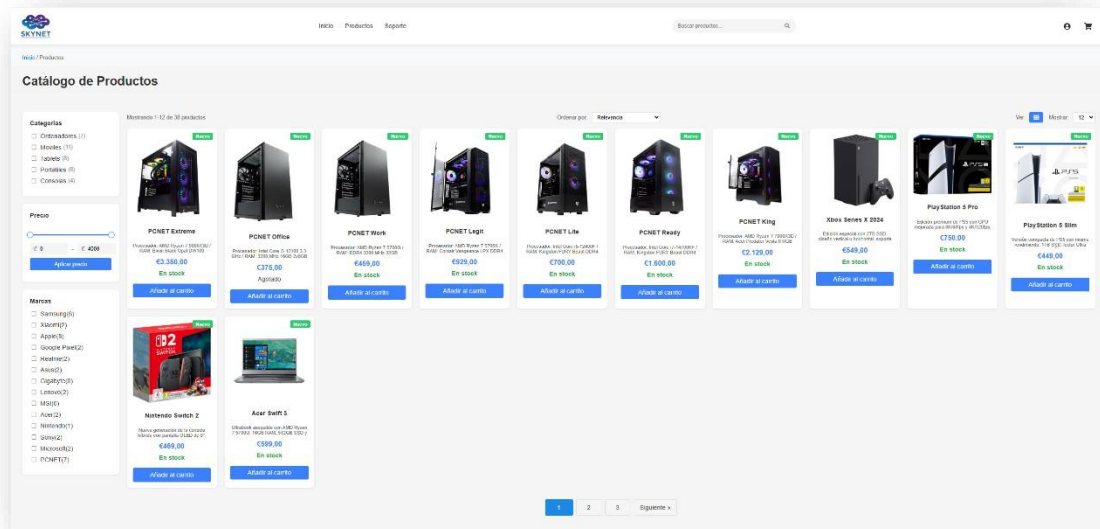


Ilustración 2. Página principal donde están los productos -- imagen de elaboración nuestra.

○ Diagramas

○ Diagrama de entidad relación

El diagrama entidad-relación está representando la estructura de la base de datos del sistema, mostrando sus principales entidades, atributos y relaciones. Se destacan entidades clave como **productos**, **clientes**, **ventas**, **carrito** y **direcciones**, junto con otras entidades auxiliares como **categorías**, **marcas**, **impuestos** y **método de pago**.

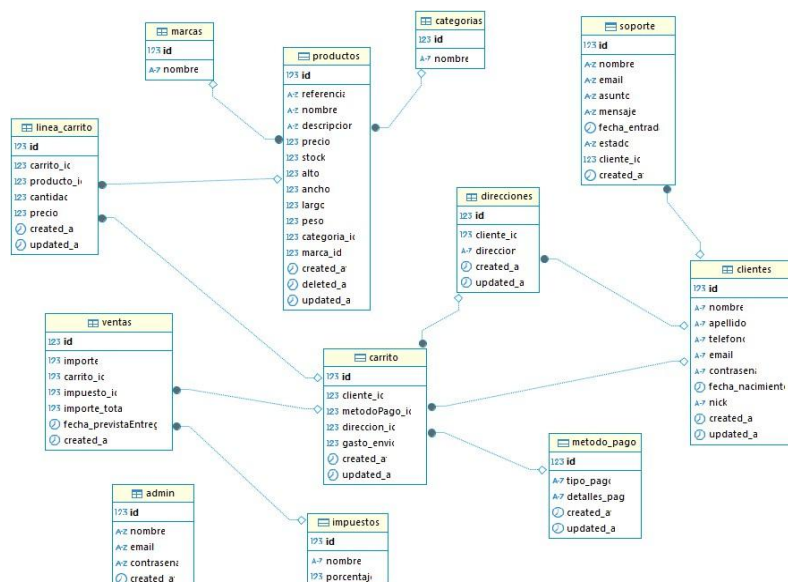


Ilustración 3. Diagrama de entidad relación de SkyNET en DBaiver -- imagen de elaboración nuestra.

○ Diagrama de casos

Usuario

- **Ver productos:** Consultar productos si están disponible.
- **Usar carrito:** Agregar productos al carrito.
- **Realizar Checkout:** Finalizar la compra y proceder al pago.
- **Enviar mensaje a soporte:** Contactar con soporte para preguntas / problemas.

