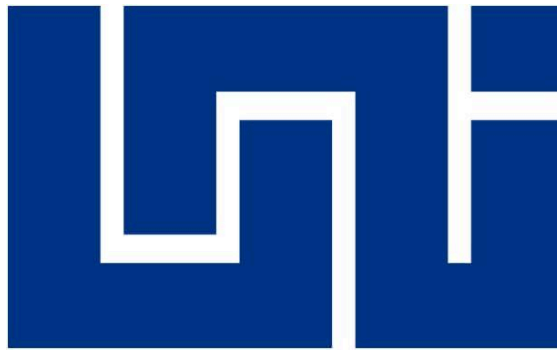


UNIVERSIDAD NACIONAL DE INGENIERÍA
RECINTO UNIVERSITARIO “SIMON BOLIVAR”
ÁREA DE CONOCIMIENTO TECNOLOGÍA DE LA INFORMACIÓN Y
LA COMUNICACIÓN
INGENIERÍA DE SISTEMAS



**Aplicación de Buenas Prácticas de Programación en la Refactorización del
Proyecto EntidadFinanciera2M6**

AUTORES:

1. Jasareth Enmanuel Mendoza Flores.
2. Jhonny Josue García Blas.
3. Melania del Carmen Cedeño Meza.
4. Eduardo Antonio Chavarria Namoyure.
5. Rommel Bayardo Calero Ruiz.
6. Jonathan Alexander Joaquín Arias.

GRUPO: 2M6-SIS-S.

DOCENTE: Ing. Abel Edmundo Marin Reyes.

MANAGUA, MAYO 24 DE 2025

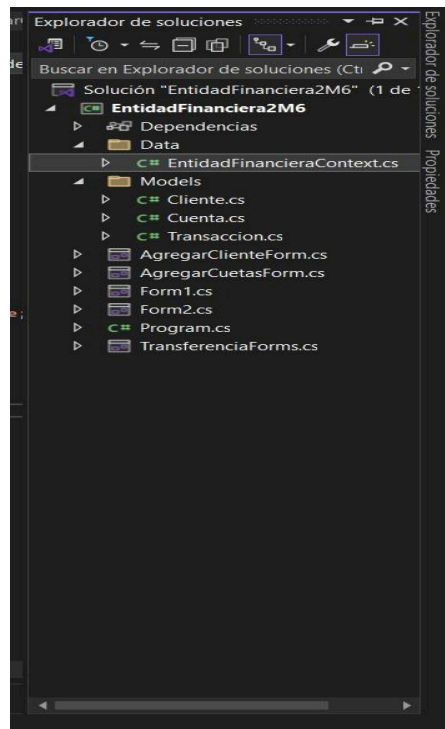
Introducción

En el presente documento se detallan las mejoras aplicadas al proyecto original titulado EntidadFinanciera2M6, con el objetivo de implementar buenas prácticas de programación. Estas mejoras fueron realizadas por el equipo como parte de la asignación grupal y se enfocan en mejorar la calidad del código, la separación de responsabilidades, la legibilidad, el mantenimiento del software y el correcto manejo de errores.

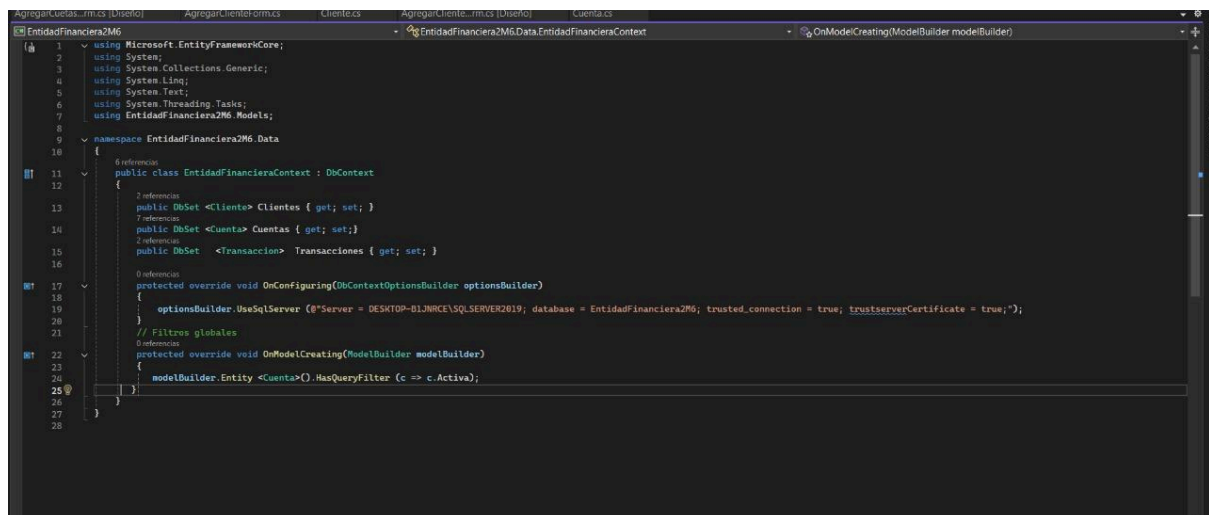
A continuación, se describen los cambios realizados en los diferentes archivos del proyecto, incluyendo capturas del antes y después, así como una justificación para cada mejora implementada.

