


TAREA: CLASE 9

CAPACITACIÓN C#

 **Ezequiel Remus**
ezequielremus@gmail.com

11 de abril de 2023

RESUMEN

En este apunte se encuentran alojadas las resoluciones a cada ejercicio de la Tarea de la clase 9 de la Capacitación en C#. En este caso, nos encontraremos trabajando con *Entity Framework*. Tomaremos como ejemplo la AppStock y haremos una suerte de *refactory* de la clase Deposito.

1. Ingeniería Inversa

1.1. Enunciado

Usando **Entity Framework** hacer *Ingeniería inversa*, creando una carpeta Entities y que allí se guarde la Clase **Deposito** y la Clase **Base De Datos** generada por entity framework.

Enviar captura de pantalla del árbol de visual studio donde se vean estas clases

1.2. Solución

Lo primero que debemos hacer es, en CodigoComun establecer mediante ingeniería Inversa de Entity Framework una clase con su conexión a la base de datos.

Para esto damos clic derecho en el proyecto, y en la pestaña del **EF Core Power Tools** seleccionamos la opción *Reverse Engineer*. Nos abre una ventana en la cual debemos elegir con qué base nos estamos conectando. En este caso, vamos a conectarnos a una Base de Datos de SQL server instalada Localmente, dicha configuración puede verse en la Figura 2.

Luego, nos aparecerá un cartel en el cual nos dice que elijamos la conexión al cual debemos clicar en **OK** (ver Figura 1).

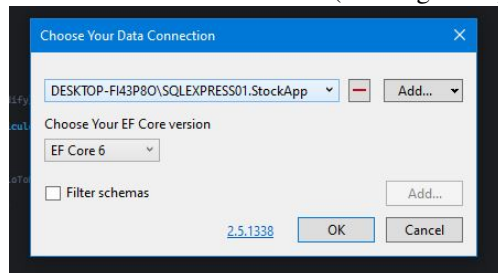


Figura 1: Establecemos a conexión

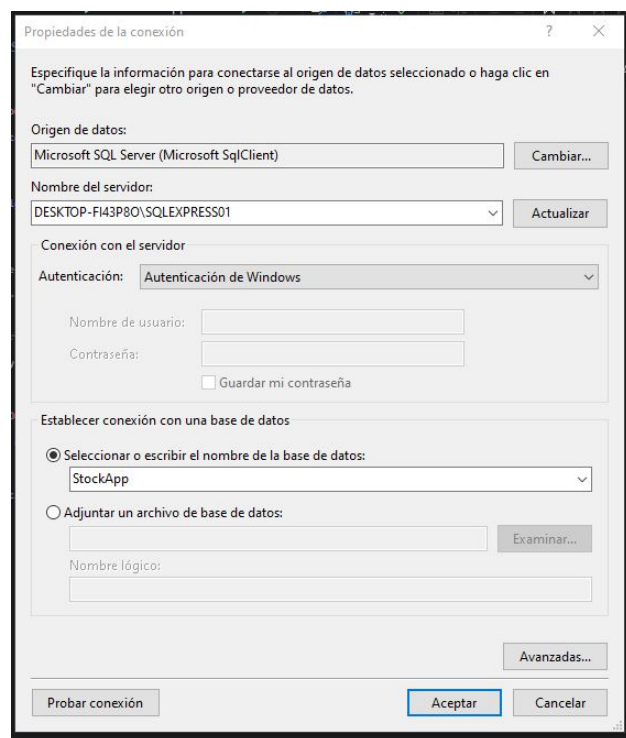


Figura 2: Establecemos a conexión

Una vez establecido, nos pedira que elijamos cual tabla vamos a utilizar, en nuestro caso es la de **Depositos** Luego de

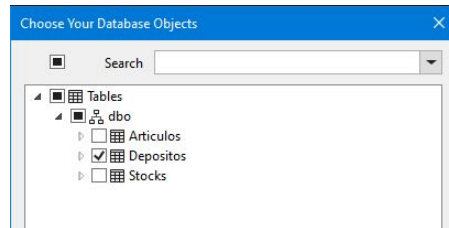


Figura 3: Caption

elegir la Tabla, nos va a pedir que le indiquemos como queremos configurar el modelo y la conexión. En este caso lo establecemos como se ve en la Figura 4. Luego de aceptar nos quedara en nuestro proyecto el árbol de la Figura 5

