

TAREA CLASE 8

CAPACITACIÓN C#

 **Ezequiel Remus**
ezequielremus@gmail.com

9 de abril de 2023

RESUMEN

En este artículo se encuentra la resolución a la Tarea de la clase n°8 del **Bootcamp de C#**. En referencias se puede encontrar un link que lleva al código de la tarea.

1. Cambio de estructura de Proyecto

1.1. Creamos el dll Código Común

Primero lo que vamos a hacer es *crear un proyecto DLL llamado Código Común*

Para esto nos paramos en nuestra solución, hacemos clic derecho vamos a agregar y dentro de agregar a *nuevo proyecto*. Nos saldrá una ventana como la que sigue

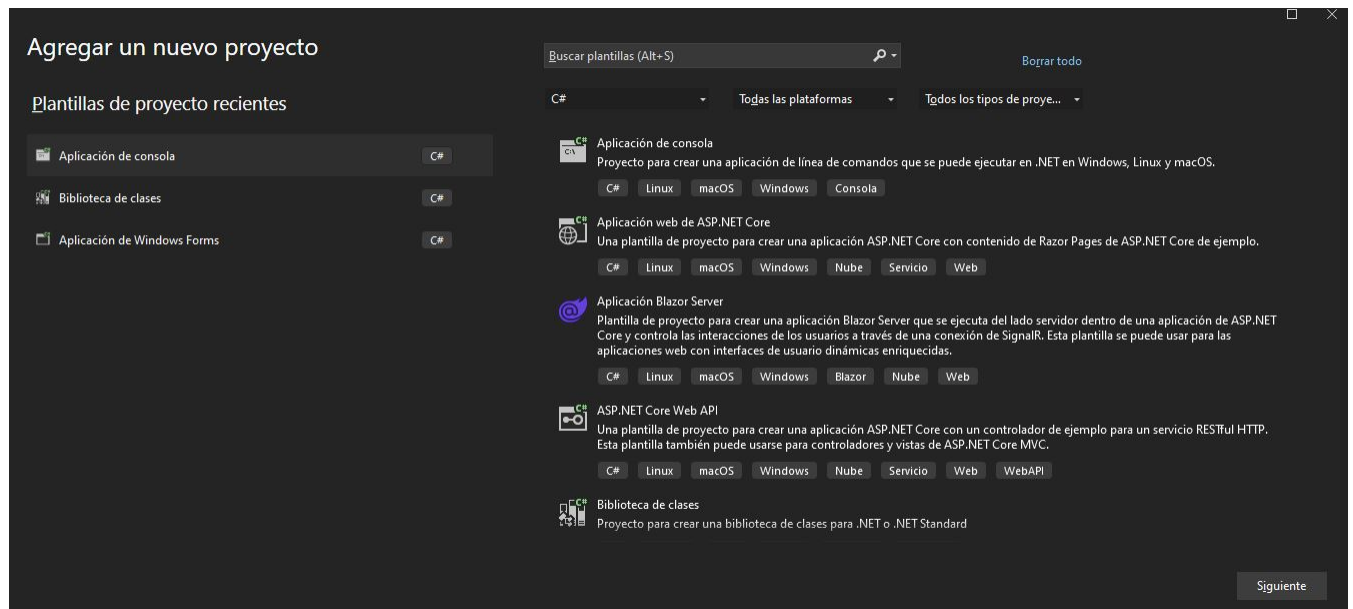


Figura 1: Ventana de creación de proyectos

Vamos a la opción de crear biblioteca de clases y creamos el proyecto con el nombre *Código Común*. Nos quedara entonces estructurado el proyecto de la siguiente manera Ahora, una vez creado el proyecto, movemos la carpeta datos

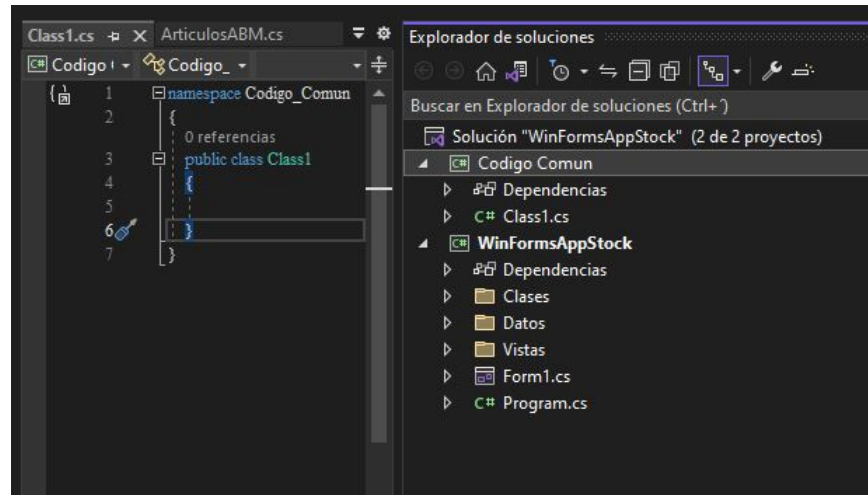


Figura 2: Estructura inicial

hacia la dll y eliminamos la carpeta datos del proyecto de windowsForm, quedandonos entonces la siguiente estructura. Ahora, para que esto realmente funcione, necesitamos realizar una serie de pasos. El primero y fundamental es el

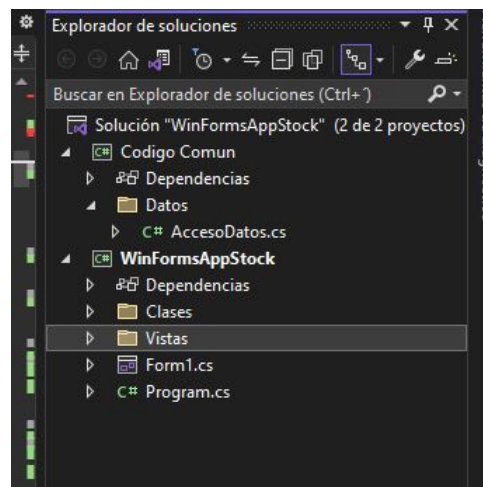


Figura 3: Muevo los datos a codigo comun

instalar las dependencias. Notemos que nosotros en la carpeta de datos trabajamos con conexiones sql, por lo que debemos instalar el **paquete nugget** adecuado. En este caso, el paquete adecuado se denomina **system.data.sqlclient**. Para instalar este paquete damos clic derecho en la pestaña de dependencias y buscamos *administrar paquetes nuget* y en el buscador buscamos la extensión deseada y la instalamos.