Antes de los cambios realizados

Diseño estructural del sistema:

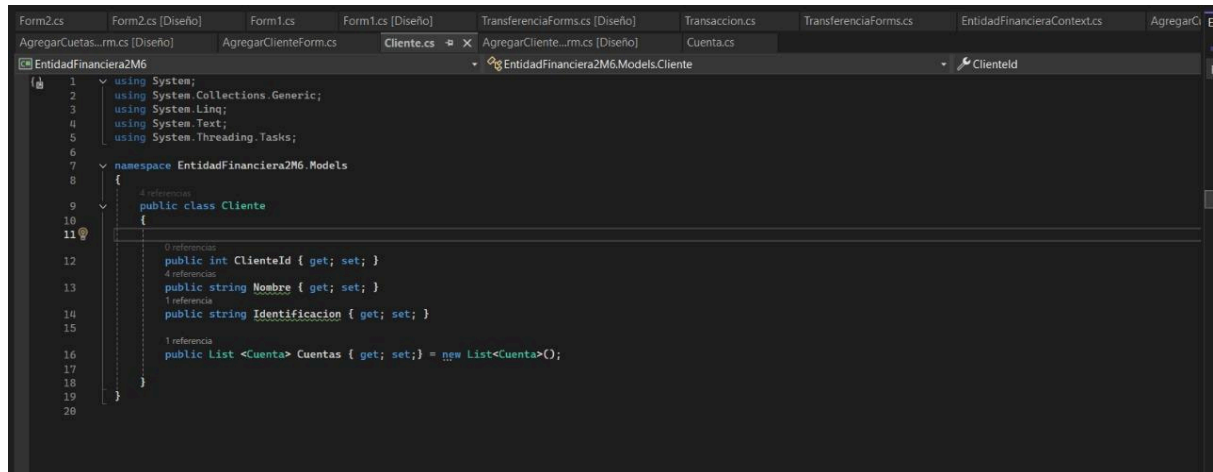


Clase de conexión a la base de datos:



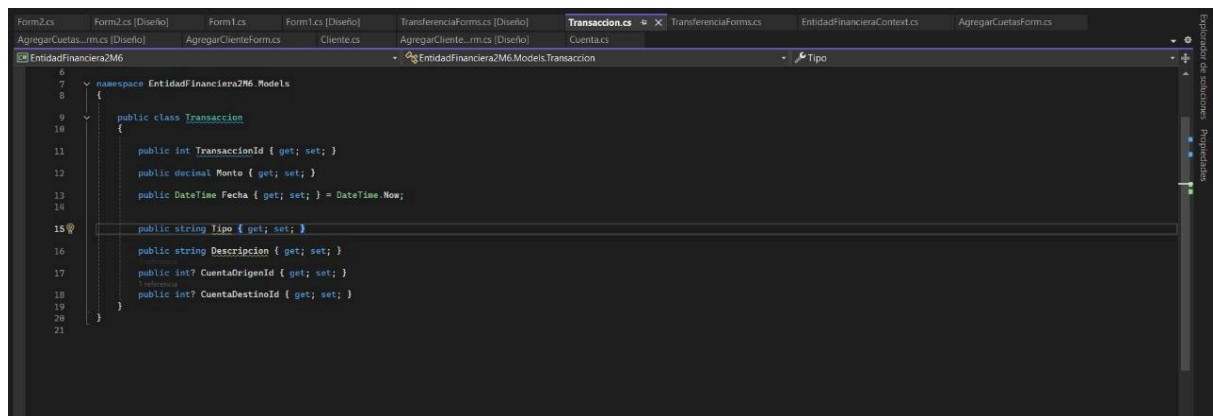
Clases

Clase cliente:



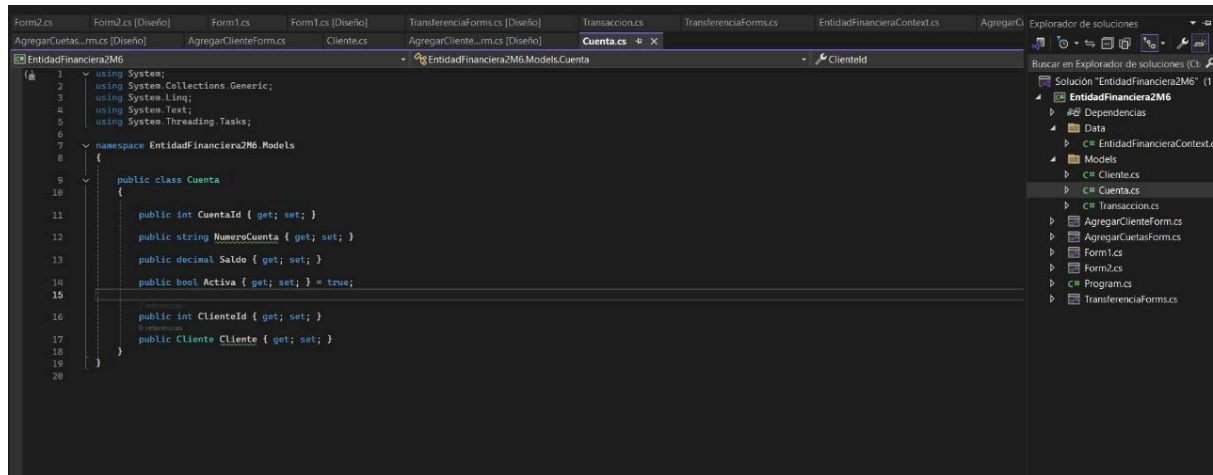
```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace EntidadFinanciera2M6.Models
8 {
9     public class Cliente
10     {
11
12         public int ClienteId { get; set; }
13         public string Nombre { get; set; }
14         public string Identificacion { get; set; }
15
16         public List<Cuenta> Cuentas { get; set; } = new List<Cuenta>();
17     }
18 }
19
20
```

Clase transacción:



