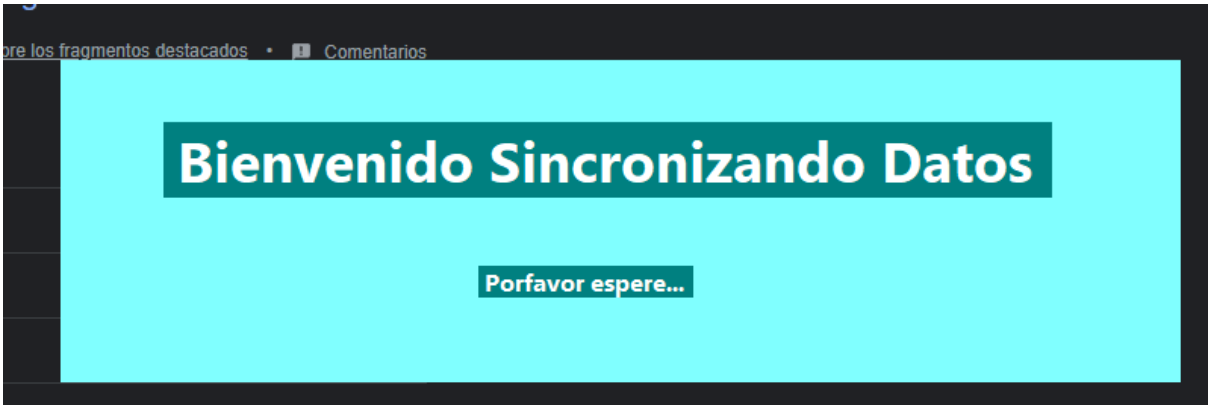
Eduardo Andres Sosa Segovia 2A

Empresa de comercio que vende arte digital. Venden cantidades limitadas en eventos físicos. La anfitriona se encarga de facturar dependiendo de si el cliente es afiliado, monotributistas ,etc.

Al iniciar se utilizan hilos para cargar los productos de la base de datos y se muestra una pantalla de carga