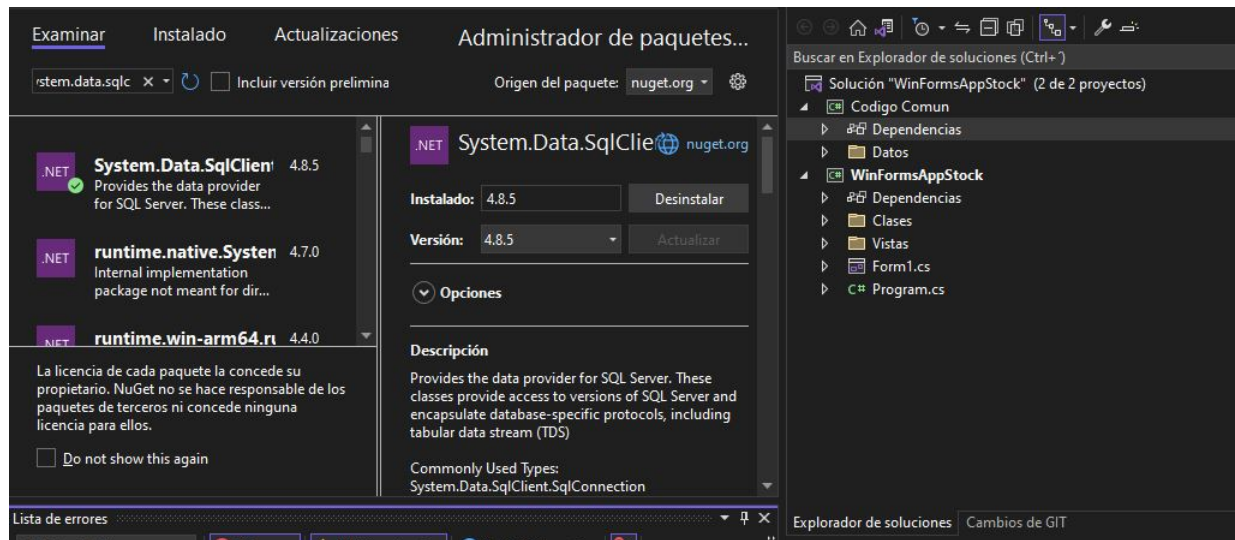


Figura 4: Nuget

Además, como es buena practica, cambiamos el namespace de la clase de AccesoDatos con el objetivo de ser concisos a la hora de declarar los usings necesarios. Debemos tener en cuenta, que debemos modificar los usings de los archivos que utilizaban este archivo, ya que lo hemos modificado y también agregar la referencia a el proyecto *Codigo Comun* en las dependencias del **WinFormsAppStock**.

Ahora, debemos realizar el mismo trabajo con la carpeta **Clases** solo que para este caso, solo hay que modificar los namespaces

En este caso la estructura nos queda de la siguiente forma

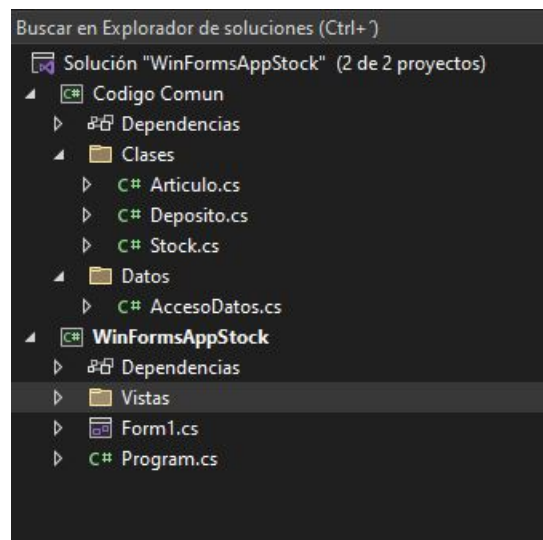


Figura 5: Estructura de la solución

Entonces, habiendo realizado todas estas modificaciones, podemos correr el programa sin problema.

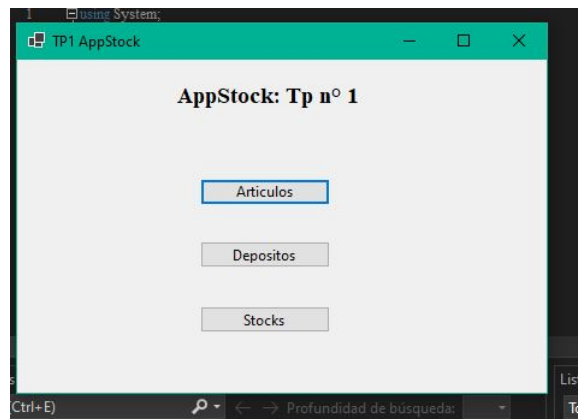


Figura 6: Corro el programa luego de modificar

Agreguemos un articulo para ver que funciona todo bien.

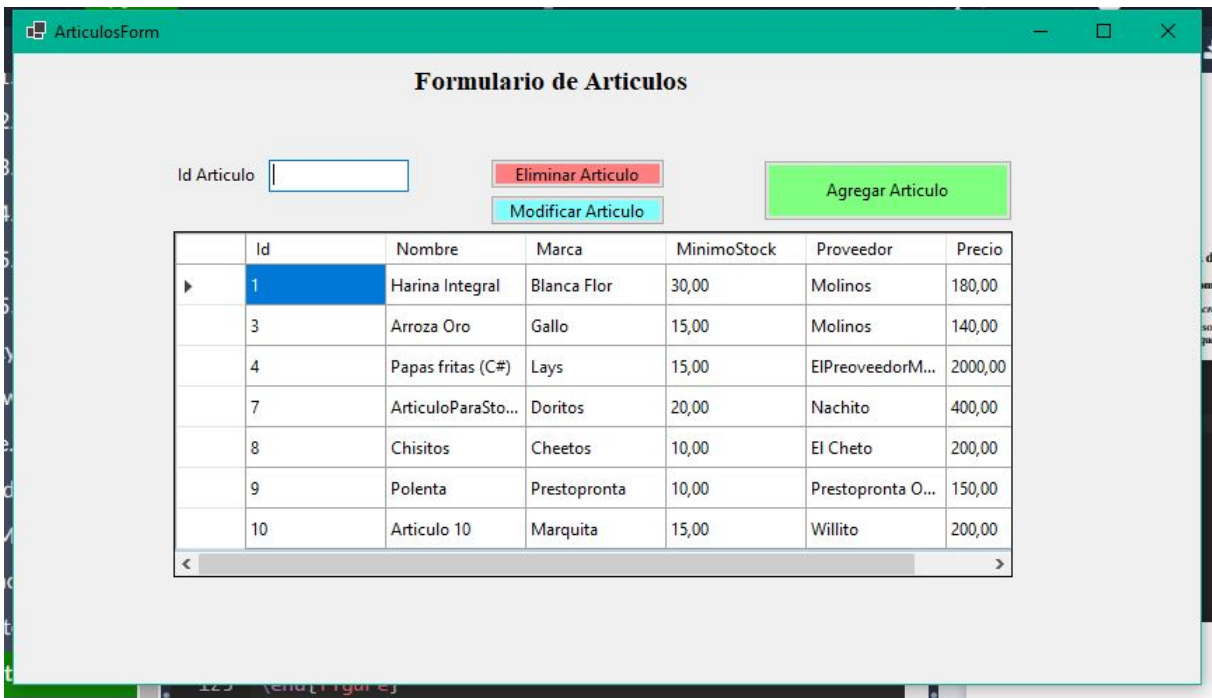


Figura 7: Momento previo a agregar un articulo

Damos clic en el botón agregar, agregamos un artículo random.