Administrador

1. **Mostrar productos:** Ver listado de productos.
2. **Modificar producto:** Editar campos del producto.
3. **Eliminar producto:** Eliminar productos.
4. **Añadir producto:** Añadir productos desde el formulario.
5. **Gestión de historial:** Ver el historial de ventas.
6. **Gestión de peticiones de soporte:** Atender y gestionar mensajes de preguntas / soporte.

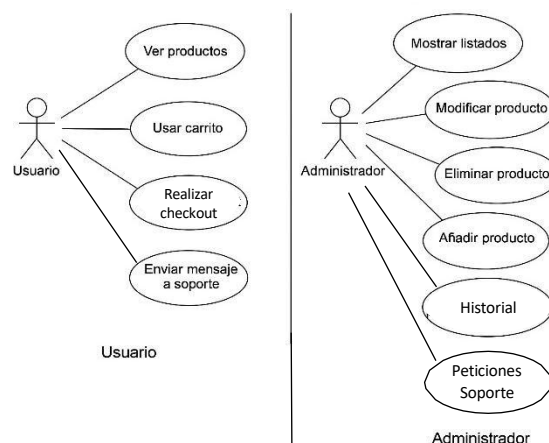
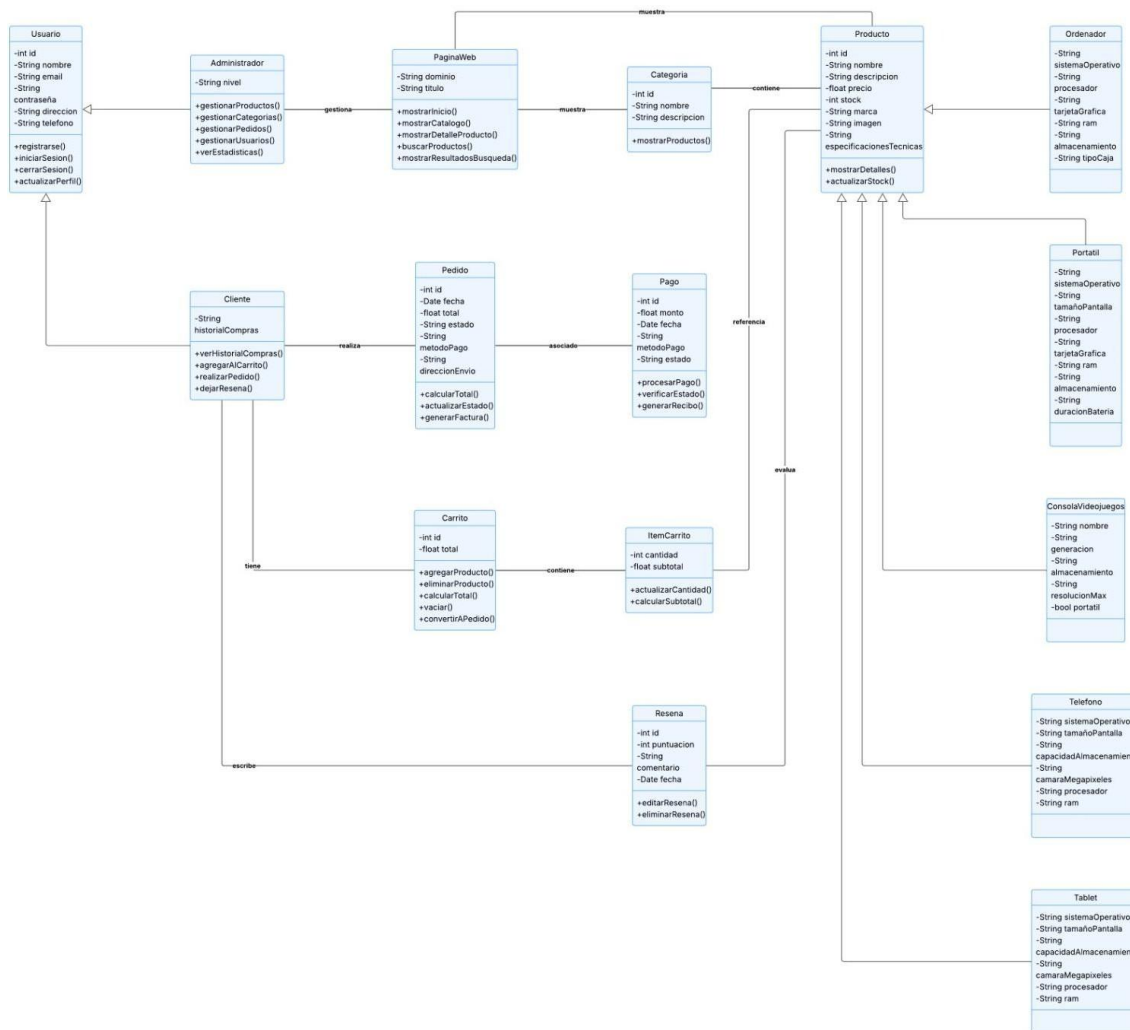


Ilustración 4. Usuario y Administrador D.C -- imagen de elaboración nuestra.