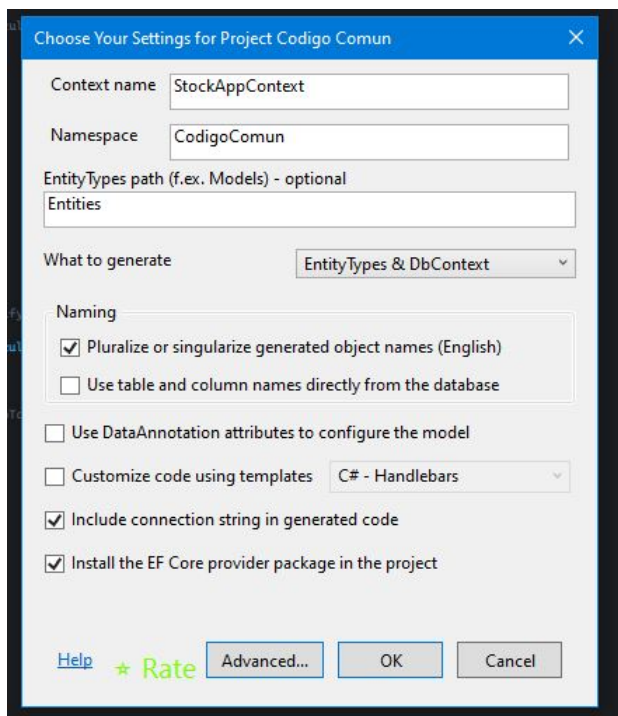


Figura 4: Configuración en Proyecto

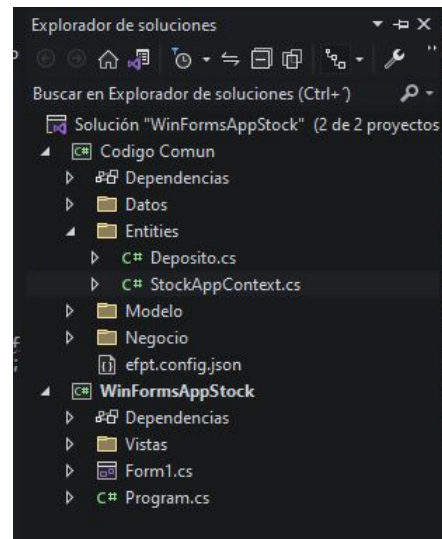


Figura 5: Árbol establecido

2. Capa de Datos: DepositoRepository

2.1. Enunciado

Crear en la Capa de Datos un “DepositoRepository” y el mismo debe implementar los siguientes métodos utilizando las clases de entity framework generadas en el punto 1

- 2.2.1 GetTodosLosDepositos()
- 2.2.2 GetDepositoPorId()
- 2.2.3 AddDeposito()
- 2.2.4 DeleteDeposito()

Enviar captura de pantalla del repository donde se vean estos métodos implementados

2.2. Solución

Ahora, en la **Capa de Datos** creamos la clase `DepositosRepository.cs` la cual se encargara de hacer la logica de conexión a base de datos según Entity Framework. Nos queda debe quedar el árbol de la Figura 6.

Luego, pasamos a escribir los métodos requeridos, sin más dejamos sus codigos en las Figuras 7, 8, 9 y 10

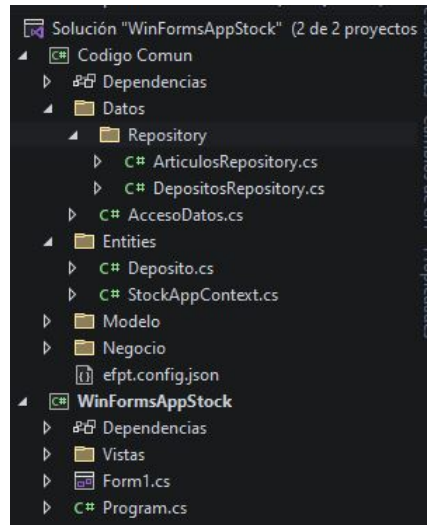


Figura 6: Nuevo árbol

2.2.1. Solución: GetTodosLosDepositos()

```

/// <summary>
/// Toma todos los depositos ubicados en la base de datos
/// </summary>
/// <returns></returns>
0 referencias
public List<Deposito> GetTodosLosDepositos()
{
    List<Deposito> depositosADevolver = new List<Deposito>();
    depositosADevolver = this.db.Depositos.ToList();

    return depositosADevolver;
}