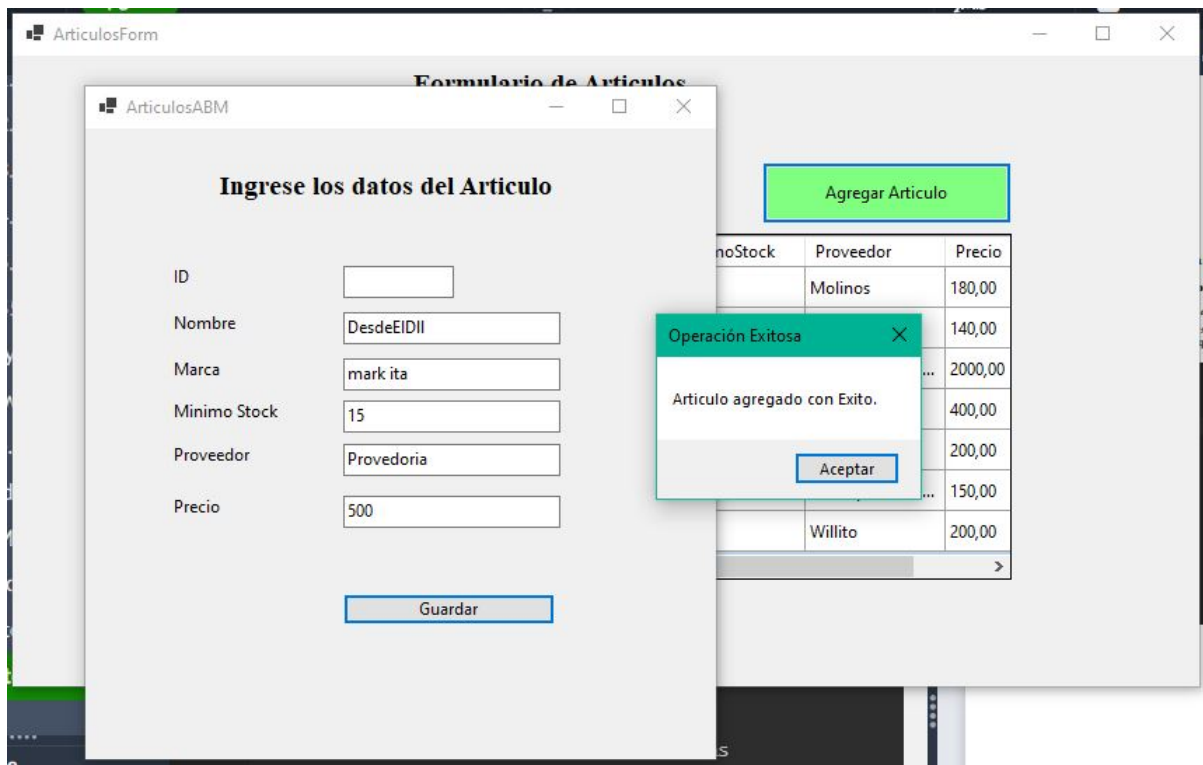


Figura 8: Agregamos el nuevo artículo

Por último, revisamos que el artículo se haya agregado. Cerramos y volvemos a abrir el formulario de artículos.

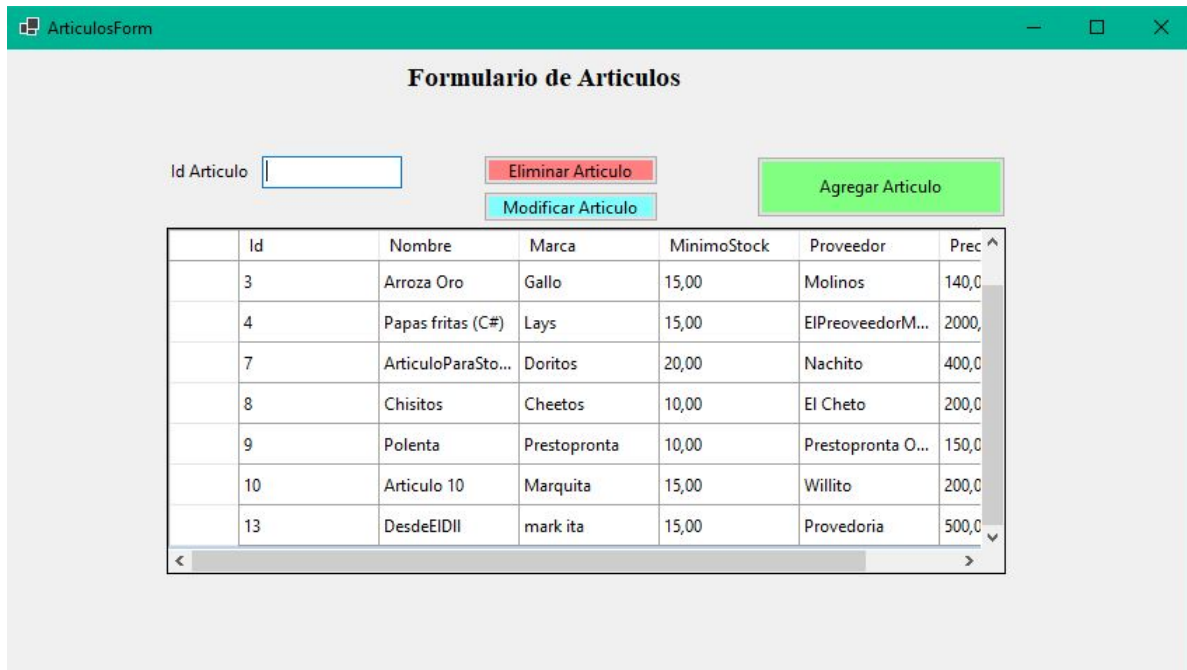


Figura 9: Verificación del Add

2. Cambiamos la Arquitectura

2.1. Cambiar nombre de la Carpeta

En **código Comun** debemos cambiar el nombre de la carpeta **Clases** por el nombre **Modelo** y luego actualizar namespaces

Agregar la carpeta **Negocio** y **Repository**.

Entonces, lo que primero hacemos es cambiar el nombre de la carpeta **Clase** y luego debemos modificar namespaces y también los usings. En la siguiente imagen podemos ver como quedarían los namespaces de las clases (en el caso de la imagen para la clase deposito), como deben ser los using y el árbol de la aplicación

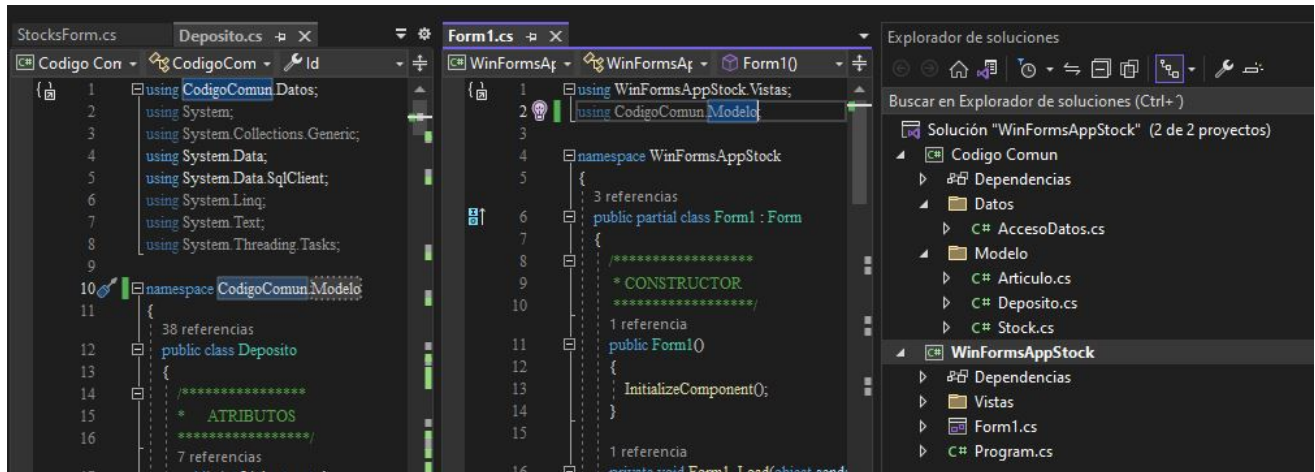


Figura 10: Cambiamos el nombre de la carpeta Clases

Además, agreguemos las Carpetas **Repository** y la Carpeta **Negocio**.