○ Diagrama UML



El diagrama UML representa la estructura y relaciones entre las principales clases del sistema SkyNET, incluyendo usuarios, productos, pedidos y gestión de la tienda.

7. Funcionamiento del sistema

• Carrito de compras:

1. Apertura/Cierre del Carrito:

- Se abre al hacer clic en el icono del carrito.
- Se cierra con el botón de cerrar o haciendo clic fuera del carrito.

2. Gestión de Productos:

- Añadir productos: Con el botón de “Añadir producto” desde la página principal o desde la página del producto.
- Modificar cantidades: Con botones +/- para cada producto.
- Eliminar productos: Con botón de la papelera puedes borrar el producto. del carrito.
- Cálculo automático: del subtotal basado en precios y cantidades.

3. Conservación del estado:

- Guarda el carrito en local Storage para mantenerlo entre sesiones.
- Recupera el carrito al cargar la página.

4. Visualización:

- Muestra notificaciones al añadir productos
- Muestra un badge con la cantidad total de productos en el icono del carrito.
- Alterna entre vista de "carrito vacío" y lista de productos.

5. Conversión de precios:

- Convierte precios al formato en euros (español).
- Convertir precios desde el formato español a decimal para cálculos.

6. Continuación de la compra:

- Botón "Finalizar compra" que redirige a checkout.php (con validación).

7. Página de Listado (interfaz.php):

- Detecta botones "Añadir al carrito" en las tarjetas de producto.
- Extrae ID, nombre y precio del producto para añadirlo al carrito.

8. Página de Detalle (producto.php):

- Botón de "Añadir al carrito".
- Extrae ID, nombre del título y precio de la página.
- Tiene en cuenta la cantidad seleccionada por el usuario.

• Gestión de stock:

1. Validaciones.

- Datos necesarios:
 - Producto_id (debe ser mayor a 0).
 - cantidad (debe ser mayor a 1).

2. Consulta Stock.

- Busca el producto en la tabla "productos".
 - Filtra por el id, si lo encuentra devuelve id con el producto.
 - Si no existe, devuelve "Producto no encontrado".
- Considera el carrito actual:
 - Recibe los productos del carrito `$_POST['cart_items']`.
 - Suma la cantidad solicitada + la cantidad que hay en el carrito.

3. Ver el stock actual de un producto.

- Desde el listado de productos (list_productos.php).
 - En la columna "STOCK" se puede ver el stock que queda actualmente.

4. Aumentar stock.

- Desde el listado de productos (list_productos.php).
 - En cada producto tiene un botón de "editar" en el cual se puede editar cualquier dato del producto incluido el de STOCK.

- Gestión de admin y usuarios:

1. Admin (Login backend) archivo "login.php".

- Recoger datos del formulario.
 - Obtiene el nombre del usuario (`nombre_admin`) y la contraseña (`password_ingresada`).
 - Usa el `trim()` para eliminar espacios en blanco al inicio/final.
- Consulta la base de datos (tabla de la base de datos "admin").
 - Hacer una consulta segura con `prepare()` y `bind_param()` para que no puedan meter código malicioso y puedan acceder.
 - Busca un admin con el nombre proporcionado.
- Verificar contraseña.
 - Si encuentra el usuario admin, pues compara la contraseña ingresada con el hash almacenado en `password_verify()`.
 - Si coincide pues inicia sesión y te redirige a la página (`index.php`).
 - Si no coincide pues no podrá iniciar sesión.
- Errores.
 - Si el nombre del usuario o la contraseña no coincide pues te muestra un error y te redirige a la página (`login.php`).