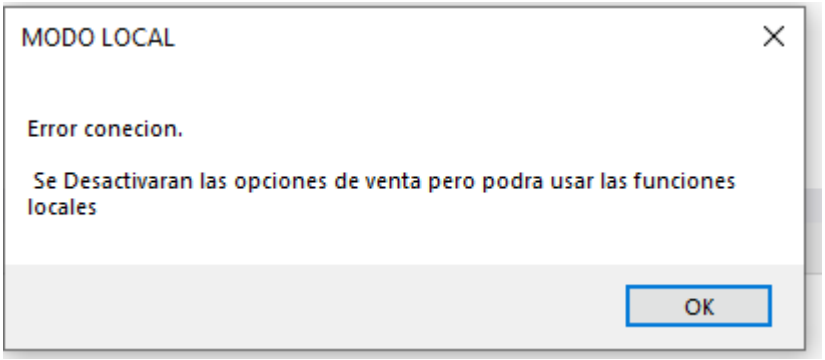


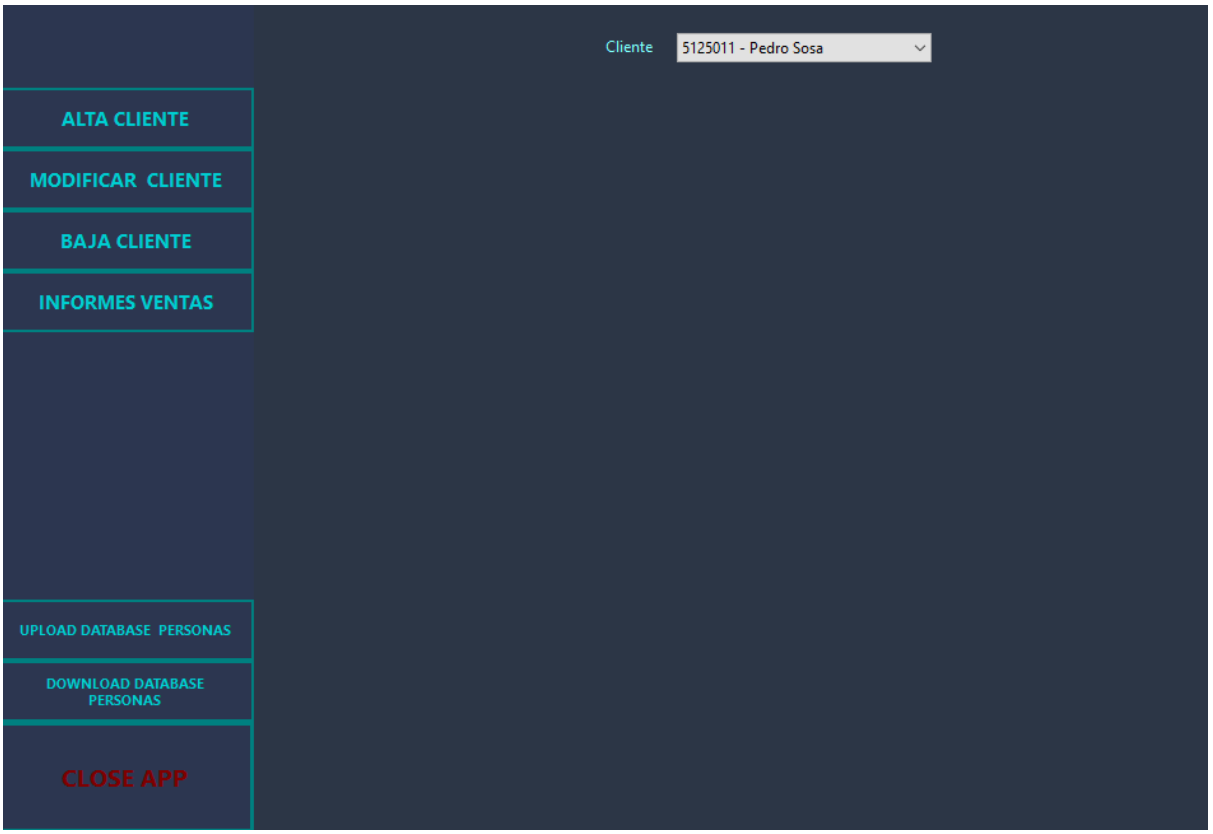

```
/// <summary>
/// Simula la carga inicial de los productos traidos desde la BD una vez completa
/// </summary>
1 reference
private void CargarProductosBD()
{
    FrmApertura frmLoading = new FrmApertura();
    frmLoading.Show();
    Action action = new Action (()=> frmLoading.Refresh());
    action.Invoke();

    int index = 0;
    int productoValido = 0;
    foreach (Producto item in listaProductos)
    {
        if (item.ProductoConStock())
        {
            Thread hiloActualizacion = new Thread(()=>SetOneProducto(item, index));
            hiloActualizacion.Start();
            Thread.Sleep(500);
            productoValido++;
        }
        index++;
    }

    frmLoading.Close();
}
```

```
/// <summary>
/// Inicia el setup de un producto a uno de los 6 slots correspondiente para display del mismo
/// </summary>
/// <param name="unProducto"></param>
/// <param name="posicion"></param>
3 references
private void SetOneProducto(Producto unProducto, int posicion)
{
    if (this.InvokeRequired)
    {
        Action d = new Action(()=>SetOneProducto(unProducto,posicion));
        this.Invoke(d);
    }
    else
    {
        if (posicion < listaProductos.Count && posicion > -1 &&  unProducto!=null)
        {
            unProducto = listaProductos[posicion];
            listaGroupBoxProductos[posicion].Text = unProducto.Nombre;
            listaLabelsRareza[posicion].Text = unProducto.Rareza.ToString();
            listaLabelsPrecio[posicion].Text = "$" + unProducto.Price.ToString();
            listaLabelsCantidad[posicion].Text = "Cant: " + unProducto.Cantidad.ToString();
            listaGroupBoxProductos[posicion].Visible = true;
        }
    }

}
```

si la conexión es exitosa se muestra el menú completo.