La carpeta **Negocio** se ocupara de organizar los servicios de un articulo, un deposito y el stock; mientras que la carpeta **Repository** se encargara organizar la logica y validación del tratamiento con datos. Esta ultima al estar relacionada con los datos la agregaremos en la carpeta **Datos**.

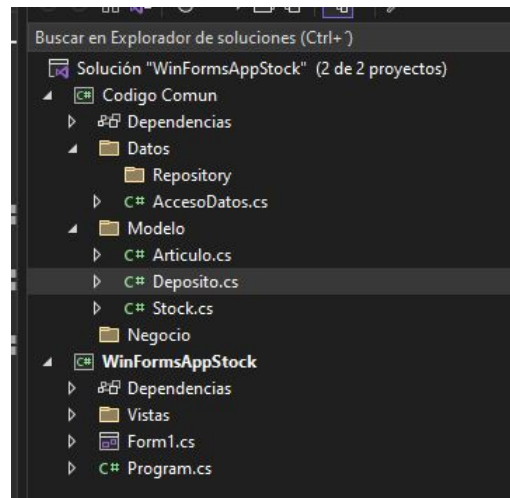


Figura 11: Agregamos las carpetas Negocio y Repository

2.2. Modelo Artículo: Capa de Datos

Para el Modelo Artículo debemos implementar en la capa de datos un Repository que contenga todos los métodos que *Conectan* con la base de datos que están en la clase artículo

Para esto, en la carpeta Repository vamos a agregar el archivo `ArticulosRepository.cs`, el cual alojara la logica necesaria para conectar a la clase Artículo con la base de datos.

Algo que debemos recordar es de *llamar al using correspondiente a las clases con las cuales estamos trabajando*. Esto es necesario justamente para poder llamar a las clases y así trabajar con estas. En este caso el using es `CodigoComun.Modelo` que es donde se encuentra ubicada la clase **Artículo**.

Agregando todas los metodos de interacción al archivo `ArticulosRepository.cs`, nos queda algo con la siguiente estructura.¹

```

3  using System.Data;
4  using System.Data.SqlClient;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8  :
9  using CodigoComun.Modelo;
10
11 namespace CodigoComun.Datos.Repository
12 {
13     0 referencias
14     public class ArticulosRepository
15     {
16         0 referencias
17         * ATRIBUTOS
18         *****
19         private AccesoDatos ac = new AccesoDatos();
20         /*****
21         * METODOS
22         *****/
23         0 referencias
24         public Artículo GetArticuloById(int idArticulo)
25
26         0 referencias
27         public List<Articulo> GetAllArticulosById()
28
29         0 referencias
30         public int AddArticuloDB(Articulo articuloToAdd)
31
32         0 referencias
33         public int DeleteArticuloOnDB(int idArticuloToDelete)
34
35         0 referencias
36         public int ModifyArticuloOnDB(Articulo articuloToModify)
37     }
38 }

```

Figura 12: Metodos de la Capa Repository de un Artículo

2.3. Capa de negocios: ArtículoServices

Esta capa es la encargada de hacer la logica de negocios. En este caso, lo utilizaremos para hacer las Altas bajas y modificaciones desde esta capa para abstraernos de la capa de Datos (El repository)

Realizamos los métodos de los articulos utilizando entonces las llamadas a base de datos. los metodos nos quedan de la siguiente forma

¹El codigo completo lo podremos encontrar en el drive cuyo link se encuentra en referencias

```

1 referencia
public string AgregarArticulo(Articulo articuloToAdd)
{
    ArticulosRepository articulosRepository = new ArticulosRepository();
    // Validaciones

    // Add
    if (articulosRepository.AddArticuloDB(articuloToAdd) == 1)
    {
        //se hizo bien
        return "Articulo Agregado con Exito";
    }
    else
    {
        //se hizo mal
        return "No se pudo agregar el Articulo";
    }
}

1 referencia
public string EliminarArticulo(int idArticuloToDelete)
{
    ArticulosRepository articulosRepository = new ArticulosRepository();
    // Validaciones

    // Delete
    if (articulosRepository.DeleteArticuloOnDB(idArticuloToDelete) == 1)
    {
        return "Articulo Eliminado con Exito";
    }
    else
    {
        //se hizo mal
        return "No se pudo eliminar el Articulo";
    }
}

1 referencia
public string ModificarArticulo(Articulo articuloToModify)
{
    ArticulosRepository articulosRepository = new ArticulosRepository();
    // Validaciones

    // Modificaciones
    if (articulosRepository.ModifyArticuloOnDB(articuloToModify) == 1)
    {
        return "El Articulo fue modificado con exito";
    }
    else
    {
        return "No se pudo modificar el Articulo.";
    }
}

```

Figura 13: Métodos del ArtículoServices

Además, los getters también deben realizarse desde los servicios, por lo que definimos los siguientes getters

```

*****
*   GETS
*****/
0 referencias
public Articulo GetPorID(int idArticulo) {
    ArticulosRepository articuloRepository = new ArticulosRepository();
    if (idArticulo > 0)
    {
        return articuloRepository.GetArticuloById(idArticulo);
    }
    else {
        return null;
    }
}

1 referencia
public List<Articulo> GetTodosPorID()
{
    ArticulosRepository articuloRepository = new ArticulosRepository();
    return articuloRepository.GetAllArticulosById();
}

```

Figura 14: Métodos del ArtículoServices

En base a estos nuevos métodos actualizamos los códigos de ArticulosForm (al igual que algunas pequeñas modificaciones en stockABM para realizar los gets de los artículos).

Primero modifiquemos el método de carga de artículos, este quedara como sigue

```
// <summary>
1 referencia
private void CargarArticulos()
{
    ArticuloServices articuloAux = new ArticuloServices();
    List<Articulo> articulosEnlaDb = articuloAux.GetTodosPorID();
    gvArticulos.DataSource = articulosEnlaDb;
}
```

Figura 15: CargarArticulos()

Ahora, habiendo modificado este metodo, modificamos el de Agregar

```
1 referencia
private void AgregarArticulo()
{
    Articulo articuloAGuardar = new Articulo();
    articuloAGuardar.Nombre = txtNombre.Text;
    articuloAGuardar.Marca = txtMarca.Text;
    articuloAGuardar.MinimoStock = Convert.ToDecimal(txtMinimoStock.Text);
    articuloAGuardar.Proveedor = txtProveedor.Text;
    articuloAGuardar.Precio = Convert.ToDecimal(txtPrecio.Text);

    // Instancio el servicio
    ArticuloServices articuloServices = new ArticuloServices();
    string mensaje = articuloServices.AgregarArticulo(articuloAGuardar);
    if (mensaje == "Articulo Agregado con Exito")
    {
        MessageBox.Show(mensaje, "Operación Exitosa");
        this.Close();
    }
    else
    {
        MessageBox.Show(mensaje, "Hubo un problema");
    }
}
```

Figura 16: AgregarArticulo()