```

Figura 7: GetTodosLosDepositos()

2.2.2. Solución: GetDepositoPorId()

```

/// <summary>
/// Toma el deposito pasado por id
/// </summary>
/// <param name="idDeposito"></param>
/// <returns></returns>
0 referencias
public Deposito GetDepositoPorId(int idDeposito)
{
    #pragma warning disable CS8600 // Se va a convertir un literal nulo o un posible valor nulo en un
    Deposito depositoADevolver = this.db.Depositos.Where(
        p => p.Id == idDeposito).FirstOrDefault();

    #pragma warning disable CS8603 // Posible tipo de valor devuelto de referencia nulo
    return depositoADevolver;
}

```

Figura 8: GetDepositoPorId()

2.2.3. Solución: AddDeposito()

```
/// <summary>
///     Utiliza Entity Framework para
///     guardar el deposito en la base de datos
/// </summary>
/// <param name="depositoToAdd"></param>
/// <returns></returns>
0 referencias
public int AddDeposito(Deposito depositoToAdd)
{
    this.db.Depositos.Add(depositoToAdd);
    int r = db.SaveChanges();
    return r;
}
```

Figura 9: AddDeposito()

2.2.4. Solución: DeleteDeposito()

```
/// <summary>
///     Utiliza Entity Framework para
///     eliminar el deposito de la base de datos
/// </summary>
/// <param name="idDepositoToDelete"></param>
/// <returns></returns>
0 referencias
public int DeleteDeposito(int idDepositoToDelete)
{
    // Obtengo al deposito
    Deposito depositoToDelete = this.GetDepositoPorId(idDepositoToDelete);
    // lo borro de la DB
    this.db.Depositos.Remove(depositoToDelete);
    // Guardo los Cambios
    int r = this.db.SaveChanges();
    return r;
}
```

Figura 10: DeleteDeposito()

3. Capa de Negocio: DepositoServices

3.1. Enunciado

Crear un **DepositoServices** que utilizando el repository del punto anterior implemente sus funcionalidades. Igual al **ArticuloServices** sin la función de actualizar que aun no lo vimos