2. Usuarios (Login frontend) archivo "login.php".

- Verificación de sesión activa.
 - Si el usuario ya esta iniciado sesión comprueba `$_SESSION["loggedin"] === true`, te redirige a "interfaz.php".
- Validación del formulario.
 - Comprueba si los campos "email" y "contrasena" están vacíos, muestra un mensaje de error `$email_err`, `$contrasena_err`.
- Consulta a la base de datos (tabla de la base de datos "clientes").
 - Usa consulta `mysqli_prepare()` para que no puedan meter código malicioso y puedan acceder.
 - Busca el usuario con el email en la tabla clientes.
- Verificación de la contraseña.
 - Si encuentra el usuario admin, pues compara la contraseña ingresada con el hash almacenado en `password_verify()`.
 - Si coincide inicia sesión y guarda los datos de sesión en `$_SESSION (id, nombre, email`.
 - Si no coincide te muestra un mensaje de error "Email o contraseña incorrectos".

3. Usuarios (Singup frontend) archivo "singup.php".

- **Iniciación de variables.**
 - Declara las variables de los campos del formulario `$nombre`, `$email`, etc. y errores `$nombre_err`, `$email_err`, etc..
- **Validación de campos.**
 - Verifica que los campos obligatorios del formulario no estén vacíos `empty(trim(...))`.
 - Para el campo "email" y "nick", comprueba que los datos no existan en la base de datos.
- **Validación de contraseña.**
 - Asegura que la contraseña tenga al menos 6 caracteres `strlen(trim($_POST["contrasena"])) < 6`.
 - Compara que `$contrasena` y `$confirmar_contrasena` coincidan.
- **Consulta a la base de datos (tabla de la base de datos "clientes").**
 - Usa consulta `mysqli_prepare()` y `bind_param()` para que no puedan meter código malicioso y puedan acceder.
 - Si no hay errores, hashea la contraseña con `password_hash()`.
 - Inserta el usuario.
- **Errores.**
 - Muestra errores en cada campo si no cumplen los requisitos.
 - Si el registro se ha realizado con éxito, estable `$exito = true`.
- **Pedidos:**

Hacer los pasos ya mencionados en el primer punto de "Funcionamiento del sistema", "Carrito de Compra".

Backend

- Obtener el historial de ventas.
 - Consulta SQL para obtener el historial de las ventas.

```
$consulta_historial = "SELECT ventas.id AS IDVenta, clientes.nombre AS
NombreCliente, GROUP_CONCAT(CONCAT(productos.nombre, '
(x', linea_carrito.cantidad, ' ) ',
linea_carrito.precio, '€') ORDER BY productos.nombre
SEPARATOR '<br>') AS ProductosComprados,
ventas.fecha_previstaEntrega AS FechaEntrega,
SUM(linea_carrito.cantidad * linea_carrito.precio) AS
Subtotal, impuestos.porcentaje AS IVA,
SUM(ventas.importe_total) AS Total
FROM clientes
JOIN carrito ON clientes.id = carrito.cliente_id
JOIN linea_carrito ON carrito.id
linea_carrito.carrito_id
JOIN productos ON linea_carrito.producto_id =
productos.id
JOIN ventas ON carrito.id = ventas.carrito_id
JOIN impuestos ON ventas.impuesto_id = impuestos.id
$condiciones
GROUP BY ventas.id, clientes.id, impuestos.porcentaje
ORDER BY clientes.nombre,ventas.fecha_previstaEntrega";
```

- La consulta genera un desglose detallado de todas las ventas incluyendo información del cliente, productos comprados, precios, impuestos y total de la compra, mostrando cada pedido en una tabla y con un botón para descargar una factura simplificada.
- Código donde se genera la tabla.

```
while ($fila = mysqli_fetch_array($listado_historial)) {
    echo "<tr>
    <td>".$fila['IDVenta']. "</td>
    <td>".$fila['NombreCliente']. "</td>
    <td>".nl2br($fila['ProductosComprados']). "</td>
    <td>".$fila['FechaEntrega']. "</td>
    <td>".$fila['Subtotal']. " €</td>
    <td>".$fila['IVA']. "%</td>
    <td>".$fila['Total']. " €</td>
    <td class='descargar-col'> <a
    href='descargar_resumen.php?id=".$fila['IDVenta']. "'>
    <img src='descargar.png' class='bmea'> </a></td> </tr>";
    }.
}
```

Frontend

- Dentro del perfil del usuario se puede visualizar su historial de compras, con la fecha del pedido, productos, total y entrega prevista.
- Obtener el historial de compras.
 - Consulta SQL para obtener el historial de compras.

```
$sql_compras = "SELECT v.id, v.importe_total,
                    v.fecha_previstaEntrega, v.created_at, c.id as
                    carrito_id
                FROM ventas v
                JOIN carrito c ON v.carrito_id = c.id
                WHERE c.cliente_id = ?
                ORDER BY v.created_at DESC";
```