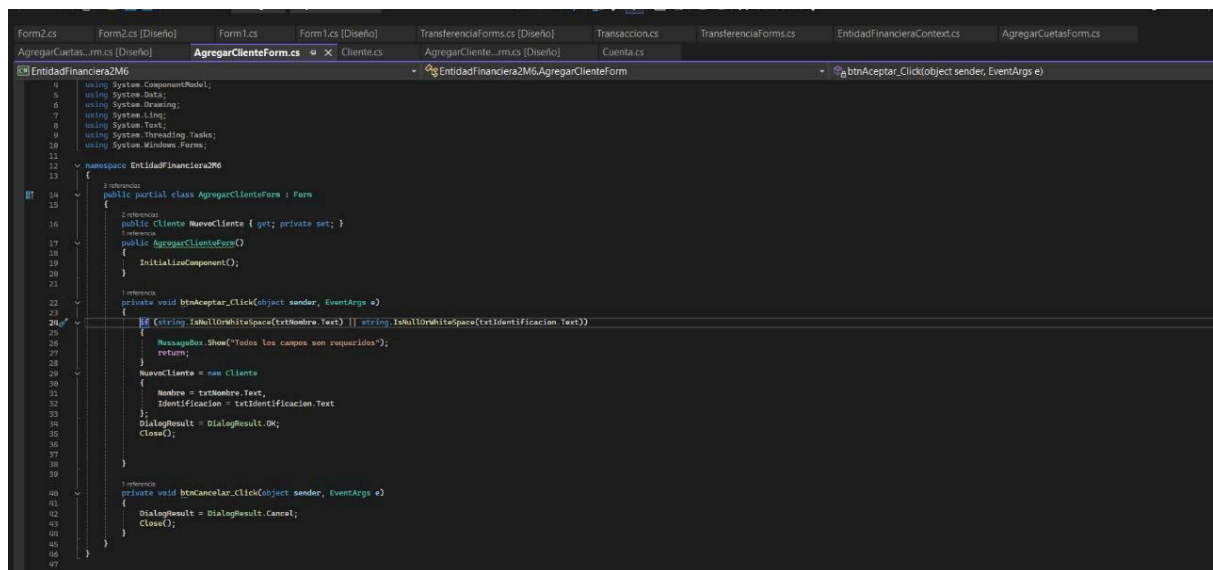
```
6
7 namespace EntidadFinanciera2M6.Models
8 {
9     public class Transaccion
10     {
11         public int TransaccionId { get; set; }
12         public decimal Monto { get; set; }
13         public DateTime Fecha { get; set; } = DateTime.Now;
14
15         public string Tipo { get; set; }
16         public string Descripcion { get; set; }
17         public int? CuentaOrigenId { get; set; }
18         public int? CuentaDestinoId { get; set; }
19     }
20 }
21
```

Clase cuenta:



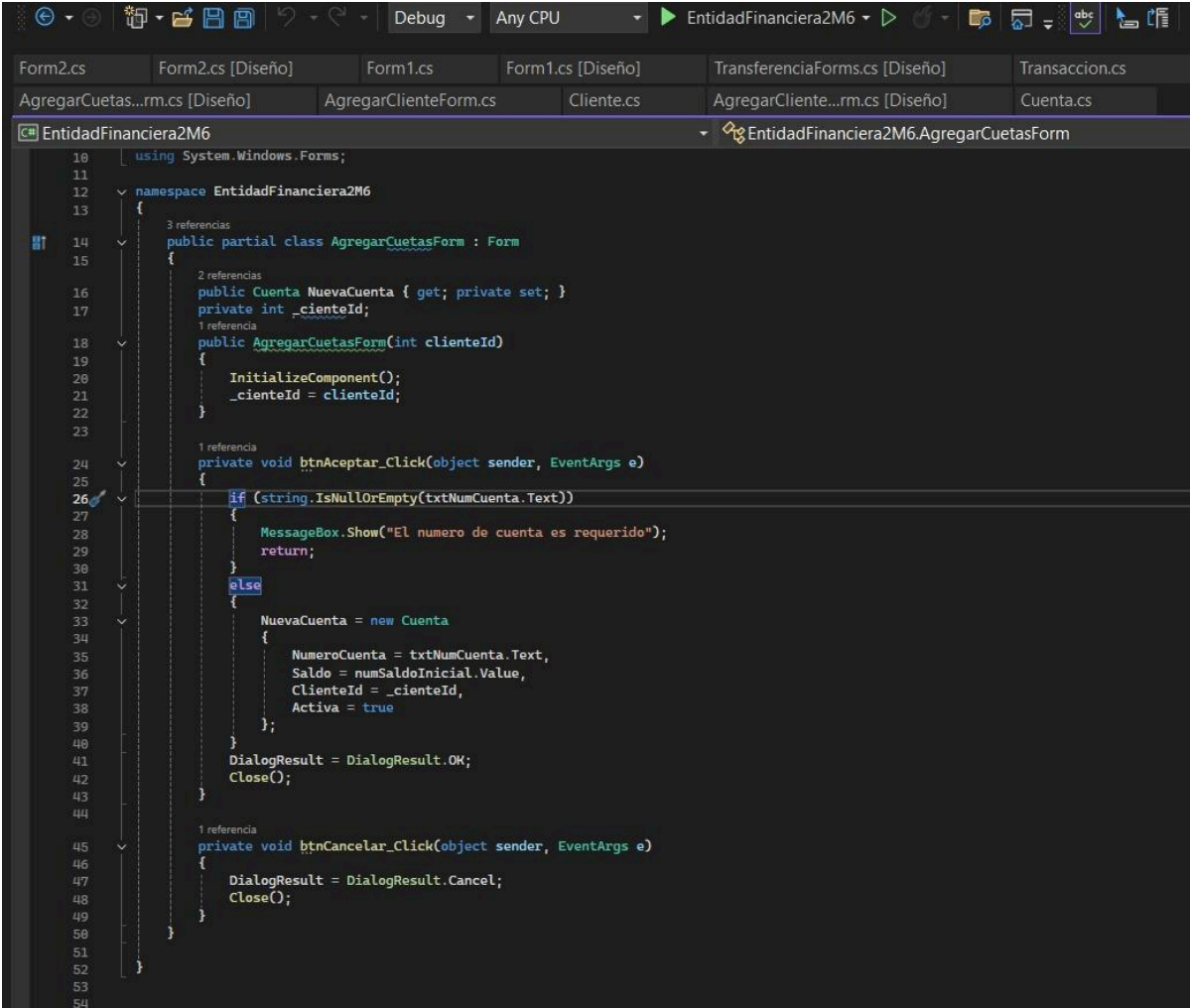
```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace EntidadFinanciera2M6.Models
8 {
9     public class Cuenta
10     {
11         public int CuentaId { get; set; }
12         public string NumeroCuenta { get; set; }
13         public decimal Saldo { get; set; }
14         public bool Activa { get; set; } = true;
15
16         public int ClienteId { get; set; }
17         public Cliente Cliente { get; set; }
18     }
19 }
20
```

