

Mi proyecto se trata de un negocio que vende hardware:

1 punto = \$ 1

Descuento por ser Premium = 13% del total de la compra.

Puntos ganados por comprar = 5% del total de la compra.

## ¿COMO FUNCIONA?: - - - - -

- Al iniciar el programa automáticamente se carga el registro de clientes

En la pantalla inicial podrás: Crear un nuevo cliente o cargar uno ya existente, una vez que tengas tu cliente listo, puedes agregar productos al carrito presionando **"Agregar Producto al Carrito"**.

Para borrar un ítem del carrito, solo selecciónalo y presiona **"Borrar"** o **"Borrar Todo"** para vaciar el carrito por completo.

Si deseas gestionar el catalogo de productos solo presiona **"Gestionar Catalogo"**, en cambio si estas conforme con tu compra y quieres finalizarla, solo presiona **"Finalizar Compra"**

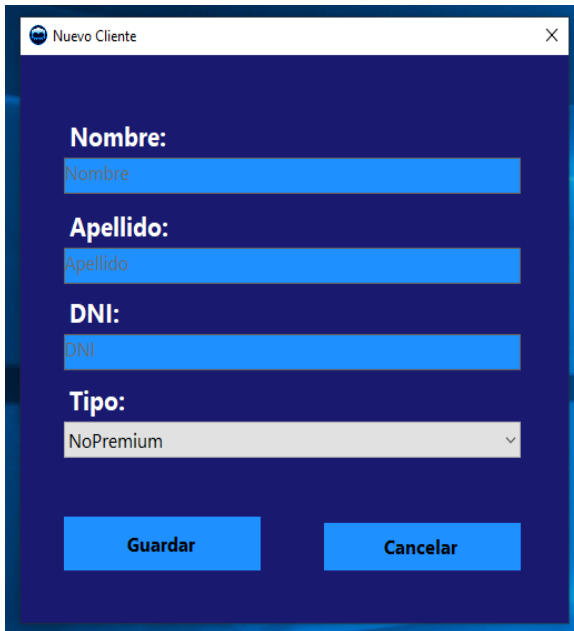
El apartado de **"Historial de compra"** muestra todas las compras que hizo el cliente seleccionado. Se actualiza cada vez que se crea o carga un cliente y cuando se finaliza una compra.

NOTA: El registro de clientes se guarda cuando se confirma que se desea cerrar el programa.

The screenshot displays the application's main interface. At the top, there's a header bar with a 'Inicio' button and a close icon. Below this, a row of four blue buttons is visible: 'Nuevo Cliente', 'Cargar Cliente', 'Agregar Producto al Carrito', and 'Gestionar Catalogo'. Underneath these buttons, there are five input fields labeled 'Nombre:', 'Apellido:', 'DNI:', 'Tipo de Cliente:', and 'Puntos:'. Each field has a corresponding blue button below it: 'Nombre', 'Apellido', 'DNI', 'Tipo', and 'Puntos'. The main area is divided into two sections: 'Carrito:' on the left and 'Historial de compras:' on the right. The 'Carrito:' section has a large empty box below it. The 'Historial de compras:' section has a large empty box below it and a 'Ordenar Por:' dropdown menu to its right. At the bottom left, it shows 'TOTAL : \$ 0'. At the bottom right, there are three blue buttons: 'Borrar', 'Borrar Todo', and 'Salir', followed by a 'Finalizar Compra' button.