- Hacer una consulta segura con `prepare()` para que no puedan meter código malicioso y puedan acceder.
- Remplaza el “?” de la consulta con el valor de `$cliente_id`.
- Envía la consulta al servidor de la base de datos `$stmt_compras->execute();`.
- Recupera los resultados de la consulta `$stmt_compras->get_result();`.
- Crea un array llamado `$compras = []` donde guardara los datos de las ventas;
- Recorre cada fila y los añade al array `while ($row = $result_compras->fetch_assoc()) {
 $compras[] = $row;
 }`.
- Crear un array para almacenar cada producto `$detalles_compras = [];`
- Va recorriendo sobre el array y va obtenido los valores de “id”, “importe_total” y “carrito_id”.
- Consulta SQL para obtener los detalles de las compras.

```
$sql_detalle = "SELECT p.nombre, p.referencia, lc.cantidad,
                    lc.precio
                FROM linea_carrito lc
                JOIN productos p ON lc.producto_id = p.id
                WHERE lc.carrito_id = ?";
```

- Hacer una consulta segura con `prepare()` para que no puedan meter código malicioso y puedan acceder.
- Remplaza el “?” de la consulta con el valor de `carrito_id`.
- Envía la consulta al servidor de la base de datos `$stmt_detalle->execute();`.
- Recupera los resultados de la consulta `$stmt_detalle->get_result();`.
- Crea un array llamado `$detalle = []` donde guardara los datos de los productos de la compra;
- Recorre cada fila y los añade al array

```
while ($row = $result_compras->fetch_assoc()) {  
    $compras[] = $row;  
}
```

.
- Asocia los productos (`$detalle`) al ID de la venta `$compra['id']`.

- **Interfaz HTML + CSS:**

Nuestras páginas presentan una **distribución de productos bien estructurado**, diseñado para ofrecer una experiencia fluida a los usuarios. La combinación de **HTML + CSS** he ha realizado de manera que sea **intuitiva, estética y funcional**.

1. **Fronted.**

- Todas las páginas siguen con el mismo estilo de CSS que sea de manera **intuitiva, estética y funcional**.
- Páginas inicio.
 - Implementación de un botón de “Contáctanos” que lleva a un formulario.
 - Implementación de botón de “Explorar Catálogo” que lleva a la página principal donde se puede ver los productos y realizar compras.
 - Información sobre quien somos y características que tenemos.
- Página principal.
 - **Navegación clara:** El menú principal que incluye vínculos a "Inicio", "Productos", "Soporte" y facilita un acceso inmediato a todas las áreas.
 - **Barra de inicio y filtros:** Permite la búsqueda de productos concretos a través de la elección de categoría, precio y marca.
 - **Diseño adaptable:** Se ajusta a diversas dimensiones de pantalla, proporcionando una experiencia óptima tanto en ordenadores, móviles y tablet.
- Páginas login, singup y soporte.
 - **Login**
 - Diseño centrado en la pantalla con un formulario compacto.
 - Validación de email y contraseña para garantizar seguridad en el ingreso.
 - Mensajes de error visibles en caso de datos incorrectos.

- Enlace de registro para nuevos usuarios.
- **Singup**
 - Formulario estructurado con campos esenciales: Nombre, Email, Contraseña y Confirmación.
 - Validación de datos antes del envío, asegurando información es correcta.
 - Enlace de inicio de sesión para quienes ya están registrados.
- **Soporte**
 - Formulario estructurado con campos esenciales: Nombre, Email, Asunto y Mensaje.
 - Validación de datos antes del envío, asegurando información correcta.
 - Mensaje de confirmación de que el mensaje se ha enviado correctamente.
- Carrito y checkout.
 - **Carrito**
 - Una ventana flotante que muestra los productos seleccionados previamente.
 - Información de los productos con su precio y cantidad.
 - Puedes aumentar, disminuir y quitar el producto del carrito.
 - Botón para finalizar la compra que le lleva a “Checkout”.
 - **Checkout**
 - Datos personales, para poder realizar la compra tiene que estar registrado, un pequeño formulario para poner tus datos personales (Nombre, Email y Teléfono).
 - Desglose del pedido en una caja a la derecha.
 - Dirección de envío, aquí se colocarán los datos de envío para realizar la entrega (Dirección, Código Postal, Provincia y País).
 - Método de pago, selección de métodos de pago (Tarjeta de crédito, Paypal y Transferencia bancaria).
 - Confirmación del pedido.