AgregarClienteForm:



```
4 using System.ComponentModel;
5 using System.Data;
6 using System.Drawing;
7 using System.Linq;
8 using System.Text;
9 using System.Threading.Tasks;
10 using System.Windows.Forms;
11
12 namespace EntidadFinanciera2M6
13 {
14     public partial class AgregarClienteForm : Form
15     {
16         public Cliente NuevoCliente { get; private set; }
17         public AgregarClienteForm()
18         {
19             InitializeComponent();
20         }
21
22         private void btnAceptar_Click(object sender, EventArgs e)
23         {
24             if (string.IsNullOrWhiteSpace(txtNombre.Text) || string.IsNullOrWhiteSpace(txtIdentificacion.Text))
25             {
26                 MessageBox.Show("Todos los campos son requeridos");
27                 return;
28             }
29             NuevoCliente = new Cliente
30             {
31                 Nombre = txtNombre.Text,
32                 Identificacion = txtIdentificacion.Text
33             };
34             DialogResult = DialogResult.OK;
35             Close();
36         }
37
38         private void btnCancelar_Click(object sender, EventArgs e)
39         {
40             DialogResult = DialogResult.Cancel;
41             Close();
42         }
43     }
44 }
45
46
47
```

AgregarCuentaForm:



```
10 using System.Windows.Forms;
11
12 namespace EntidadFinanciera2M6
13 {
14     3 referencias
15     public partial class AgregarCuentasForm : Form
16     {
17         2 referencias
18         public Cuenta NuevaCuenta { get; private set; }
19         private int _clienteId;
20         1 referencia
21         public AgregarCuentasForm(int clienteId)
22         {
23             InitializeComponent();
24             _clienteId = clienteId;
25         }
26
27         1 referencia
28         private void btnAceptar_Click(object sender, EventArgs e)
29         {
30             if (string.IsNullOrEmpty(txtNumCuenta.Text))
31             {
32                 MessageBox.Show("El numero de cuenta es requerido");
33                 return;
34             }
35             else
36             {
37                 NuevaCuenta = new Cuenta
38                 {
39                     NumeroCuenta = txtNumCuenta.Text,
40                     Saldo = numSaldoInicial.Value,
41                     ClienteId = _clienteId,
42                     Activa = true
43                 };
44                 DialogResult = DialogResult.OK;
45                 Close();
46             }
47         }
48
49         1 referencia
50         private void btnCancelar_Click(object sender, EventArgs e)
51         {
52             DialogResult = DialogResult.Cancel;
53             Close();
54         }
55     }
56 }
```

Form1:

```
EntidadFinanciera2M6
1 using EntidadFinanciera2M6.Data;
2 using Microsoft.EntityFrameworkCore;
3 using EntidadFinanciera2M6.Models;
4
5 namespace EntidadFinanciera2M6
6 {
7     3 referencias
8     public partial class Form1 : Form
9     {
10         private EntidadFinancieraContext _db = new EntidadFinancieraContext();
11         1 referencia
12         public Form1()
13         {
14             InitializeComponent();
15             CargarDatos();
16         }
17
18         5 referencias
19         private void CargarDatos()
20         {
21             //Ejecucion de consultas
22             //Uso de linq-filtrado de informacion
23
24             var clientes = _db.Clientes.Include(c => c.Cuentas).ToList();
25             var cuentas = _db.Cuentas.Include(c => c.Cliente).Where(c => c.Activa).
26                 Select(c => new
27                 {
28                     c.CuentaId,
29                     c.NumeroCuenta,
30                     c.Saldo,
31                     c.Activa,
32                     c.ClienteId,
33                     c.Cliente.Nombre
34                 }).ToList();
35
36             // dgvClientes.DataSource = _db.Clientes.ToList();
37             dgvClientes.DataSource = clientes;
38             dgvCuentas.DataSource = cuentas;
39         }
40
41         1 referencia
42         private void btnAgregarCliente_Click(object sender, EventArgs e)
43         {
44             var form = new AgregarClienteForm();
45             if (form.ShowDialog() == DialogResult.OK)
46             {
47                 _db.Clientes.Add(form.NuevoCliente);
48                 _db.SaveChanges();
49                 CargarDatos();
50             }
51         }
52     }
53 }
54
55 72 % No se encontraron problemas.
```

```
EntidadFinanciera2M6
49 1 referencia
50 private void btnAgregarCuenta_Click(object sender, EventArgs e)
51 {
52     if (dgvClientes.SelectedRows.Count == 0)
53     {
54         MessageBox.Show("Seleccione un cliente primero");
55         return;
56     }
57     var clienteId = (int)dgvClientes.SelectedRows[0].Cells["ClienteId"].Value;
58     var form = new AgregarCuentasForm(clienteId);
59     if (form.ShowDialog() == DialogResult.OK)
60     {
61         _db.Cuentas.Add(form.NuevaCuenta);
62         _db.SaveChanges();
63         CargarDatos();
64     }
65 }
66
67 1 referencia
68 private void btnDesactivarCuenta_Click(object sender, EventArgs e)
69 {
70     if (dgvCuentas.SelectedRows.Count == 0)
71     {
72         MessageBox.Show("Seleccione una cuenta para desactivar");
73         return;
74     }
75     var cuentaId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;
76     var cuenta = _db.Cuentas.Find(cuentaId);
77     if (cuenta != null)
78     {
79         cuenta.Activa = false;
80         _db.SaveChanges();
81         CargarDatos();
82     }
83 }
84
85 1 referencia
86 private void RealizarTransacc(int origenCuenta, int destinoCuenta, decimal monto)
87 {
88     //Transacciones completas
89     //Implementacion de transacciones
90     //Nivel de aislamiento (serializable para operaciones financieras)
91     using var transaccion = _db.Database.BeginTransaction(System.Data.IsolationLevel.Serializable);
92     try
93     {
94         var cuentaOrigen = _db.Cuentas.FirstOrDefault(c => c.CuentaId == origenCuenta);
95         var cuentaDestino = _db.Cuentas.FirstOrDefault(c => c.CuentaId == destinoCuenta);
96
97         if (cuentaOrigen.Saldo < monto)
98         {
99             throw new Exception("Saldo insuficiente");
100         }
101     }
102 }
103
104 72 % No se encontraron problemas.
```

```

EntidadFinanciera2Mb
EntidadFinanciera2Mb.Form1
97      cuentaDestino.Saldo -= monto;
98      db.Transacciones.Add(new Transaccion
99      {
100          Monto = monto,
101          Fecha = DateTime.Now,
102          Tipo = "Transferencia",
103          Descripcion = "Transferencia",
104          CuentaOrigenId = origenCuenta,
105          CuentaDestinoId = destinoCuenta
106      });
107      _db.SaveChanges();
108      //Completamos la transaccion
109      transaccion.Commit();
110      MessageBox.Show("Trasferencia realizada con éxito");
111      CargarDatos();
112  }
113  }
114  catch (Exception ex)
115  {
116      //reversion de transacciones
117      transaccion.Rollback();
118      MessageBox.Show($"Error en la transferencia: {ex.Message}");
119  }
120  }
121
122  1 referencia
123  private void btnTransferencia_Click(object sender, EventArgs e)
124  {
125      if (dgvCuentas.SelectedRows.Count != 2)
126      {
127          MessageBox.Show("Seleccione exactamente 2 cuentas");
128          return;
129      }
130      else
131      {
132          var cuentaOrigenId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;
133          var cuentaDestinoId = (int)dgvCuentas.SelectedRows[1].Cells["CuentaId"].Value;
134          var form = new TransferenciaForm(cuentaOrigenId, cuentaDestinoId);
135          if (form.ShowDialog() == DialogResult.OK)
136          {
137              RealizarTransacc(cuentaOrigenId, cuentaDestinoId, form.Monto);
138          }
139      }
140  }
141
142  1 referencia
143  private void button1_Click(object sender, EventArgs e)
144  {