### ALTA DE CLIENTE:

En este apartado podrás crear un cliente, siempre y cuando el DNI no pertenezca a otro cliente existente



Formulario "Nuevo Cliente" con los siguientes campos:

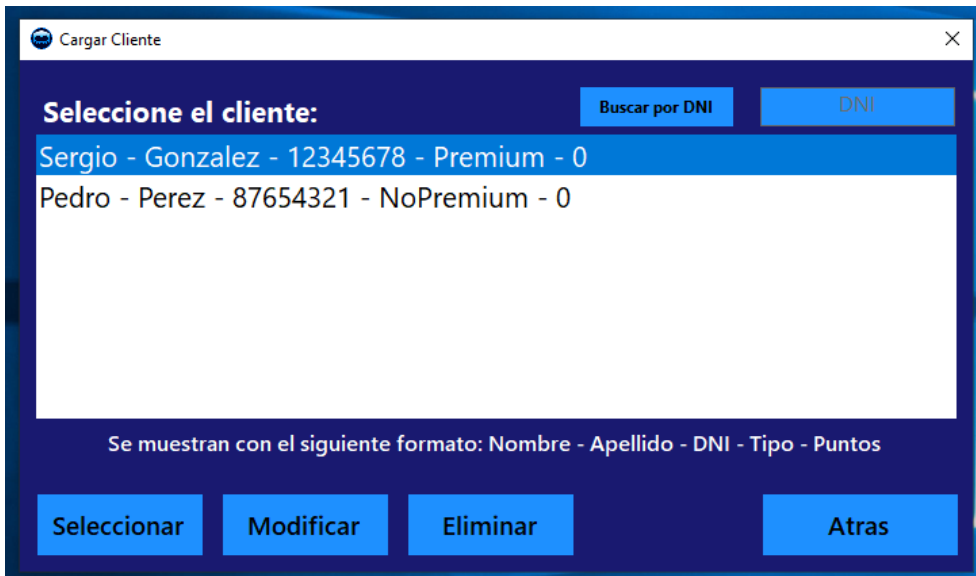
- Nombre:** Campo de texto con el placeholder "Nombre".
- Apellido:** Campo de texto con el placeholder "Apellido".
- DNI:** Campo de texto con el placeholder "DNI".
- Tipo:** Selector de lista desplegable con la opción "NoPremium" seleccionada.

Botones de acción: "Guardar" y "Cancelar".

### BAJA O MODIFICACION DE CLIENTE:

En este apartado podrás seleccionar/Modificar/Eliminar un cliente que ya este seleccionado.

**Nota:** No se puede acceder a este apartado si el historial del cliente se está actualizando o si no hay clientes registrados.



Formulario "Cargar Cliente" con la siguiente estructura:

- Seleccione el cliente:** Encabezado de la lista de clientes.
- Botones de búsqueda:** "Buscar por DNI" y "DNI".
- Lista de clientes:**
  - Sergio - Gonzalez - 12345678 - Premium - 0
  - Pedro - Perez - 87654321 - NoPremium - 0
- Formato de visualización:** Se muestran con el siguiente formato: Nombre - Apellido - DNI - Tipo - Puntos
- Botones de acción:** "Seleccionar", "Modificar", "Eliminar" y "Atras".

## AGREGAR PRODUCTO AL CARRITO:

Al seleccionar un producto se cargan se muestran sus datos.

El comboBox se carga en una tarea distinta para simular una conexión con la base de datos, por lo que, si se presiona “Cancelar” antes de que cargue, se lanza un mensaje.



**Agregar Producto**

**Producto:**  
Ryzen 5 5600x

**Marca:**  
AMD

**Tipo:**  
Procesador

**Precio:**  
200,5

**Aceptar** **Cancelar**

## GESTIONAR CATALOGO:

En esta ventana podrás Agregar/Editar/Eliminar un producto del catálogo. La carga del dataGrid esta en una tarea aparte para simular un retraso con la base de datos.

No se puede agregar/modificar un producto para que tenga el mismo nombre, marca y tipo de otro producto.