2. Backend.

- Todas las páginas siguen con el mismo estilo de CSS que sea de manera **intuitiva, estética y funcional**.
- Login.
 - Diseño centrado en la pantalla con un formulario compacto.
 - Validación de nombre y contraseña para garantizar seguridad en el ingreso.
 - Mensajes de error visibles en caso de datos incorrectos.
- Formulario de productos.
 - Diseño centrado en la pantalla con un formulario compacto.
 - Campos para rellenar de todos los datos que queremos tener de los productos (Referencia, Nombre, Descripción, Precio, Stock, Peso, Alto, Ancho, Largo, Imagen, Categoría, Marca).
 - Validaciones de que todos los campos cumplan con los requisitos.
 - Botón para guardarlo en la base de datos y las imágenes en una carpeta con su nombre.
- Listado de productos.
 - Diseño extendido en toda la pantalla.
 - Una tabla donde se reflejan cada producto con cada parámetro (id, referencia, nombre, descripción, precio, stock, ancho, largo, peso, categoría, marca y acciones).
 - Botones de “editar producto”, “borrar producto”.
 - Filtros en la parte superior (Nombre, Categoría, Marca).
 - Botones de enviar, Exportar XML Y Buscar con Xpath.
- Historial ventas.
 - Diseño extendido en toda la pantalla.
 - Una tabla donde se reflejan cada pedido de cada cliente (id, nombre cliente, productos comprados, fecha de entrega, subtotal, iva, total y descargar).
 - Botón de descargar.
 - Filtro para filtrar por nombre en la parte superior con botón de enviar.
- Listado soporte.
 - Diseño extendido en toda la pantalla.
 - Una tabla donde se reflejan cada incidencia / preguntas (id, nombre, asunto, mensaje, fecha entrega, estado, cliente y acciones).
 - Botones de “en proceso”, “resultado” y “borrar”.
 - Filtro para filtrar por nombre en la parte superior con botón de enviar.

- Validación del formulario (SkyNET/Backend/Productos/Scripts/validar_formulario.js):
 - Formulario productos
 1. **Referencia del producto:**
 - Debe tener entre **2 y 50 caracteres**.
 - Debe tener **letras y números (sin espacios ni símbolos)**.
 2. **Nombre del producto:**
 - Debe tener entre **2 y 100 caracteres**.
 3. **Descripción del producto:**
 - Debe tener entre **2 y 500 caracteres**.
 4. **Precio del producto:**
 - Debe ser un **número mayor que 0**.
 5. **Stock:**
 - Debe ser un **número entero positivo (0 o mayor)**.
 6. **Peso del producto:**
 - Debe ser un **número entero positivo (0 o mayor)**.
 7. **Dimensiones (alto, ancho y largo):**
 - Debe ser un **número entero positivo (0 o mayor)**.
 8. **imagen del producto:**
 - Es **obligatoria**.
 - Debe tener una extensión válida: **.jpg, .jpeg, .png, .webp o .gif**.
 - El tamaño del archivo no debe superar **1 MB**.
 9. **Categorías y Marcas:**
 - Se debe tener una opción seleccionada.
 10. **Envío del formulario:**
 - Si **todas las validaciones** se cumplen, el formulario se envía automáticamente.
 - Si **alguna validación falla**, el formulario **no se envía** y se muestran los errores junto a cada campo.

- **Búsqueda con XPath:**

Extraer datos de la base de datos y los convierte en XML. Filtrar productos por **nombre, categoría y marca** usando las expresiones XPath. Realizar búsquedas rápidas dentro del XML sin hacer muchas consultas SQL.