Enviar Captura de esta Clases donde se vean los métodos

3.2. Solución

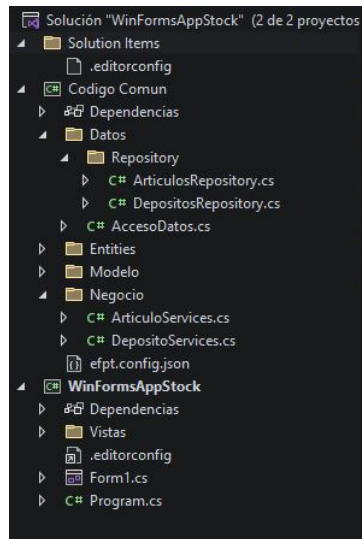


Figura 11: Nuevo árbol

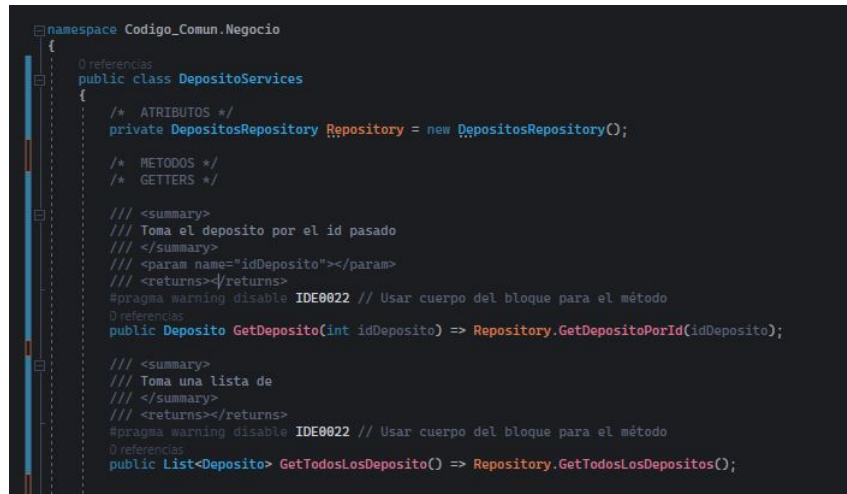


Figura 12: Atributos y Getters



Figura 13: A.B.M

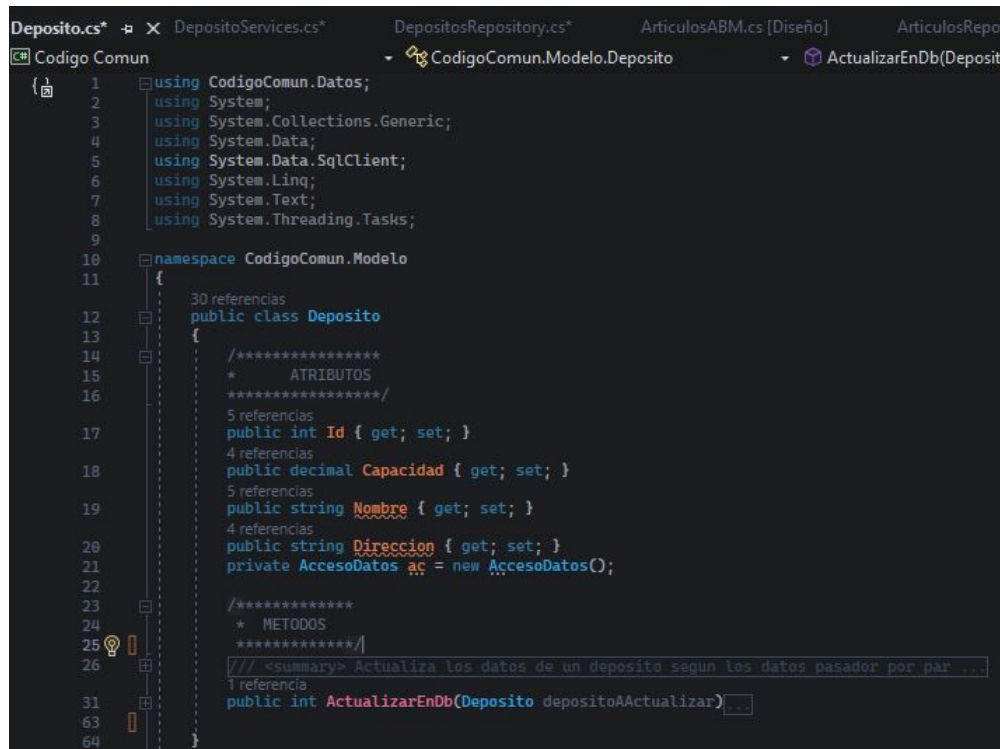
4. Formateo

4.1. Enunciado

Eliminar de la Clase **Deposito** de la carpeta **Modelo**, los métodos que conectan a la base de datos. Solo dejar el de Actualizar.

4.2. Solución

Este punto no es más que borrar los métodos, exepctuando el de **ActualizarEnDb()** y esperar que todo salga bien.



```

Deposito.cs*  DepositoServices.cs*  DepositosRepository.cs*  ArticulosABM.cs [Diseño]  ArticulosRepos
Codigo Comun
{
1  using CodigoComun.Datos;
2  using System;
3  using System.Collections.Generic;
4  using System.Data;
5  using System.Data.SqlClient;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace CodigoComun.Modelo
11 {
12     30 referencias
13     public class Deposito
14     {
15         /*****
16         *   ATRIBUTOS
17         *****/
18         5 referencias
19         public int Id { get; set; }
20         4 referencias
21         public decimal Capacidad { get; set; }
22         5 referencias
23         public string Nombre { get; set; }
24         4 referencias
25         public string Direccion { get; set; }
26         private AccesoDatos ac = new AccesoDatos();
27
28         /*****
29         *   METODOS
30         *****/
31         /// <summary> Actualiza los datos de un deposito segun los datos pasador por par ...
32         1 referencia
33         public int ActualizarEnDb(Deposito depositoActualizar) ...
34
35     }
36 }

```

Figura 14: A.B.M

5. Modificar en la capa de presentación (Parte 1)

5.1. Enunciado

Modificar en la capa de presentación (Proyecto winform) que al buscar todos los depósitos se haga utilizando el services del punto anterior.