**Gestion de Catalogo**

**Catalogo de Productos :** **Ordenar Por :**

Id	Nombre	Marca	Tipo	Precio
1	Ryzen 5 5600x	AMD	Procesador	200,5
2	AB350-m	Gigabyte	MotherBoard	80,7
3	i5 12600KF	Intel	Procesador	180,9
4	RX 6600xt	AMD	GPU	350,9
5	Disco Solido 512gb	Team Group	SSD	120,5
6	Fuente 500w	ThermalTake	PSU	100,25
7	RTX 3060ti	NVIDIA	GPU	450,75
8	Gabinete MATREXX 55	DeepCool	Case	90,78
9	Memoria 8gb 2666mhz	Kingston	RAM	35,24
10	Memoria 16gb 3200mhz	Team Group	RAM	60,66

**Agregar** **Editar** **Eliminar** **Volver**

### FINALIZAR COMPRA:

En este apartado verás el resumen de toda la compra y, además, si el cliente tiene puntos, podrás usarlos para descontarlo al TOTAL.

El valor del TOTAL ya tiene aplicado el descuento por ser Premium, en caso de serlo. También, en amarillo aparecerá el producto más caro de la compra.

Cuando se confirme la compra, se generará automáticamente un ticket (Se genera dentro de la carpeta "Base de datos").

Confirmar Compra

Nombre:

Sergio

Apellido:

Gonzalez

DNI:

12345678

Tipo de Cliente:

Premium

Puntos:

0

Carrito:

Puntos que obtienes al comprar : 27

Nombre	Marca	Tipo	Precio
Disco Solido 512gb	Team Group	SSD	120,5
Ryzen 3 3200g	AMD	Procesador	130,45
RTX 3050	Zotac	GPU	300,23

El producto mas caro es: RTX 3050 - Zotac - GPU - \$ 300,23

TOTAL : \$ 479,53

Descuento : \$ 71,65

Usar Puntos

Confirmar

Cancelar

¿DONDE USO LOS TEMAS?: -----

**SQL:** El script para crear la base de datos está en la carpeta “Base de datos” ubicada en la misma ruta que la solución del proyecto.

**CONEXIÓN A BASE DE DATOS:** En “BD\_Productos.cs” \*\*\*\*\*

```
0 referencias
static BD_Productos()
{
    BD_Productos.conexion = "Server=.;Database=BD_Productos;Trusted_Connection=True;";
}
/// <summary>
/// Lee la base de datos y retorna una lista de todos los productos leídos
/// </summary>
/// <returns></returns>
3 referencias
public static List<Producto> ObtenerCatalogoProductos()
{
    try
    {
        List<Producto> catalogo = new List<Producto>();
        string query = "select * from CatalogoProductos";
        using (SqlConnection connection = new SqlConnection(BD_Productos.conexion))
        {
            SqlCommand cmd = new SqlCommand(query, connection);
            connection.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                int id = reader.GetInt32(0);
                string nombre = reader.GetString(1);
                string marca = reader.GetString(2);
                string tipo = reader.GetString(3);
                float precio = (float)reader.GetDouble(4);
                Producto producto = new Producto(id, nombre, marca, tipo, precio);
                catalogo.Add(producto);
            }
        }
        return catalogo;
    }
    catch (Exception)
    {
        throw new ErrorBaseDatosException("Ocurrió un error al cargar el catalogo de productos");
    }
}
```

```
/// <summary>
/// Agrega el producto recibido por parametros, a la base de datos.
/// </summary>
/// <param name="producto"></param>
1 referencia
public static void AgregarProducto(Producto producto)
{
    try
    {
        string query = "insert into CatalogoProductos (nombre, marca, tipo, precio) values (@nombre, @marca, @tipo, @precio)";
        using (SqlConnection connection = new SqlConnection(BD_Productos.conexion))
        {
            SqlCommand cmd = new SqlCommand(query, connection);
            connection.Open();
            cmd.Parameters.AddWithValue("nombre", producto.Nombre);
            cmd.Parameters.AddWithValue("marca", producto.Marca);
            cmd.Parameters.AddWithValue("tipo", producto.Tipo);
            cmd.Parameters.AddWithValue("precio", producto.Precio);
            cmd.ExecuteNonQuery();
        }
    }
    catch (Exception)
    {
        throw new ErrorBaseDatosException("Ocurrió un error al agregar el producto");
    }
}
```

```

/// <summary>
/// Actualiza los datos del producto recibido, en la base de datos.
/// </summary>
/// <param name="producto"></param>
1 referencia
public static void ActualizarProducto(Producto producto)
{
    try
    {
        string query = "update CatalogoProductos set nombre=@nombre , marca=@marca , tipo=@tipo , precio=@precio where id=@id";
        using (SqlConnection connection = new SqlConnection(BD_Productos.conexion))
        {
            SqlCommand cmd = new SqlCommand(query, connection);
            connection.Open();
            cmd.Parameters.AddWithValue("id", producto.Id);
            cmd.Parameters.AddWithValue("nombre", producto.Nombre);
            cmd.Parameters.AddWithValue("marca", producto.Marca);
            cmd.Parameters.AddWithValue("tipo", producto.Tipo);
            cmd.Parameters.AddWithValue("precio", producto.Precio);
            cmd.ExecuteNonQuery();
        }
    }
    catch (Exception)
    {
        throw new ErrorBaseDatosException("Ocurrió un error al actualizar el producto");
    }
}