- **Exportación de productos en XML:**

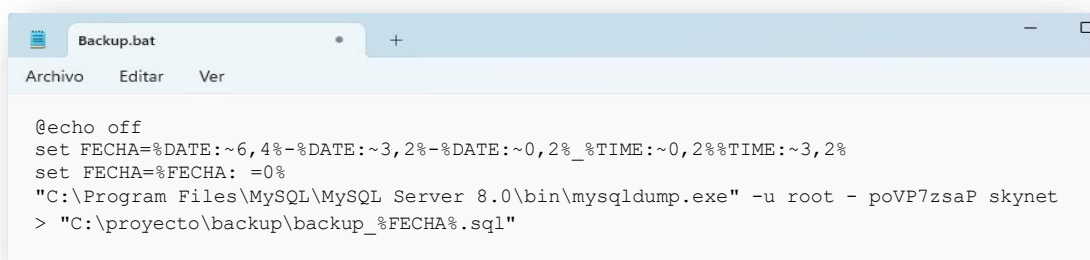
1. **Base de datos consulta:** Los productos son extraídos desde la consulta SQL donde se filtran por nombre, categoría y marca.
2. **Creación de archivo XML:** Un objeto XMLWriter se inicia para abrir el archivo listado_productos.xml.
3. **Generación de estructura XML:** Se crean el XML (producto, nombre, precio, etc.) que representan cada producto de la base de datos.
4. **Escritura de datos XML:** A partir de los resultados de la consulta, se añade cada producto con sus atributos al XML.
5. **Finalización y guardado del XML:** El documento XML se cierra y se almacena en el servidor.
6. **Visibilidad del archivo:** Se da la opción de verlo.

- **Backups automáticos:**

Tener una copia de seguridad es bueno por protección contra pérdidas, recuperación rápida ante fallos y defensa contra ataques de cibernéticos.

- La copia de seguridad se realiza cada sábado de la semana a las 03:00h de la madrugada.
- La configuración e implementación se ha realizado en Windows.
- Implementación.

En el [anexo](#) se incluye enlace a **Backup automático** donde se explica detalladamente como implementar la copia de seguridad de la base de datos en Windows.



```
@echo off
set FECHA=%DATE:~6,4%-DATE:~3,2%-DATE:~0,2%_TIME:~0,2%%TIME:~3,2%
set FECHA=%FECHA: =0%
"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqldump.exe" -u root -pVP7zsaP skynet
> "C:\proyecto\backup\backup_%FECHA%.sql"
```

Ilustración 5. Archivo del Backup -- imagen de elaboración nuestra

8. Algunas de las pruebas.

La parte donde yo he trabajado es en Backend por lo que las pruebas realizadas han sido en Backend, he realizado alguna en Frontend.

Backend

1. Añadir producto y editar producto.

Al rellenar todos los campos y al enviar el formulario las validaciones de JavaScript no funcionaban correctamente.

- **Campos.** (formulario_productos_php).

- Referencia

Tenía que tener al menos una letra y un número, pero no funcionaba bien, metía "1111" y no me saltaba error, ya que quería que al menos tengas una letra y un número.

Solución: (Archivo → verificar_formulario.js)

```
if (!/[a-zA-Z]/.test(referencia)) {  
    console.log("Error: La referencia NO tiene letras.");  
    errorReferencia.textContent = "Debe contener al menos una  
    letra.";  
    control = false;  
}  
  
if (!/[0-9]/.test(referencia)) {  
    console.log("Error: La referencia NO tiene números.");  
    errorReferencia.textContent = "Debe contener al menos un  
    número.";  
    control = false;  
}
```

Comprueba que el campo referencia tenga al menos una letra de a – z
o A –Z [a-zA-Z] minúscula mayúscula, y que tenga al menos un número del 0-9 [0-9]. a través del método "referencia".

- Imagen

No se guardaba la imagen con el nombre que se metía en el campo "Nombre imagen"

Solución:

```
if(isset($_FILES['imagen']) && $_FILES['imagen']['error'] ==  
UPLOAD_ERR_OK) {  
    $target_dir = "../../Frontend/imagenes_productos/";  
    $nombre_imagen_personalizado = $respuesta['nimagen'];  
    $imageFileType = strtolower(pathinfo($_FILES["imagen"]["name"],  
    PATHINFO_EXTENSION));  
    $target_file = $target_dir . $nombre_imagen_personalizado . "." .  
    $imageFileType;  
  
    if (move_uploaded_file($_FILES["imagen"]["tmp_name"], $target_file)) {  
        echo "La imagen se ha subido con el nombre: " .  
        htmlspecialchars($nombre_imagen_personalizado . "." .  
        $imageFileType);  
    }
```

```

    } else {
        echo "Lo sentimos, hubo un error al cargar tu archivo.";
    }
}

```

Este código guarda la imagen a un a carpeta que está en el proyecto en "SkyNET/Frontend/imagenes_productos" con el nombre que pusimos en el formulario \$nombre_imagen_personalizado = \$respuesta['nimagen'];. Usa move_uploaded_file() para guardar la imagen desde la carpeta temporal a su carpeta definitiva.