```



```

121
122 1 referencia
123 private void btnTransferencia_Click(object sender, EventArgs e)
124 {
125     if (dgvCuentas.SelectedRows.Count != 2)
126     {
127         MessageBox.Show("Seleccione exactamente 2 cuentas");
128         return;
129     }
130     else
131     {
132         var cuentaOrigenId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;
133         var cuentaDestinoId = (int)dgvCuentas.SelectedRows[1].Cells["CuentaId"].Value;
134
135         var form = new TransferenciaForms(cuentaOrigenId, cuentaDestinoId);
136         if (form.ShowDialog() == DialogResult.OK)
137         {
138             RealizarTransacc(cuentaOrigenId, cuentaDestinoId, form.Monto);
139         }
140     }
141 }
142
143 1 referencia
144 private void button1_Click(object sender, EventArgs e)
145 {
146     var form = new Form2();
147     form.ShowDialog();
148 }
149
150

```

Form2:

```

Agregarcuentas...m6 [Diseño]  Agregarcuentas...m6 [Diseño]  Clientes  Agregarcuentas...m6 [Diseño]  Cuentas
EntidadFinanciera2M6
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10 using System.Windows.Forms;
11
12 namespace EntidadFinanciera2M6
13 {
14
15     3 referencias
16     public partial class Form2 : Form
17     {
18         1 referencia
19         private EntidadFinancieraContext ef = new EntidadFinancieraContext();
20         public Form2()
21         {
22             InitializeComponent();
23             Cargar();
24         }
25
26         1 referencia
27         private void Cargar()
28         {
29             dataGridView1.DataSource = ef.Transacciones.ToList();
30         }
31
32         1 referencia
33         private void button1_Click(object sender, EventArgs e)
34         {
35
36         }
37     }
38 }

```

TransferenciaForms:

```
EntidadFinanciera2M6
using EntidadFinanciera2M6.Data;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace EntidadFinanciera2M6
{
    3 referencias
    public partial class TransferenciaForms : Form
    {
        2 referencias
        public decimal Monto { get; set; }
        private int _cuentaOrigenId;
        private int _cuentaDestinoId;
        private EntidadFinancieraContext _Ef;
        1 referencia
        public TransferenciaForms(int cuentaOrigenId, int cuentaDestinoId)
        {
            _cuentaOrigenId = cuentaOrigenId;
            _cuentaDestinoId = cuentaDestinoId;
            _Ef = new EntidadFinancieraContext();
            InitializeComponent();
            CargarCuentas();
        }
        1 referencia
        private void CargarCuentas()
        {
            var cuentaOrigen = _Ef.Cuentas.
                Include(c => c.Cliente).
                First(c => c.CuentaId == _cuentaOrigenId);

            var cuentaDestino = _Ef.Cuentas.
                Include(c => c.Cliente).
                First(c => c.CuentaId == _cuentaDestinoId);

            lblCuentaOrigen.Text = $"Cuenta Origen: {cuentaOrigen.Cliente.Nombre} - {cuentaOrigen.NumeroCuenta}";
            lblCuentaDestino.Text = $"Cuenta Destino: {cuentaDestino.Cliente.Nombre} - {cuentaDestino.NumeroCuenta}";
            lblSaldo.Text = $"Saldo Disponible {cuentaOrigen.Saldo:c}";
        }
        1 referencia
        private void btnAceptar_Click(object sender, EventArgs e)
        {
            if (numMonto.Value > 0)
            {
                Monto = numMonto.Value;
                DialogResult = DialogResult.OK;
                Close();
            }
            else
            {
                MessageBox.Show("El monto tiene que ser mayor a 0");
            }
        }
    }
}
```

```
Form2.cs*   Form2.cs [Diseño]*   Form1.cs   Form1.cs [Diseño]   TransferenciaForms.cs [Diseño]   Transaccion.cs
AgregarCuetas...rm.cs [Diseño]   AgregarClienteForm.cs   Cliente.cs   AgregarCliente...rm.cs [Diseño]   Cuenta.cs

EntidadFinanciera2M6
    _Ef = new EntidadFinancieraContext();
    InitializeComponent();
    CargarCuentas();
}
1 referencia
private void CargarCuentas()
{
    var cuentaOrigen = _Ef.Cuentas.
        Include(c => c.Cliente).
        First(c => c.CuentaId == _cuentaOrigenId);

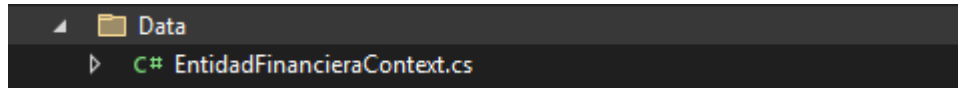
    var cuentaDestino = _Ef.Cuentas.
        Include(c => c.Cliente).
        First(c => c.CuentaId == _cuentaDestinoId);

    lblCuentaOrigen.Text = $"Cuenta Origen: {cuentaOrigen.Cliente.Nombre} - {cuentaOrigen.NumeroCuenta}";
    lblCuentaDestino.Text = $"Cuenta Destino: {cuentaDestino.Cliente.Nombre} - {cuentaDestino.NumeroCuenta}";
    lblSaldo.Text = $"Saldo Disponible {cuentaOrigen.Saldo:c}";
}
1 referencia
private void btnAceptar_Click(object sender, EventArgs e)
{
    if (numMonto.Value > 0)
    {
        Monto = numMonto.Value;
        DialogResult = DialogResult.OK;
        Close();
    }
    else
    {
        MessageBox.Show("El monto tiene que ser mayor a 0");
    }
}
}
```