Si sucede error al intentar conectarse a base de datos se disparará un evento y se mostrará una interfaz con solo las funciones locales activas.






```
/// <summary>
/// Clase principal para el control de base de datos de los productos.
/// Se puede modificar, la carga sencilla esta inaccesible al usuario se usa para actualizar cuando se modifique y la baja se realiza de forma logica por lo que no la usamos en el proyecto.
/// </summary>
12 references
public class ProductoDAO
{
    static string cadenaConexion;
    static SqlCommand comando;
    static SqlConnection conexion;

    //Delegado Eventos Cuando falla la conexion al servidor la app entra en modo local y solo se pueden usar las funciones de archivos
    public delegate void FalloDB();
    public static event FalloDB OnFalloConexionDataBase;
```

```
/// <summary>
/// Lee la bade de datos y retorna una lista con todos los productos que hay en esta
/// </summary>
/// <returns></returns>
3 references
public static List<Producto> Leer()
{
    List<Producto> productos = new List<Producto>();

    try
    {
        conexion.Open();
        comando.CommandText = $"SELECT * FROM PRODUCTOS";
        SqlDataReader dataReader = comando.ExecuteReader();

        while (dataReader.Read())
        {
            productos.Add(new Producto
                          (
                          Convert.ToDouble(dataReader["PRECIO"]),
                          Convert.ToInt32(dataReader["ID"]),
                          dataReader["NOMBRE"].ToString(),
                          Convert.ToInt32(dataReader["CANTIDAD"]),
                          Convert.ToInt32(dataReader[("RAREZA")])
                          ));
        }

        dataReader.Close();
    }
    catch (Exception)
    {

        OnFalloConexionDataBase();
    }
    finally
    {
        conexion.Close();
    }

    return productos;
}
```