```

```

/// <summary>
/// Elimina el producto recibido por parametro, de la base de datos.
/// </summary>
/// <param name="producto"></param>
1 referencia
public static void EliminarProducto(Producto producto)
{
    try
    {
        string query = "delete from CatalogoProductos where id=@id";
        using (SqlConnection connection = new SqlConnection(BD_Productos.conexion))
        {
            SqlCommand cmd = new SqlCommand(query, connection);
            connection.Open();
            cmd.Parameters.AddWithValue("id", producto.Id);
            cmd.ExecuteNonQuery();
        }
    }
    catch (Exception)
    {
        throw new ErrorBaseDatosException("Ocurrió un error al eliminar el producto");
    }
}

```

```

/// <summary>
/// Lee la base de datos y la ordena de acuerdo al criterio especificado y retorna una lista con su contenido.
/// </summary>
/// <param name="criterio"></param>
/// <returns></returns>
8 referencias
public static List<Producto> ObtenerCatalogoProductosOrdenado(string criterio)
{
    List<Producto> catalogo = new List<Producto>();
    string query = $"select * from CatalogoProductos order by {criterio}"; //Intenté usar el @criterio pero me tiraba exception siempre
    using (SqlConnection connection = new SqlConnection(BD_Productos.conexion))
    {
        SqlCommand cmd = new SqlCommand(query, connection);
        connection.Open();
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            int id = reader.GetInt32(0);
            string nombre = reader.GetString(1);
            string marca = reader.GetString(2);
            string tipo = reader.GetString(3);
            float precio = (float)reader.GetDouble(4);
            Producto producto = new Producto(id, nombre, marca, tipo, precio);
            catalogo.Add(producto);
        }
    }
    return catalogo;
}

```

```

/// <summary>
/// Retorna una lista ordenada segun el criterio recibido por parametros
/// </summary>
/// <param name="dato"></param>
/// <returns></returns>
1 referencia
public static List<Producto> OrdenarCatalogoProductos(EDatosProducto dato)
{
    try
    {
        List<Producto> catalogo = null;
        switch (dato)
        {
            case EDatosProducto.NombreAsc:
                catalogo = ObtenerCatalogoProductosOrdenado("nombre asc");
                break;
            case EDatosProducto.NombreDesc:
                catalogo = ObtenerCatalogoProductosOrdenado("nombre desc");
                break;
            case EDatosProducto.MarcaAsc:
                catalogo = ObtenerCatalogoProductosOrdenado("marca asc");
                break;
            case EDatosProducto.MarcaDesc:
                catalogo = ObtenerCatalogoProductosOrdenado("marca desc");
                break;
            case EDatosProducto.TipoAsc:
                catalogo = ObtenerCatalogoProductosOrdenado("tipo asc");
                break;
            case EDatosProducto.TipoDesc:
                catalogo = ObtenerCatalogoProductosOrdenado("tipo desc");
                break;
            case EDatosProducto.PrecioAsc:
                catalogo = ObtenerCatalogoProductosOrdenado("precio asc");
                break;
            case EDatosProducto.PrecioDesc:
                catalogo = ObtenerCatalogoProductosOrdenado("precio desc");
                break;
        }
        return catalogo;
    }
    catch (Exception)
    {
        throw new ErrorBaseDatosException("Ocurrió un error al ordenar los productos");
    }
}