Después de los cambios realizados

1. Carpeta **Data**

Archivo: **EntidadFinancieraContext.cs**



Estado: No se realizaron modificaciones.

Motivo:

La clase EntidadFinancieraContext cumple su función como clase de contexto de Entity Framework Core, encargada de gestionar la conexión con la base de datos y mapear las entidades. Dado que su implementación ya sigue buenas prácticas y no contiene lógica adicional ni duplicada, no fue necesario realizar cambios en esta sección.

2. Carpeta **Models**

Archivos: **Cliente.cs**, **Cuenta.cs**, **Transaccion.cs**



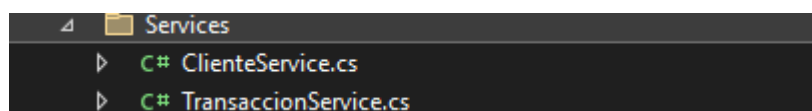
Estado: No se realizaron modificaciones.

Motivo:

Estos archivos contienen únicamente la definición de las entidades de datos utilizadas en el sistema. Las clases están correctamente estructuradas y cumplen con su propósito de representar los modelos del dominio. No contienen lógica compleja ni errores evidentes, por lo tanto, no se aplicaron modificaciones.

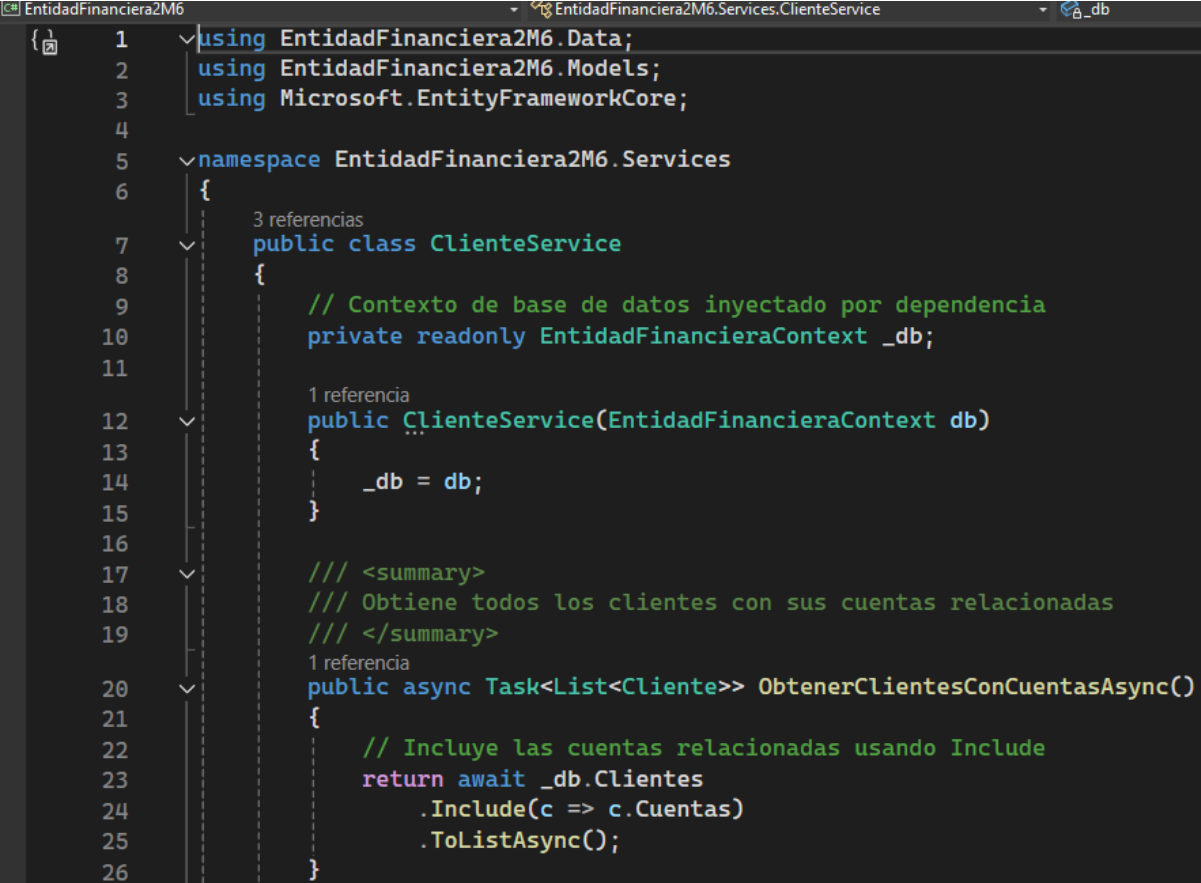
3. Carpeta **Services**

Archivo: **ClienteService.cs** (nuevo archivo creado)



¿Qué se hizo?