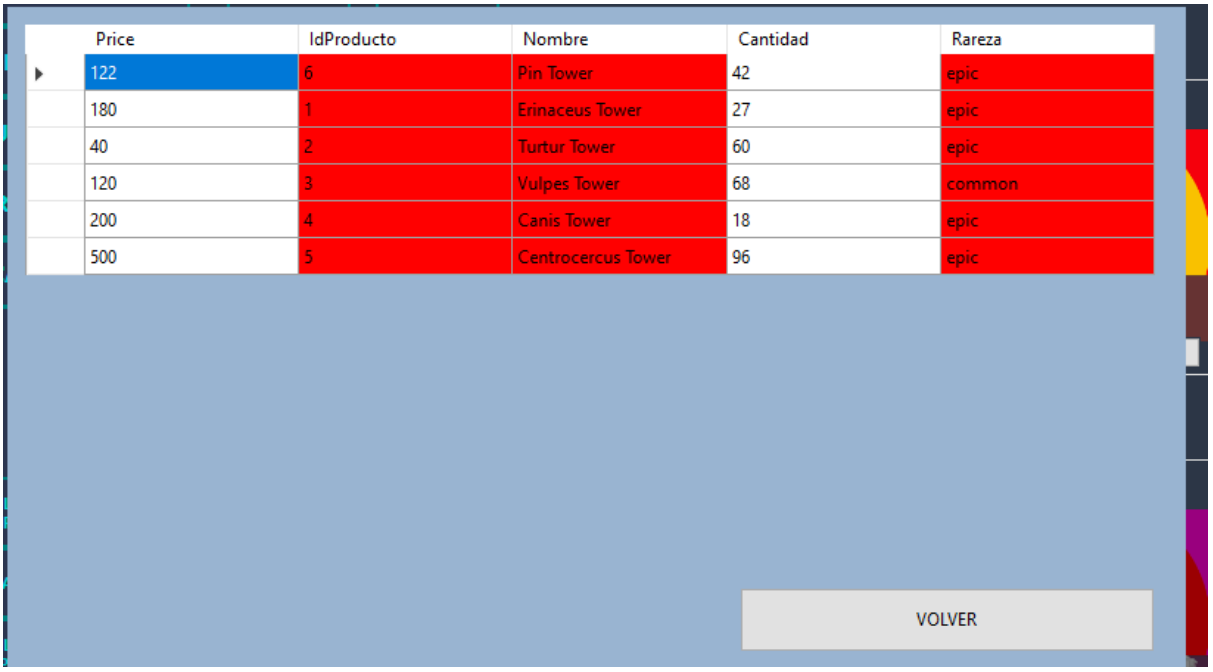
En el Frm Principal se captura el evento

```
public partial class EstoNoEsCompraGamer : Form
{
```

```
/// <summary>
/// Captura el evento cuando falla la carga de la base de datos por conexion
/// </summary>
1 reference
private void BackUpData_EventHandler()
{
    string message = "Error conecion. \n\n Se Desactivaran las opciones de venta pero podra usar las funciones locales";
    string title = "MODO LOCAL";
    MessageBoxButtons buttons = MessageBoxButtons.OK;
    DialogResult result = MessageBox.Show(message, title, buttons);
    btnClearCar.Hide();
    grbCart.Hide();
    btnVentasCarrito.Hide();
    btnModificarProductos.Hide();
    btnDownloadProductos.Hide();
}

1 reference
private void btnModificarProductos_Click(object sender, EventArgs e)
{
    if(listaProductos.Count>0)
    {
        Carrito.VaciaCarrito(listaProductos);
        FrmModificarProductos frmModificarProductos = new FrmModificarProductos(listaProductos);
        frmModificarProductos.ShowDialog();
        listaProductos = ProductoDAO.Leer();
        ActualizarListadoProductos();
    }
    else
    {
        MessageBox.Show("Asegurece de estar conectado al servidor", "Error Lista de productos Vacia", MessageBoxButtons.YesNo);
    }
}
```

El botón Modificar productos.

Trae del servidor la lista de producto y los muestra en un grid, se bloquean ciertas características que controla la sede principal y solo permite modificar el precio y cantidad de los productos. Las cerdas editables son las únicas editables y sin sombreado rojo.