```

## DELEGADOS: En "BD\_Productos.cs" \*\*\*\*\*

```
/// <summary>
/// Busca en la base de datos si el producto ya existe en la base de datos, en caso de que exista, invoca el Action recibido por parametro
/// </summary>
/// <param name="producto"></param>
/// <param name="accion"> </param>
/// <returns></returns>
1 referencia
public static bool VerificarSiExiste(Producto producto, Action accion)
{
    try
    {
        bool resultado = false;
        string query = "select id from CatalogoProductos where nombre=@nombre and marca=@marca and tipo=@tipo";
        using (SqlConnection connection = new SqlConnection(BD_Productos.conexion))
        {
            SqlCommand cmd = new SqlCommand(query, connection);
            connection.Open();
            cmd.Parameters.AddWithValue("nombre", producto.Nombre);
            cmd.Parameters.AddWithValue("marca", producto.Marca);
            cmd.Parameters.AddWithValue("tipo", producto.Tipo);
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                int id = reader.GetInt32(0);
                if (id > -1)
                {
                    resultado = true;
                }
            }
            if (resultado && accion is not null)
            {
                accion.Invoke();
            }
        }
        return resultado;
    }
    catch (Exception)
    {
        throw new ErrorBaseDatosException("Ocurrió un error al verificar si existe el producto");
    }
}
```

```
public delegate bool DelegateVerificarSiExiste(Producto producto, Action accion);
```

```
DelegateVerificarSiExiste delegadoVerificarSiExiste = BD_Productos.VerificarSiExiste;
if (!delegadoVerificarSiExiste(producto, MensajeYaExisteElProducto))
{
    this.DialogResult = DialogResult.OK;
}
```



## HILOS (TASK): En "Tareas.cs" \*\*\*\*\*

```
private async void FrmGestionarCatalogo_Load(object sender, EventArgs e)
{
    try
    {
        dtagrdrCatalogo.Enabled = false;
        dtagrdrCatalogo.DataSource = await Tareas.ActualizarCatalogo_task(cts.Token);
        imgCargar.Visible = false;
        dtagrdrCatalogo.Enabled = true;
    }
}
```

```
public static class Tareas
{
    /// <summary>
    /// Lee la informacion de la base de datos en un hilo distinto y retorna una lista con el contenido en caso de que no se cancele la tarea.
    /// </summary>
    /// <param name="cancellationToken"></param>
    /// <returns></returns>
    1 referencia
    public static async Task<List<Producto>> ActualizarCatalogo_task(CancellationToken cancellationToken)
    {
        List<Producto> catalogo = await Task.Run(() =>
        {
            Thread.Sleep(2000);
            if (cancellationToken.IsCancellationRequested)
            {
                throw new TaskCanceledException();
            }
            return BD_Productos.ObtenerCatalogoProductos();
        });
        return catalogo;
    }
}
```

## Pequeños task en "FrmPrincipal.cs"

```
3 referencias
private async void Actualizar(Cliente cliente, CancellationToken cancellationToken)
{
    this.ActualizarInformacionCliente(cliente);
    this.lstCarrito.Items.Clear();
    this.vnta.Carrito.Clear();
    this.AcualizarTotalCarrito();
    this.imgHistorial.Visible = true;
    this.lstHistorialCompras.Enabled = false;
    await Task.Run(() =>
    {
        Thread.Sleep(4000);
        if (cancellationToken.IsCancellationRequested)
        {
            throw new TaskCanceledException();
        }
    });
    this.ActualizarHistorial(cliente.HistorialDeCompra);
    this.lstHistorialCompras.Enabled = true;
    this.imgHistorial.Visible = false;
}
```