5.2. Solución

Acá tenemos que agregar el using de **CodigoComun.Negocios** para poder utilizar el DepositoServices. Además, como esta etapa trabaja con la clase Deposito que esta en Entities, debemos tambien llamar a **CodigoComun.Entities**.

Nos surgira el problema de que debemos usar la clase Deposito del Models (para que siga funcionando el ActualizarEnDb()) y la clase Deposito del Entities, la solución más rapida es que cuando debamos usar una u otra definirla con su ruta completa, o sea o bien usar CodigoComun.Entities.Deposito o CodigoComun.Modelo.Deposito.

Habiendo hecho esta aclaración, debemos recordar que debemos modificar tambien la utilización de los Getters en la clase Stock, mas allá de esto para implementar la nueva carga de datos utilizando el DepositoServices debemos realizar los cambios que se ven en la figura 15


```

// <summary>
// Toma la una lista de depositos desde la base de datos
// los agrega al GridView gvDepositos
// </summary>
1 referencia
private void CargarDepositos()
{
    DepositoServices depositos = new DepositoServices();
    gvDepositos.DataSource = depositos.GetTodosLosDepositos();

    //Deposito depositoAux = new Deposito();
    //List<Deposito> depositoEnlaDb = depositoAux.GetTodosLosDepositos();
    //gvDepositos.DataSource = depositoEnlaDb;
}

```

Figura 15: Carga de datos en el DepositosForm

Además, la carga de datos del id pasado como parametro a la hora de modificar tambien hay que modificarla. El codigo queda como se muestra en la figura 16

```

// <summary>
// Carga los datos del deposito pedido en los textBox
// del formulario
// </summary>
// <param name="idDepositoAModificar"></param>
1 referencia
public void CargarDatosDepositoParaModificar(int idDepositoAModificar)
{
    // Instancio un DepositoServices
    DepositoServices depositoServices = new DepositoServices();
    // Instancio un Deposito
    CodigoComun.Entities.Deposito depositoEnDb = new CodigoComun.Entities.Deposito();

    // Guardo el deposito
    depositoEnDb = depositoServices.GetDeposito(idDepositoAModificar);

    txtId.Text = idDepositoAModificar.ToString();
    txtCapacidad.Text = depositoEnDb.Capacidad.ToString();
    txtNombre.Text = depositoEnDb.Nombre;
    txtDireccion.Text = depositoEnDb.Direccion;
}

```

Figura 16: Carga de datos en el DepositosABM a la hora de Modificar

En el siguiente punto se podrá ver que funciona

6. Modificar en la capa de presentación (Parte 2)

6.1. Enunciado

Modificar en la capa de presentación (Proyecto winform) el alta de un deposito y para eliminar un deposito.

Enviar para el Alta un print de pantalla del código utilizando el services y un print donde se vea que el alta de deposito sigue funcionando con esta nueva funcionalidad

6.2. Solución

En el caso del alta del deposito o para eliminar, debemos recordar que la función que utilizamos en DepositoServices nos devuelve un mensaje, por lo que simplemente la modificación es instanciar un DespoitoServices para poder utilizar los métodos requeridos y cambiar las condiciones mediante el uso de los mensajes que nos devuelven las funciones. Los codigos quedaran como sigue:

```

/// <summary>
///     Agrega los datos del formulario
///     a la base de datos
/// </summary>
1 referencia
private void AgregarDeposito()
{
    DepositoServices depositoServices = new DepositoServices();
    CodigoComun.Entities.Deposito depositoAGuardar = new CodigoComun.Entities.Deposito();

    depositoAGuardar.Capacidad = Convert.ToDecimal(txtCapacidad.Text);
    depositoAGuardar.Nombre = txtNombre.Text;
    depositoAGuardar.Direccion = txtDireccion.Text;

    string mensaje = depositoServices.AddDeposito(depositoAGuardar);
    if (mensaje == "Deposito Agregado con exito")
    {
        MessageBox.Show(mensaje, "Operación Exitosa");
        this.Close();
    }
    else
    {
        MessageBox.Show(mensaje, "Hubo un problema");
    }
}