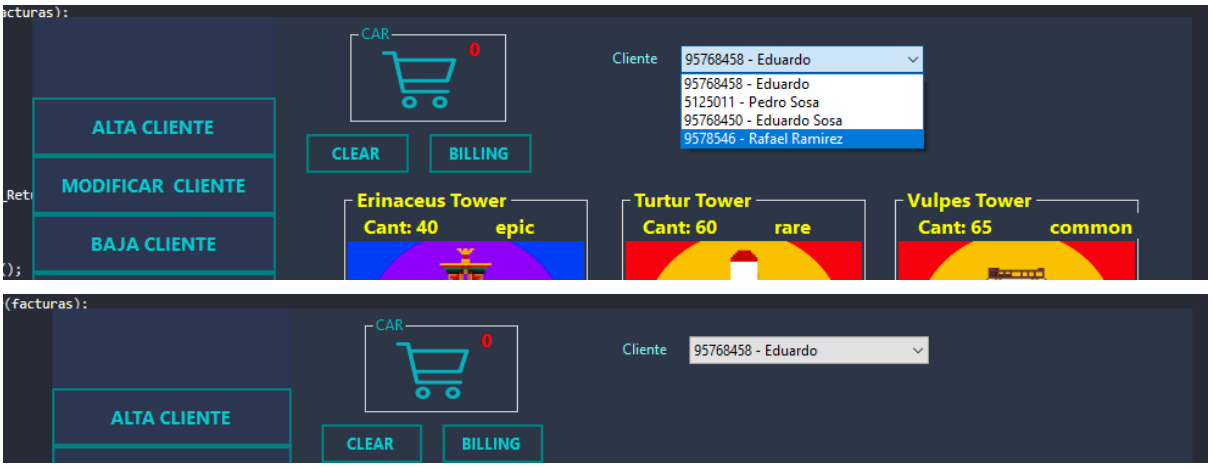


```
1 reference
private void dtgProductos_CellEndEdit(object sender, DataGridViewCellEventArgs e)
{

    if(!inputInvalid(this.dtgProductos.CurrentCell.Value))
    {
        dtgProductos[e.ColumnIndex, e.RowIndex].Value = "0";
    }
    stock = (List<Producto>)dtgProductos.DataSource;
    ProductoDAO.ActualizarProductos((List<Producto>)dtgProductos.DataSource);


}
```

```
1 reference
private void dtgProductos_DataError(object sender, DataGridViewDataErrorEventArgs e)
{
    if(e.ColumnIndex==3)
    {
        MessageBox.Show("Solo debe ingresar numero enteros en cantidad \nCorrija error", "ERROR INPUT", MessageBoxButtons.OK,MessageBoxIcon.Warning);

    }
    else
    {
        MessageBox.Show("Solo debe ingresar numero \nCorrija error", "ERROR INPUT", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}

/// <summary>
/// Analisa los inpust del usuario y si no son numeros positivos devuelve false
/// </summary>
/// <param name="data"></param>
/// <returns></returns>
1 reference
private bool inputInvalid(object data)
{
    bool retorno = false;
    if(data is Int32)
    {
        if((Int32)data<0)
        {

            retorno = false;
        }
        else
        {
            retorno = true;
        }
    }
    if (data is double)
    {
        if ((double)data < 0)
        {
            retorno = false;
        }
        else
        {
            retorno = true;
        }
    }

    return retorno;
}
```

```
1 reference
private void RestricionesDataGrid()
{
    dtgProductos.Columns[1].ReadOnly = true;
    dtgProductos.Columns[1].DefaultCellStyle.BackColor = Color.Red;
    dtgProductos.Columns[2].ReadOnly = true;
    dtgProductos.Columns[2].DefaultCellStyle.BackColor = Color.Red;
    dtgProductos.Columns[4].ReadOnly = true;
    dtgProductos.Columns[4].DefaultCellStyle.BackColor = Color.Red;


}
```

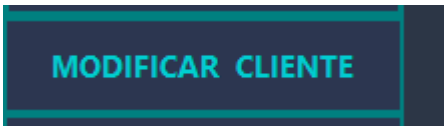
El cliente debe ser elegido con un dropbox entre la lista de clientes activos para facturar lo que vaya sumando al carrito.



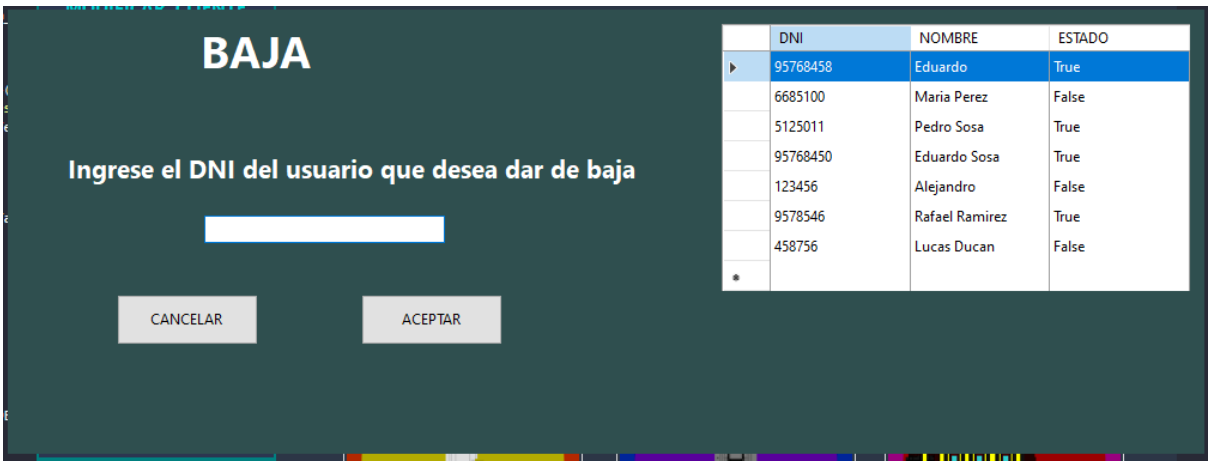
Si el cliente no está en la lista se podrá dar un alta para agregarlo y posteriormente aparecerá en el droplist.