Luego, el de modificar, el cual es similar al de agregar.

```
// <summary>
1 referencia
private void ModificarArticulo()
{
    Articulo articuloAModificar = new Articulo();
    Articulo articuloAux = new Articulo();

    articuloAModificar.Id = Convert.ToInt32(txtId.Text);
    articuloAModificar.Nombre = txtNombre.Text;
    articuloAModificar.Marca = txtMarca.Text;
    articuloAModificar.MinimoStock = Convert.ToDecimal(txtMinimoStock.Text);
    articuloAModificar.Proveedor = txtProveedor.Text;
    articuloAModificar.Precio = Convert.ToDecimal(txtPrecio.Text);

    // Instancio el servicio
    ArticuloServices articuloServices = new ArticuloServices();
    string mensaje = articuloServices.ModificarArticulo(articuloAModificar);
    if (mensaje == "El Articulo fue modificado con exito")
    {
        MessageBox.Show("El Articulo fue modificado con exito", "Operación Exitosa");
        this.Close();
    }
    else
    {
        MessageBox.Show(mensaje, "Hubo un problema");
    }
}
```

Figura 17: ModificarArticulo()

Y por ultimo vamos al ArticulosForm y modificamos el de eliminar

```
1 referencia
private void btnEliminarArticulo_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(txtIdArticulo.Text))
    {
        MessageBox.Show("Por favor, ingrese un numero de Id correspondiente a un Articulo.", "Cuidado!");
    }
    else
    {
        int idArticuloAEliminar = Convert.ToInt32(txtIdArticulo.Text);
        ArticuloServices articuloServices = new ArticuloServices();

        string mensaje = articuloServices.EliminarArticulo(idArticuloAEliminar);
        if (mensaje == "Articulo Eliminado con Exito")
        {
            MessageBox.Show(mensaje, "Operación Exitosa");
        }
        else
        {
            MessageBox.Show(mensaje, "Hubo un Problema");
        }
    }
}
```

Figura 18: EliminarArticulo()

Habiendo realizado estas modificaciones y las necesarias según las llamadas de los Getters de articulo en StockABM, compilamos y probamos que funcione todo bien

Empecemos agregando un articulo.

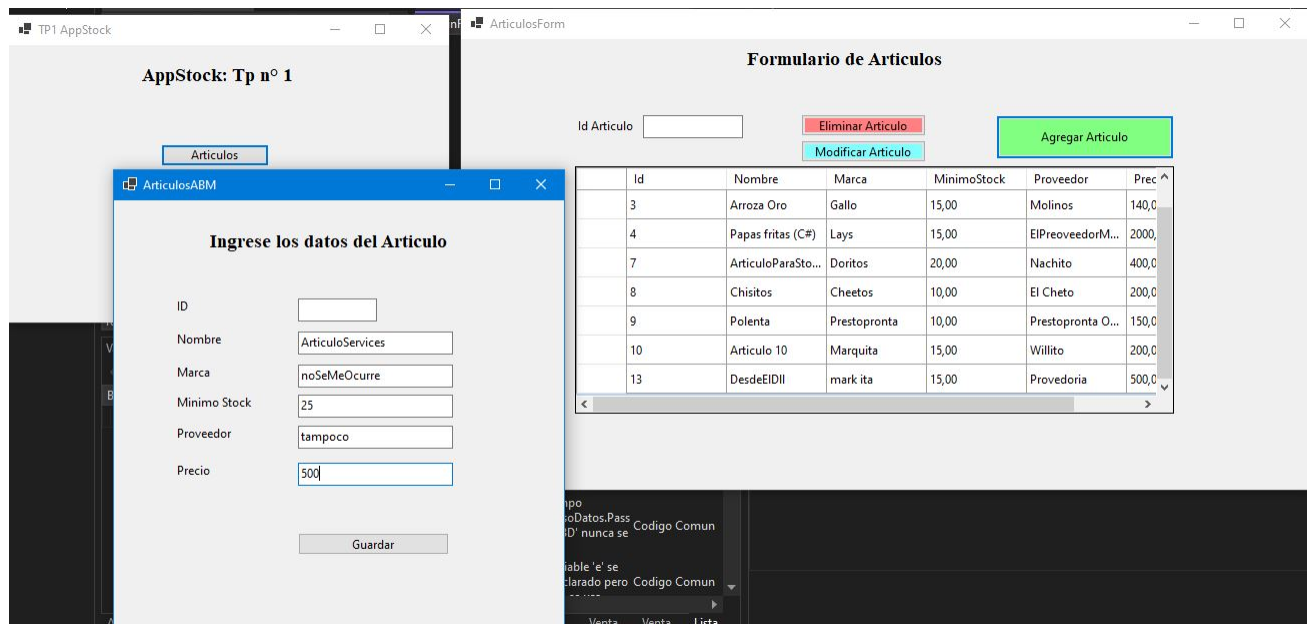


Figura 19: Agregamos un articulo random

Y vemos si se agrego



Figura 20: Se agrega con exito

Luego, modifiquemos el artículo agregado anteriormente.