```

Figura 17: AgregarDeposito()

```

/// <summary>
///     Elimina un deposito de la base de datos
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 referencia
private void btnEliminarDeposito_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(txtIdDeposito.Text))
    {
        MessageBox.Show("Por favor, ingrese un numero de Id correspondiente a un Deposito.", "Cuidado!");
    }
    else
    {
        int idDepositoAEliminar = Convert.ToInt32(txtIdDeposito.Text);
        // Instancio un DepositoServices
        DepositoServices depositoAux = new DepositoServices();

        string mensaje = depositoAux.DeleteDeposito(idDepositoAEliminar);
        if (mensaje == "Deposito Eliminado con exito")
        {
            MessageBox.Show(mensaje, "Operación Exitosa!");
        }
        else
        {
            MessageBox.Show(mensaje, "Hubo un problema");
        }
    }
}

```

Figura 18: EliminarDeposito()

Una vez terminado de hacer todas las modificaciones solo queda probar como funciona, para esto vamos a Agregar un Deposito

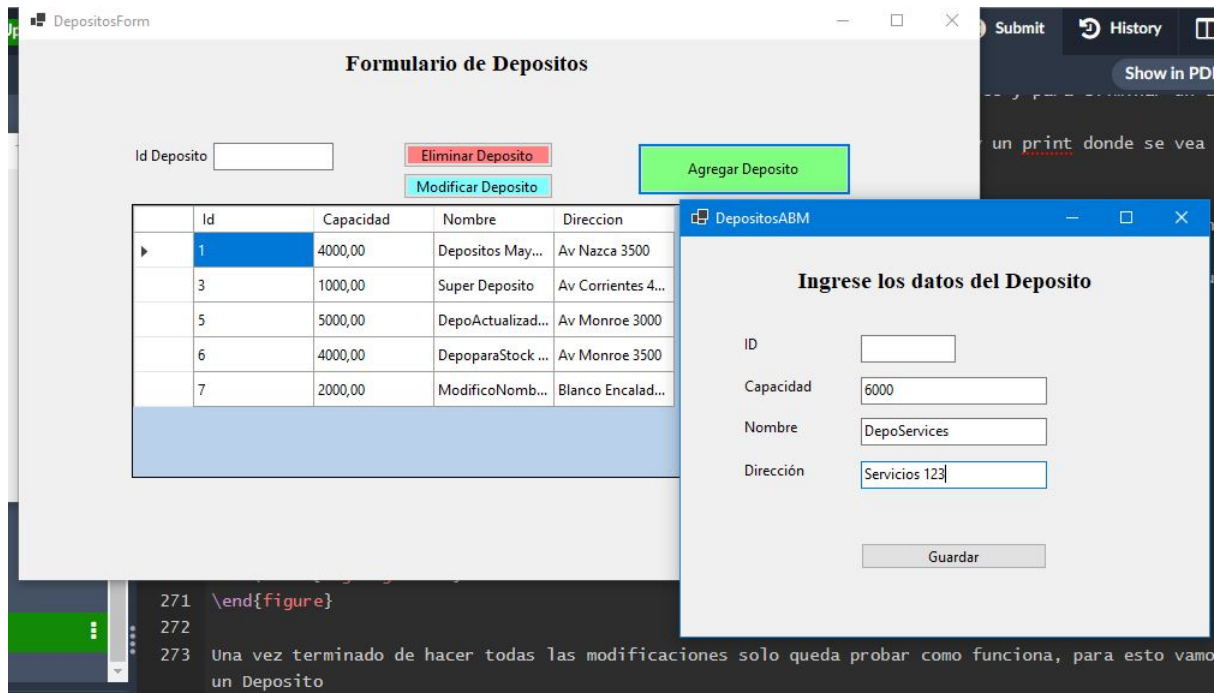


Figura 19: Agregamos el deposito

Veamos si realmente funciona. Además, podemos apreciar que funcionan los getters ya que se muestran los depósitos