Modificar cliente

te permite cambiar todos los campos permitidos menos el dni incluso reactivar clientes previamente dados de bajo o incluso darlo de baja a clientes activos

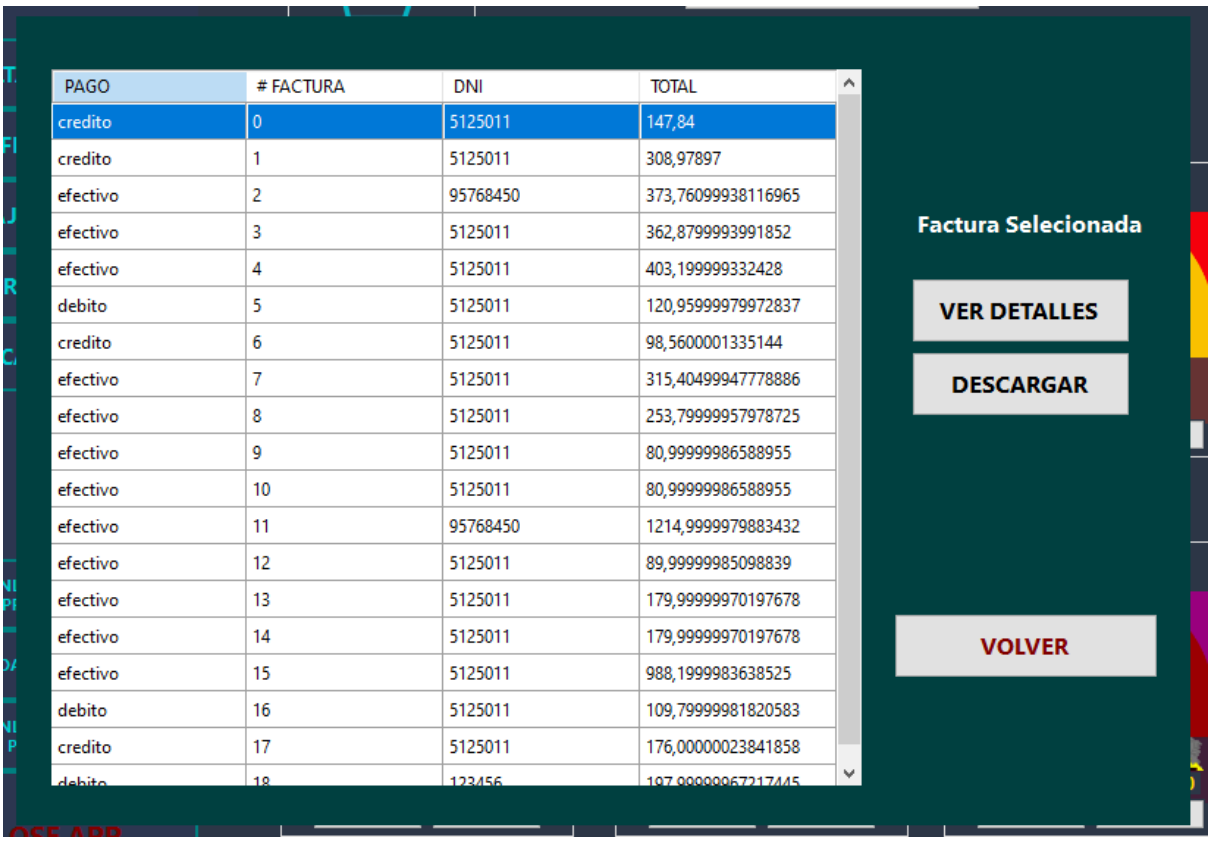


Baja Cliente te permite dar de baja de forma mas rapida solo ingresando el DNi del cliente



Informes sales te permite:

Ver todas las facturas previas y se puede ver la selecionada en detalle o descargarla



Download te permite descargar la base de datos con la que se está trabajando actualmente en una carpeta de elección.

El programa funciona con una base de datos default para facturas.

Baja lógica. Si un producto se queda sin unidades este se apaga.

```
15 references
private void ActualizarListadoProductos()
{

    int index = 0;
    int productoValido = 0;
    foreach (Producto item in listaProductos)
    {
        if (item.ProductoConStock())
        {
            SetOneProducto(item, index);
            productoValido++;
        }
        else if(!item.ProductoConStock())
        {
            listaGroupBoxProductos[index].Visible = false;
        }
            index++;
    }


}
/// <summary>
/// Simula la carga inicial de los productos traidos desde la BD una vez completa
/// </summary>
1 reference
private void CargarProductosBD()
{
    FrmApertura frmLoading = new FrmApertura();
    frmLoading.Show();
    Action action = new Action (()=> frmLoading.Refresh());
    action.Invoke();

    int index = 0;
    int productoValido = 0;
    foreach (Producto item in listaProductos)
    {
        if (item.ProductoConStock())
        {
            Thread hiloActualizacion = new Thread(()=>SetOneProducto(item, index));
            hiloActualizacion.Start();
            Thread.Sleep(500);
            productoValido++;