2. Listado Productos.

- Filtros nombre, categorías y marcas.

Los filtros no funcionaban, intentaba filtrar y me daba un error.

```

$consulta_productos = "SELECT pr.id as id, pr.referencia as referencia,
pr.nombre as nombre, pr.descripcion as descripcion,
pr.precio as precio, pr.stock as stock, pr.alto as
alto, pr.ancho as ancho, pr.largo as largo, pr.peso as
peso, ca.nombre as categoria, m.nombre as marca,
pr.updated_at as fecha_modificacion
FROM productos pr
LEFT JOIN categorias ca ON pr.categoria_id = ca.id
LEFT JOIN marcas m ON m.id = pr.marca_id
ORDER BY ca.nombre ASC";

```

Solución:

Quitar el "ORDER BY ca.nombre ASC" de la consulta y ponerlo debajo de los filtros, ya que cuando utilizaba alguno de los filtros lo que hacía es añadir el WHERE después de la consulta al ejecutar el filtro, el ORDER BY siempre tiene que está detrás del WHERE.

```

$consulta_productos = "SELECT pr.id as id, pr.referencia as referencia, pr.nombre as
nombre, pr.descripcion as descripcion, pr.precio as precio,
pr.stock as stock, pr.alto as alto, pr.ancho as ancho, pr.largo
as largo, pr.peso as peso, ca.nombre as categoria, m.nombre as
marca, pr.updated_at as fecha_modificacion
FROM productos pr
LEFT JOIN categorias ca ON pr.categoria_id = ca.id
LEFT JOIN marcas m ON m.id = pr.marca_id"; //Quitar el ORDER BY

$filtro_nombre='';
$filtro_categoria='';
$filtro_marca='';

if(isset($_REQUEST) && count($_REQUEST)>0) {
    $condiciones = " where true";
    $filtro_nombre = $_REQUEST['nombre'];
    $filtro_categoria = $_REQUEST['categoria'];
    $filtro_marca = $_REQUEST['marca'];

    if ($filtro_nombre){
        $condiciones .= " AND pr.nombre LIKE '%" . $filtro_nombre . "%'";
    }
    if($filtro_categoria > 0){
        $condiciones .= " AND pr.categoria_id = " . $filtro_categoria;
    }
    if($filtro_marca > 0) {
        $condiciones .= " AND pr.marca_id = " . $filtro_marca;
    }
    $consulta_productos .= $condiciones;
}
$consulta_productos .= " ORDER BY ca.nombre ASC";
$listado_productos = mysqli_query( $conexion, $consulta_productos);

```

Frontend

- Barra de navegación de las páginas (interfaz.php, producto.php y perfil.php).

- Barra de navegación

Problema que la barra de navegación no se queda fija al hacer scroll.

Solución:

Añadir estos parámetros al estilo del header.(interfaz.php, producto.php y perfil.php).

`position: fixed;` → Deja la barra fija en la parte superior.

`z-index: 600;` → Se pone la barra por encima de todo menos el menu de carrito de compra.

- Prueba de valor limite.

Comprar más cantidad de un producto de la que tenemos y no me ha dejado por lo que funciona correctamente.

9. Mejoras y Escalabilidad.

Mejoras.

- Funcionamiento de la pestaña “opiniones” de la página de “producto.php”
- Implementar que al generar el documento del historial que en vez de que sea un txt que sea un pdf mejor detallado.
- Implementar que las compras que realice los clientes les llegue un mensaje o correo de que ha realizado esa compra.
- Implementar botón en soporte para poder mandar mensaje al propio cliente y que le salga en su perfil.
- Solucionar problema cuando comprar más de lo que hay en stock te redirige a carrito.php y no tenemos ese archivo.

Escalabilidad.

- Implementación de APIs para ofrecer más formas de pago.
- Implementación con Meta Ads y Google Shopping API.
- Migración a instancias cloud. Para soporte de más usuarios concurrentes (como Black Friday).
- Implementar Redis/Memcached para consultas frecuentes – Reduce un 40% la carga del servidor.

10. Anexos.

- [Análisis Funcional.](#)
- [Manual Usuario \(Gestión tienda\).](#)
- [Manual Usuario \(Usuarios\).](#)
- [Manual Implementación Backup en Windows.](#)
- [Manual Instalación.](#)