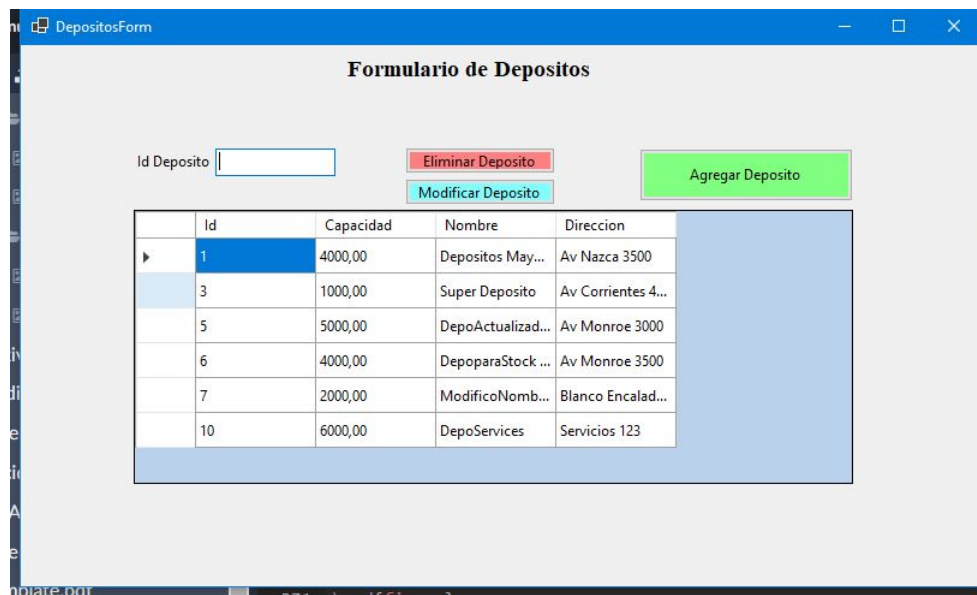


Figura 20: Se aprecia que se agrego correctamente

Ahora, probemos la funcion de Delete, eliminemos el Deposito recién creado

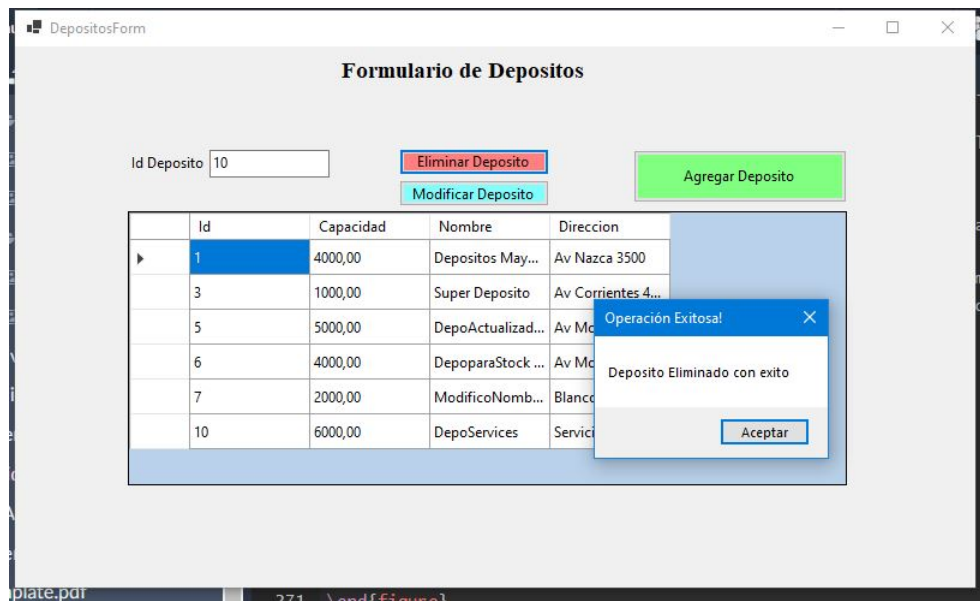


Figura 21: Se aprecia que se agrego correctamente

Y verifiquemos que se elimino



Figura 22: Se aprecia que se elimino correctamente

7. Referencias

1. Link al Drive con el Código