        }
        index++;
    }

    frmLoading.Close();
}
```

Tema	Nombre	Usando En
10	Excepciones	En la interfaz de serialización En varios pero en FrmMain principalmente para manejo de notificación de errores formatos de archivos
11	Pruebas Unitarias	Se realizaron pruebas de funcionalidad de clase Factura y de la clase Persona
12	Tipos Genericos	La serialización y el manejo de archivos Se usó una lista Genérica para manejar clientes de distintas clases y tener que hacer bajas y modificaciones más complejas que una baja lógica
13	Interfaces	Implementado la clase Factura para determinar cálculo de tributación dependiendo del tipo de cliente
14	Archivos	Se usó para la realización de Impresiones de Facturas Principalmente y exportación de estas en archivos de texto
15	Serialización	List<Persona> (Herencia) con upload y download en .xml List<Producto> (!Herencia) con upload y download en .xml y .Json List<Facturas> Base de datos que controla Id autoIncremental

16-17	SQL	ProductoDao carga inicial de productos y modificación. La baja se maneja de forma lógica en otras clase
17	Delegados función lambda	Usados en el lanzamiento de eventos principalmente. Action en el form principal al cargar los datos y la pantalla de carga CargarProducto(void) SetProducto(producto,int) ProductoDao para lanzar el evento de OnFalloConexion
18	Hilos	Usado para lanzar simular una pantalla de carga mientras se carga los productos de la base de datos SQL CargarProductosBD()
19	Eventos	ProductoDao para lanzar el evento de OnFalloConexion
20	Método Extension	Usado para agregar más funcionalidad al Producto y realizar la baja lógica con un método que devuelve true o false si hay stock private void ActualizarListadoProductos() private void CargarProductosBD()