- Se creó esta clase para centralizar la lógica relacionada con la gestión de clientes y sus cuentas bancarias.
- Se implementaron métodos específicos con responsabilidades bien definidas (siguiendo el principio de responsabilidad única).
- Se agregaron comentarios explicativos en cada método para facilitar la comprensión del código.
- Se aplicaron validaciones robustas a las entradas, con mensajes claros en caso de errores.



```
1  using EntidadFinanciera2M6.Data;
2  using EntidadFinanciera2M6.Models;
3  using Microsoft.EntityFrameworkCore;
4
5  namespace EntidadFinanciera2M6.Services
6  {
7      3 referencias
8      public class ClienteService
9      {
10         // Contexto de base de datos inyectado por dependencia
11         private readonly EntidadFinancieraContext _db;
12
13         1 referencia
14         public ClienteService(EntidadFinancieraContext db)
15         {
16             _db = db;
17         }
18
19         /// <summary>
20         /// Obtiene todos los clientes con sus cuentas relacionadas
21         /// </summary>
22         1 referencia
23         public async Task<List<Cliente>> ObtenerClientesConCuentasAsync()
24         {
25             // Incluye las cuentas relacionadas usando Include
26             return await _db.Clientes
27                 .Include(c => c.Cuentas)
28                 .ToListAsync();
29         }
30     }
```

```

28      /// <summary>
29      /// Agrega un nuevo cliente al sistema
30      /// </summary>
31      1 referencia
32      public async Task<Cliente> AgregarClienteAsync(Cliente cliente)
33      {
34          // Validar que el nombre no esté vacío
35          if (string.IsNullOrEmpty(cliente.Nombre))
36          {
37              throw new ArgumentException("El nombre del cliente no puede estar vacío");
38          }
39
40          await _db.Clientes.AddAsync(cliente);
41          await _db.SaveChangesAsync();
42          return cliente;
43      }
44
45      /// <summary>
46      /// Obtiene todas las cuentas activas con información del cliente
47      /// </summary>
48      1 referencia
49      public async Task<List<Cuenta>> ObtenerCuentasActivasAsync()
50      {
51          // Solo cuentas activas y con información del cliente
52          return await _db.Cuentas
53              .Include(c => c.Cliente)
54              .Where(c => c.Activa)
55              .ToListAsync();

```

```

56      /// <summary>
57      /// Agrega una nueva cuenta para un cliente
58      /// </summary>
59      1 referencia
60      public async Task<Cuenta> AgregarCuentaAsync(Cuenta cuenta)
61      {
62          // Validar que el ID del cliente sea válido
63          if (cuenta.ClienteId <= 0)
64          {
65              throw new ArgumentException("El ID del cliente no es válido");
66          }
67
68          // Verificar que el cliente exista
69          var cliente = await _db.Clientes.FindAsync(cuenta.ClienteId);
70          if (cliente == null)
71          {
72              throw new InvalidOperationException("El cliente especificado no existe");
73          }
74
75          await _db.Cuentas.AddAsync(cuenta);
76          await _db.SaveChangesAsync();
77          return cuenta;
78      }

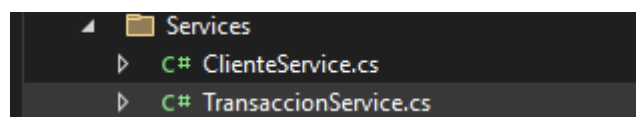
```

```
79  // <summary>
80  // Desactiva una cuenta existente
81  // </summary>
82  1 referencia
83  public async Task<bool> DesactivarCuentaAsync(int cuentaId)
84  {
85      var cuenta = await _db.Cuentas.FindAsync(cuentaId);
86      if (cuenta == null)
87      {
88          throw new InvalidOperationException("La cuenta especificada no existe");
89      }
90      cuenta.Activa = false;
91      await _db.SaveChangesAsync();
92      return true;
93  }
94  }
95  }
```

¿Por qué se hizo?

- La separación de la lógica de negocio de la interfaz gráfica es una práctica esencial para la mantenibilidad del código.
- Esta clase facilita el aislamiento de funcionalidades, permite una mejor reutilización de código y simplifica la realización de pruebas unitarias.

Archivo: **TransaccionService.cs** (nuevo archivo creado)



¿Qué se hizo?