## Y “FrmAgregarProducto.cs”

```
1 referencia
private async void FrmAgregarProducto_Load(object sender, EventArgs e)
{
    try
    {
        cmbProducto.Enabled = false;
        lista = await Task.Run(()=>
        {
            Thread.Sleep(2000);
            if (cts.Token.IsCancellationRequested)
            {
                throw new TaskCanceledException();
            }
            return BD_Productos.ObtenerCatalogoProductos();
        });
        foreach (Producto item in lista)
        {
            cmbProducto.Items.Add(item.Nombre);
        }
        cmbProducto.Enabled = true;
    }
}
```

**EVENTOS:** En “Ventas.cs” \*\*\*\*\*

```
public delegate void EmitirTicketHandler(Ventas sender, VentaEventArgs e);

public event EmitirTicketHandler EventoTicket;
```

```
1 referencia
public void EjecutarEmisionDeTicket(string contenido)
{
    VentaEventArgs e = new VentaEventArgs();
    e.Contenido = contenido;
    if(EventoTicket is not null)
    {
        EventoTicket(this, e);
    }
}
```

## Y en “FrmFinalizarCompra”

```
1 referencia
public void EmitirTicketRecibido(Ventas sender, VentaEventArgs e)
{
    string path = $"../../../../../Base de datos/Ticket_{sender.Usuario.Nombre}_{sender.Usuario.Apellido}-{DateTime.Now.ToString("HH-mm-ss")} + ".txt";
    try
    {
        using (StreamWriter sw = new StreamWriter(path))
        {
            sw.WriteLine($"***** Este es un Ticket de compra ***** \n");
            sw.WriteLine(((IMostrar)sender).MostrarInformacion());
            sw.WriteLine("Fecha: " + DateTime.Now.ToString());
            sw.WriteLine($"Total: $ {sender.TotalCarrito}");
            sw.WriteLine($"Descuento aplicado: $ {sender.Descuento}");
            sw.WriteLine($"Puntos gastados: {this.puntosGastados}");
            sw.WriteLine($"Puntos obtenidos por comprar: {this.puntosObtenidos}");
            sw.WriteLine($"***** {e.Contenido} *****");
        }
    }
}
```

```
venta.EventoTicket += this.EmitirTicketRecibido;
```

**METODOS DE EXTENSIÓN:** En “VentasExtendido.cs” \*\*\*\*\*

Nota: Sé que el uso mas apropiado para las extensiones es para agregar funcionalidad a las clases que no tenemos acceso como String, Int, etc. Pero decidí hacer una extensión de mi clase Venta para agregarle una función más útil.

```
public static class VentaExtendido
{
    /// <summary>
    /// Calcula la cantidad de puntos que obtiene el usuario
    /// </summary>
    /// <param name="venta"></param>
    /// <returns></returns>
    1 referencia
    public static int CalcularPuntos(this Ventas venta)
    {
        return (int)(venta.TotalCarrito * (float)0.05);
    }
    /// <summary>
    /// Resta los puntos del usuario al total del carrito y retorna el resultado
    /// </summary>
    /// <param name="venta"></param>
    /// <returns></returns>
    2 referencias
    public static float UsarPuntosDelCliente(this Ventas venta)
    {
        return venta.TotalCarrito - venta.Usuario.Puntos;
    }
}
```

```
/// <summary>
/// Retorna la descripcion del producto mas caro encontrado en la lista recorrida
/// </summary>
/// <param name="venta"></param>
/// <returns></returns>
1 referencia
public static string ObtenerDescripcionProductoMasCaro(this Ventas venta)
{
    Producto producto = null;
    float precioMasAlto = 0;
    bool flag = true;
    venta.Carrito.ForEach(x =>
    {
        if (flag || precioMasAlto < x.Precio)
        {
            precioMasAlto = x.Precio;
            producto = x;
            flag = false;
        }
    });
    return $"El producto mas caro es: {producto.Nombre} - {producto.Marca} - {producto.Tipo} - $ {producto.Precio}";
}
```