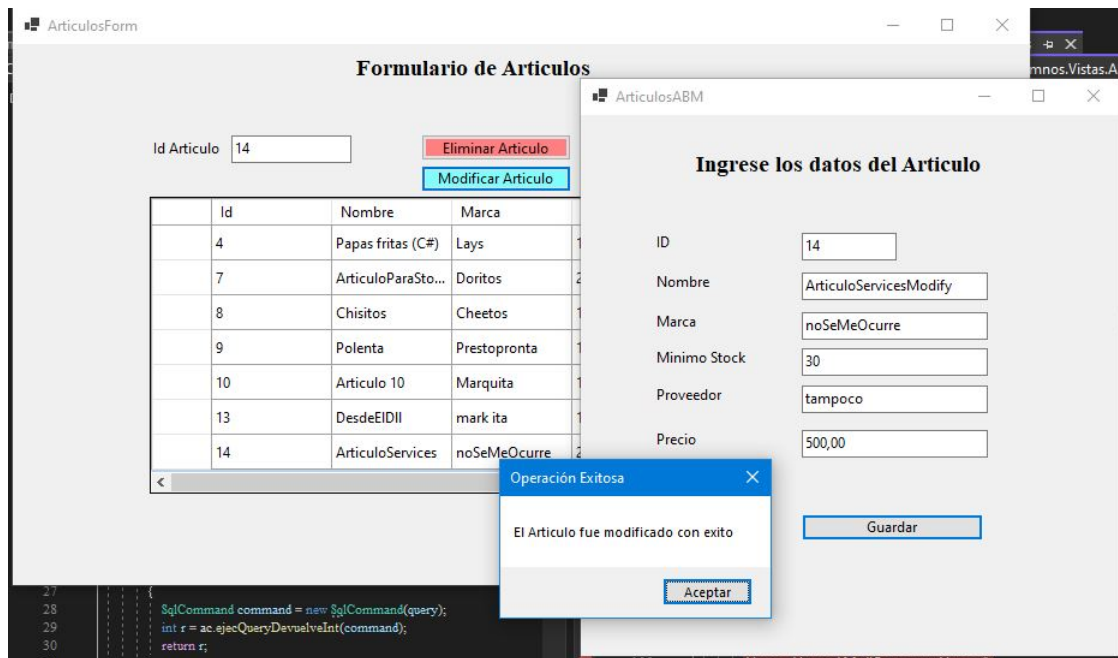


Figura 21: Modificamos el artículo

Y revisamos que se haya modificado con éxito

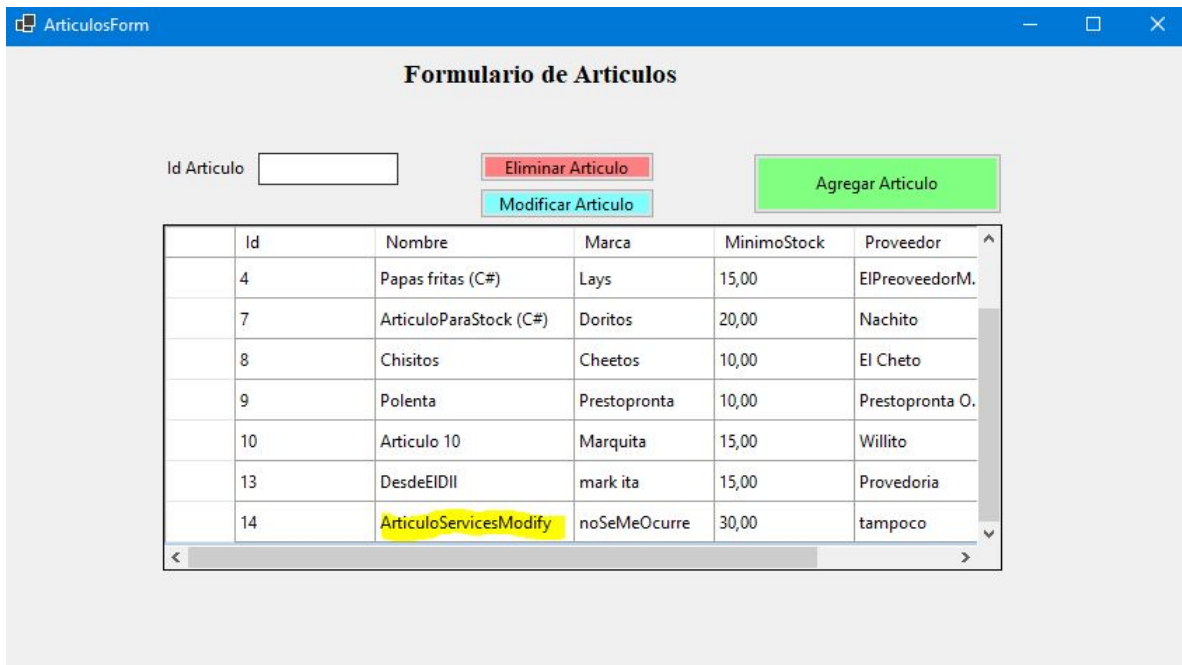


Figura 22: Verificado el cambio

Por ultimo, eliminamos el articulo que agregamos para probar el método de eliminar.

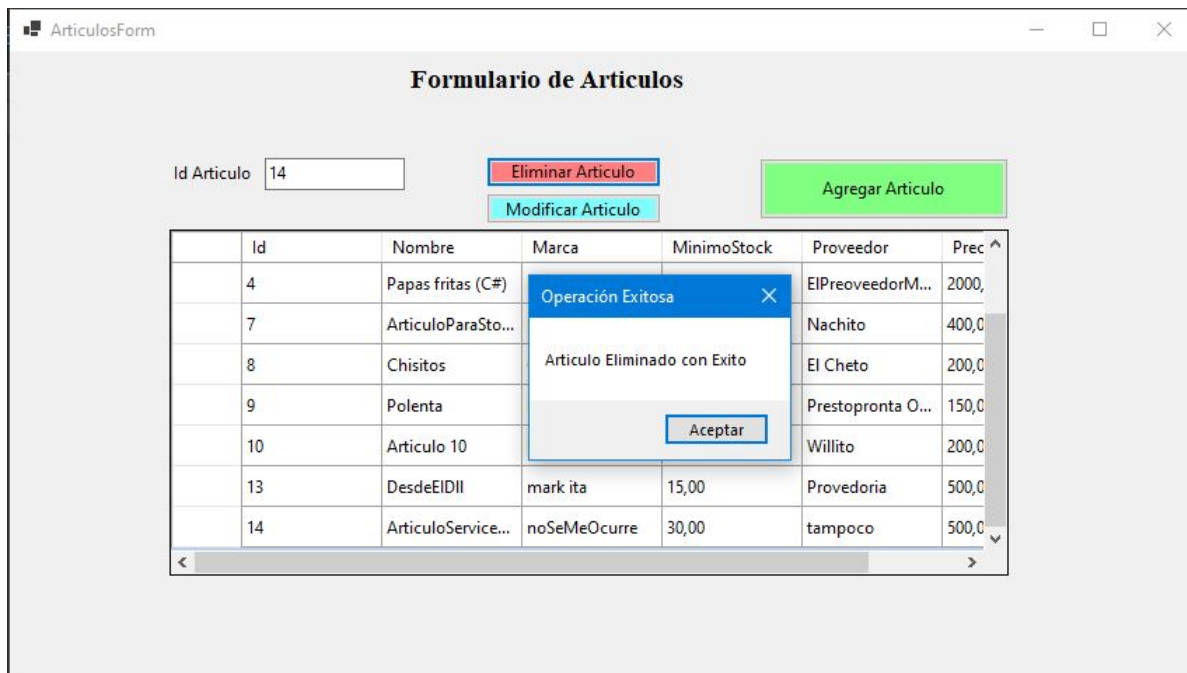


Figura 23: Borrarnos el articulo

Y verificamos que se haya eliminado

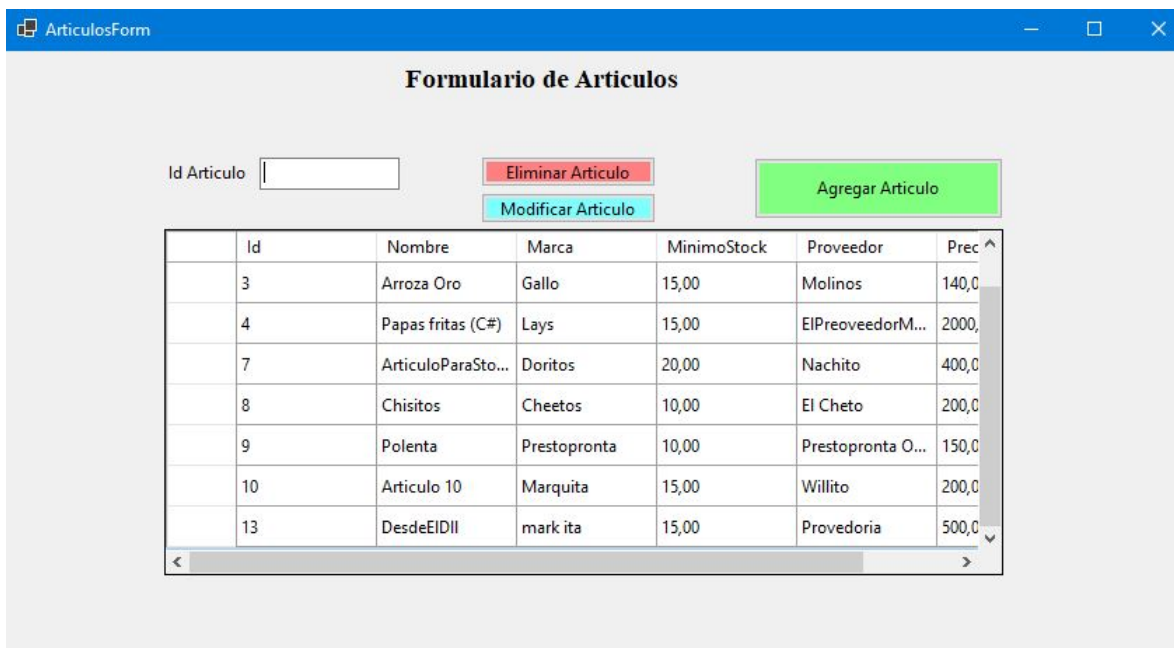
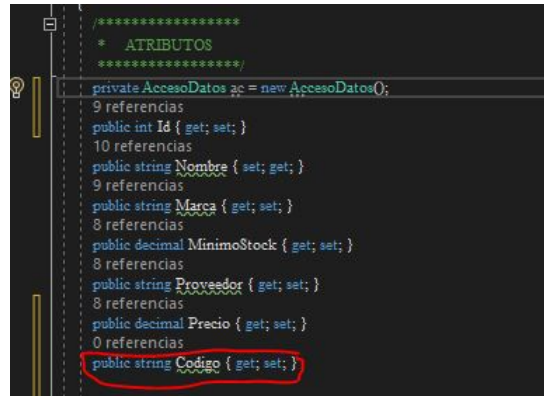


Figura 24: Verificación

3. Cambio en el sistema

3.1. Agrego Atributo en Artículo

Primero, agregamos un atributo nuevo en la clase Artículo, el cual se llamara *Codigo* y sera del tipo *string*



```

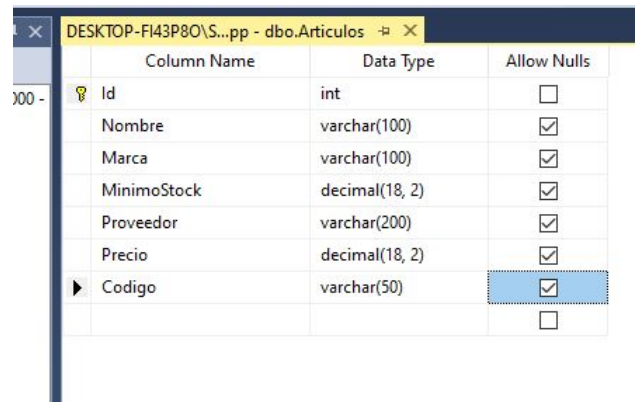
/*****
 * ATRIBUTOS
 *****/
private AccesoDatos ac = new AccesoDatos();
9 referencias
public int Id { get; set; }
10 referencias
public string Nombre { get; set; }
9 referencias
public string Marca { get; set; }
8 referencias
public decimal MinimoStock { get; set; }
8 referencias
public string Proveedor { get; set; }
8 referencias
public decimal Precio { get; set; }
0 referencias
public string Codigo { get; set; }

```

Figura 25: Agregamos el nuevo atributo

3.2. Modifico SQL Server

Agrego en la tabla de datos del Artículo el nuevo atributo con un *varchar(50)*



Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Nombre	varchar(100)	<input checked="" type="checkbox"/>
Marca	varchar(100)	<input checked="" type="checkbox"/>
MinimoStock	decimal(18, 2)	<input checked="" type="checkbox"/>
Proveedor	varchar(200)	<input checked="" type="checkbox"/>
Precio	decimal(18, 2)	<input checked="" type="checkbox"/>
Codigo	varchar(50)	<input checked="" type="checkbox"/>

Figura 26: Tabla modificada

3.3. Modificamos ArticulosRepository

Ahora, debemos modificar todas las queries, pues modificamos la base de datos.

Haciendo los cambios necesarios a cada clase nos quedan como sigue:

Primero los Getters


```

4 referencias
public Artículo GetArticuloById(int idArticulo)
{
    try
    {
        string select = $"select * from Articulos where id={idArticulo}";
        SqlCommand command = new SqlCommand(select);
        DataTable dt = ac.execDT(command);

        if (dt.Rows.Count <= 0)
        {
            //no se encuentra pedido para actualizar estado
            return null;
        }

        Artículo articuloADevovlerConDatosDelaBaseDeDatos = new Artículo();
        foreach (DataRow dr in dt.Rows)
        {
            articuloADevovlerConDatosDelaBaseDeDatos.Id = Convert.ToInt32(dr["Id"]);
            articuloADevovlerConDatosDelaBaseDeDatos.Nombre = dr["Nombre"].ToString();
            articuloADevovlerConDatosDelaBaseDeDatos.Marca = dr["Marca"].ToString();
            articuloADevovlerConDatosDelaBaseDeDatos.MinimoStock = Convert.ToDecimal(dr["MinimoStock"]);
            articuloADevovlerConDatosDelaBaseDeDatos.Proveedor = dr["Proveedor"].ToString();
            articuloADevovlerConDatosDelaBaseDeDatos.Precio = Convert.ToDecimal(dr["Precio"]);
            articuloADevovlerConDatosDelaBaseDeDatos.Codigo = dr["Codigo"].ToString();
        }

        return articuloADevovlerConDatosDelaBaseDeDatos;
    }
    catch (Exception ex)
    {
        return null;
    }
}

```

Figura 27: GetArticuloById()

```

1 referencia
public List<Artículo> GetAllArticulosById()
{
    try
    {
        string select = "select * from Articulos ";
        SqlCommand command = new SqlCommand(select);
        DataTable dt = ac.execDT(command);

        if (dt.Rows.Count <= 0)
        {
            //no se encuentra pedido para actualizar estado
            return null;
        }

        List<Artículo> articulosADevovlerConDatosDelaBaseDeDatos = new List<Artículo>();
        foreach (DataRow dr in dt.Rows)
        {
            Artículo articuloAux = new Artículo();
            articuloAux.Id = Convert.ToInt32(dr["Id"]);
            articuloAux.Nombre = dr["Nombre"].ToString();
            articuloAux.Marca = dr["Marca"].ToString();
            articuloAux.MinimoStock = Convert.ToDecimal(dr["MinimoStock"]);
            articuloAux.Proveedor = dr["Proveedor"].ToString();
            articuloAux.Precio = Convert.ToDecimal(dr["Precio"]);
            articuloAux.Codigo = dr["Codigo"].ToString();
            //agrego a la lista
            articulosADevovlerConDatosDelaBaseDeDatos.Add(articuloAux);
        }

        return articulosADevovlerConDatosDelaBaseDeDatos;
    }
    catch (Exception ex)
    {
        return null;
    }
    finally
    {
    }
}

```

Figura 28: GetAllArticulosById()

Luego, seguimos por los Add, Delete y Modify

```
1 referencia
public int AddArticuloDB(Articulo articuloToAdd)
{
    string query = $"insert into Articulos (Nombre, Marca, MinimoStock, Proveedor, Precio,Codigo)";
    query += $" values({articuloToAdd.Nombre}, " +
        $" {articuloToAdd.Marca}, " +
        $" {articuloToAdd.MinimoStock}, " +
        $" {articuloToAdd.Proveedor}, " +
        $" {articuloToAdd.Precio}) " +
        $" {articuloToAdd.Codigo}";
    try
    {
        SqlCommand command = new SqlCommand(query);
        int r = ac.ejecQueryDevuelveInt(command);
        return r;
    }
    catch (Exception ex)
    {
        //Log.Escribir(0, "ERROR", "Ocurrio un error en ChangeAddPartidaToProduct actualizando que guar
        return -1;
    }
    finally
    {
        ac.DesConectar();
    }
}
```

Figura 29: AddArticuloOnDb()

```
1 referencia
public int ModifyArticuloOnDB(Articulo articuloToModify)
{
    string MinStockAux = articuloToModify.MinimoStock.ToString();
    string PrecioAux = articuloToModify.Precio.ToString();

    MinStockAux = MinStockAux.Replace(",",".");
    PrecioAux = PrecioAux.Replace(",",".");

    string query = $"update articulos set Nombre={articuloToModify.Nombre}, " +
        $"Marca={articuloToModify.Marca}, " +
        $"MinimoStock={MinStockAux}, " +
        $"Proveedor={articuloToModify.Proveedor}, " +
        $"Precio={PrecioAux} " +
        $"Codigo={articuloToModify.Codigo}, " +
        $"where id = {articuloToModify.Id}";
    try
    {
        SqlCommand command = new SqlCommand(query);
        int r = ac.ejecQueryDevuelveInt(command);
        return r;
    }
    catch (Exception ex)
    {
        return -1;
    }
    finally
    {
        ac.DesConectar();
    }
}
```

Figura 30: ModifyArticulosById()

```
1 referencia
public int DeleteArticuloOnDB(int idArticuloToDelete)
{
    string query = $"delete articulos where id={idArticuloToDelete}";
    try
    {
        SqlCommand command = new SqlCommand(query);
        int r = ac.ejecQueryDevuelveInt(command);
        return r;
    }
    catch (Exception ex)
    {
        return -1;
    }
    finally
    {
        ac.DesConectar();
    }
}
```

Figura 31: DeleteArticulosById()

3.4. Validar Código

Ahora realizaremos una modificación para que no existan dos códigos iguales. Para esto, vamos al `ArticuloServices` y modificamos el método `AgregarArticulo()` como sigue

```

public string AgregarArticulo(Articulo articuloToAdd)
{
    ArticulosRepository articulosRepository = new ArticulosRepository();
    ArticuloServices articuloServices = new ArticuloServices();
    List<Articulo> articulos = articulosRepository.GetAllArticulosById();
    // Validaciones
    foreach (Articulo art in articulos) {
        if (articuloToAdd.Codigo == art.Codigo) {
            return "El Código del artículo ya existe. Elija otro código";
        }
    }
    // Add
    if (articulosRepository.AddArticuloDB(articuloToAdd) == 1)
    {
        //se hizo bien
        return "Artículo Agregado con Exito";
    }
    else
    {
        //se hizo mal
        return "No se pudo agregar el Artículo";
    }
}

```

Figura 32: `ArticuloServices/AgregarArticulo()`

3.5. Trabajamos en Winforms Articulos

Debemos modificar el `ArticulosABM`, tanto en diseño para poder agregar el título como el código, en método `AgregarArticulo()` y también en el `CargarDatosArticuloParaModificar()` y en `ModificarArticulo()`

Habiendo realizado esas modificaciones, debería funcionar. Vamos a compilarlo y a agregar un artículo².

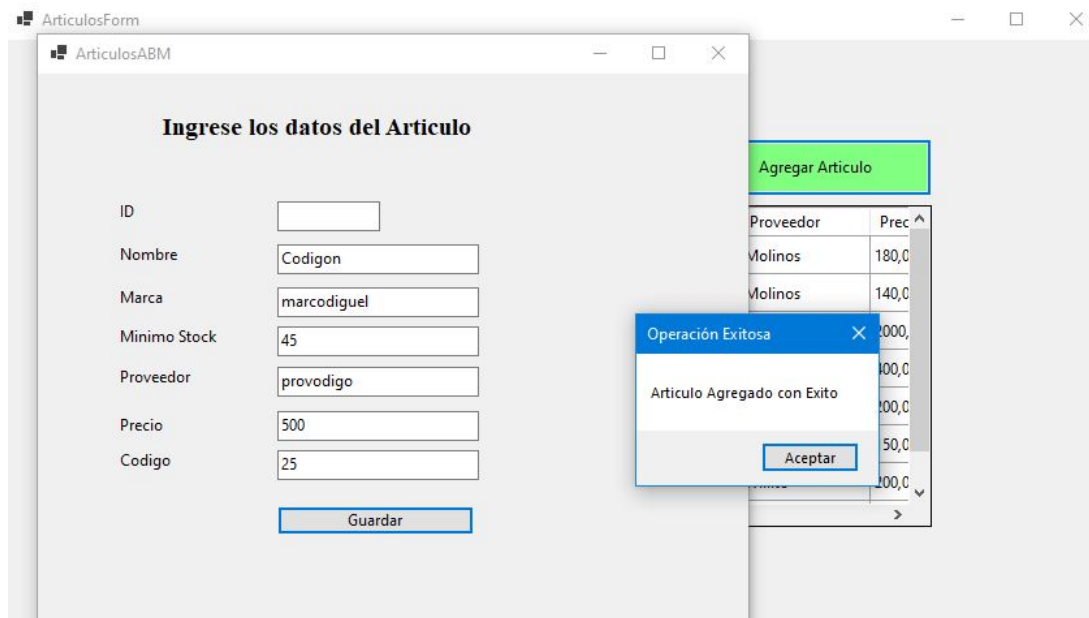


Figura 33: Agrego un artículo con *codigo* = 25

²Previamente utilizando *queries* en la base de datos le agregue códigos diferentes a los otros artículos. Se podría haber hecho de otra forma con un pequeño *script* pero decidí hacerlo a mano

Ahora, probemos agregar uno que tenga el mismo código para ver si esta bien hecha la validación

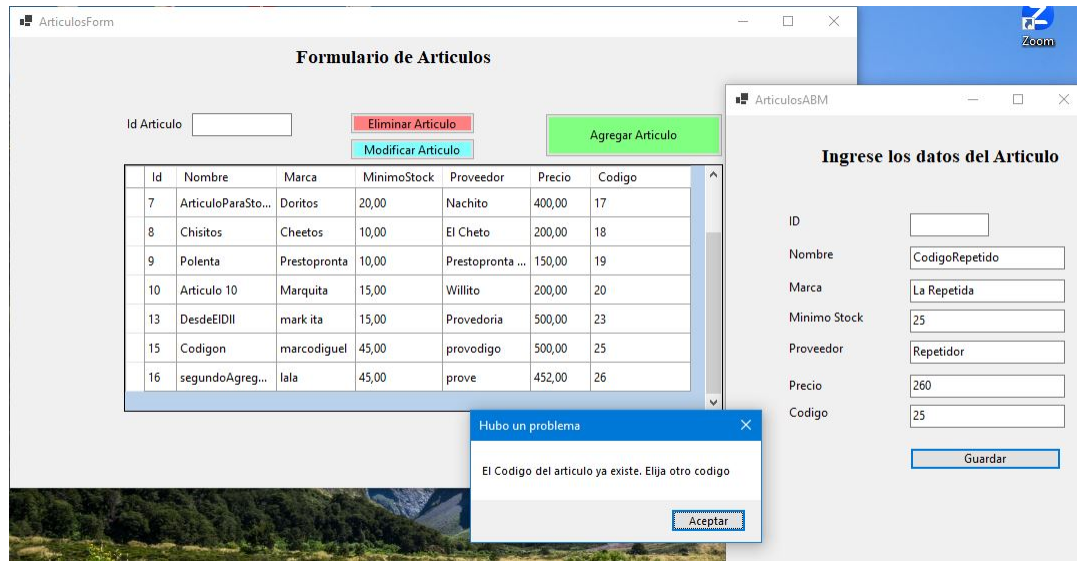


Figura 34: Funcionamiento de la validación

4. Referencias

1. Link al Drive con el Código