- Se creó una clase dedicada para manejar todas las transacciones financieras del sistema (transferencias entre cuentas).
- Se encapsuló el proceso de transferencia incluyendo validaciones y manejo de errores.
- Se implementó el uso de transacciones de base de datos para garantizar la integridad de la operación.
- Se agregaron comentarios explicativos en cada parte del código.

```

EntidadFinanciera2M6
EntidadFinanciera2M6.Services.TransaccionService
RealizarTransferenciaAsync(int cuentaOrigenId, int cuentaDe

1 using EntidadFinanciera2M6.Data;
2 using EntidadFinanciera2M6.Models;
3 using Microsoft.EntityFrameworkCore;
4
5 namespace EntidadFinanciera2M6.Services
6 {
7     3 referencias
8     public class TransaccionService
9     {
10         // Contexto de base de datos inyectado por dependencia
11         private readonly EntidadFinancieraContext _db;
12
13         1 referencia
14         public TransaccionService(EntidadFinancieraContext db)
15         {
16             _db = db;
17         }
18
19         /// <summary>
20         /// Realiza una transferencia entre cuentas con manejo de transacciones
21         /// </summary>
22         /// <param name="cuentaOrigenId">ID de la cuenta origen</param>
23         /// <param name="cuentaDestinoId">ID de la cuenta destino</param>
24         /// <param name="monto">Monto a transferir</param>
25         /// <returns>True si la transferencia fue exitosa, False en caso contrario</returns>
26         1 referencia
27         public async Task<bool> RealizarTransferenciaAsync(int cuentaOrigenId, int cuentaDestinoId, decimal monto)
28         {
29             // Se utiliza una transacción para asegurar la integridad de la operación
30             using var transaccion = await _db.Database.BeginTransactionAsync(System.Data.IsolationLevel.Serializable);
31             try
32             {
33                 // Buscar las cuentas involucradas
34                 var cuentaOrigen = await _db.Cuentas.FirstOrDefaultAsync(c => c.CuentaId == cuentaOrigenId);
35                 var cuentaDestino = await _db.Cuentas.FirstOrDefaultAsync(c => c.CuentaId == cuentaDestinoId);
36
37                 // Validar que ambas cuentas existan
38                 if (cuentaOrigen == null || cuentaDestino == null)
39                 {
40                     throw new InvalidOperationException("Una o ambas cuentas no existen");
41                 }
42             }
43         }
44     }
45 }

```

```

EntidadFinanciera2M6
EntidadFinanciera2M6.Services.TransaccionService
RealizarTransferenciaAsync(int cuentaOrigenId, int cuentaDestinoId

40 // Validar que ambas cuentas estén activas
41 if (!cuentaOrigen.Activa || !cuentaDestino.Activa)
42 {
43     throw new InvalidOperationException("Una o ambas cuentas están inactivas");
44 }
45
46 // Validar que la cuenta origen tenga saldo suficiente
47 if (cuentaOrigen.Saldo < monto)
48 {
49     throw new InvalidOperationException("Saldo insuficiente en la cuenta origen");
50 }
51
52 // Realizar la transferencia
53 cuentaOrigen.Saldo -= monto;
54 cuentaDestino.Saldo += monto;
55
56 // Registrar la transacción en la base de datos
57 var nuevaTransaccion = new Transaccion
58 {
59     Monto = monto,
60     Fecha = DateTime.Now,
61     Tipo = "Transferencia",
62     Descripcion = $"Transferencia de cuenta {cuentaOrigen.NumeroCuenta} a {cuentaDestino.NumeroCuenta}",
63     CuentaOrigenId = cuentaOrigenId,
64     CuentaDestinoId = cuentaDestinoId
65 };
66
67 await _db.Transacciones.AddAsync(nuevaTransaccion);
68 await _db.SaveChangesAsync();
69 await transaccion.CommitAsync(); // Confirmar la transacción
70
71 return true;
72 }
73 catch (Exception)
74 {
75     // Si ocurre un error, se revierte la transacción
76     await transaccion.RollbackAsync();
77     throw;
78 }
79
80 }

```

¿Por qué se hizo?

- La encapsulación de la lógica financiera en una clase específica mejora la claridad, seguridad y mantenibilidad del sistema.
- Al centralizar el manejo de transacciones, se minimiza la posibilidad de errores, se mejora la trazabilidad y se facilita la auditoría del sistema.

Una de las buenas prácticas implementadas en el archivo **TransaccionService.cs** es el uso de comentarios XML, los cuales permiten documentar formalmente los métodos públicos. Este tipo de comentarios son reconocidos por el entorno de desarrollo (por ejemplo, Visual Studio) y facilitan la comprensión del propósito y funcionamiento del código al mostrar ayuda contextual durante la programación.

En este caso, se añadió un bloque de documentación al método encargado de realizar la transferencia entre cuentas (<summary>, <returns>, <param>).

¿Qué significa este bloque de comentario?

<summary>: Resume de forma breve pero clara la funcionalidad principal del método. En este caso, indica que el método realiza una transferencia con control de transacciones (para garantizar la integridad de los datos).

<param>: Documenta los parámetros que recibe el método. Aquí se explican los tres parámetros: la cuenta de origen, la cuenta de destino y el monto a transferir.

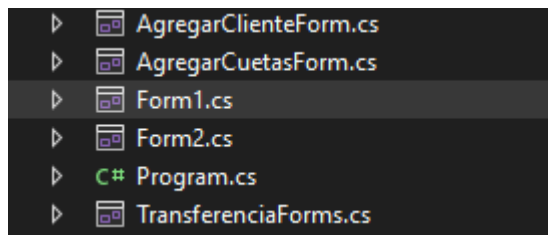
<returns>: Indica lo que devuelve el método. En este caso, un valor booleano que informa si la transferencia se completó con éxito o no.

¿Por qué se agregó esto?

Este tipo de documentación:

- Mejora la comprensión del código por parte de cualquier miembro del equipo o futuros desarrolladores.
- Es útil para la generación automática de documentación técnica del proyecto.
- Refuerza la mantenibilidad y profesionalismo del proyecto, alineándose con estándares de desarrollo utilizados en entornos empresariales y académicos.

4. Archivo: **Form1.cs** (modificado)



¿Qué se hizo?

- Se agregaron comentarios explicativos a lo largo del archivo, especialmente en la inicialización, manejo de eventos y llamadas a servicios.
- Se reorganizó el código para separar la lógica de negocio de la interfaz gráfica.
- Se conectaron los servicios ClienteService y TransaccionService para delegar correctamente la lógica.
- Se mejoró el manejo de excepciones usando bloques try-catch adecuados y mensajes de error comprensibles.
- Se eliminó código duplicado y se mejoró la estructura general del formulario.

```

EntidadFinanciera2M6
EntidadFinanciera2M6.Form1
1  using EntidadFinanciera2M6.Data;
2  using EntidadFinanciera2M6.Models;
3  using EntidadFinanciera2M6.Services;
4  using Microsoft.EntityFrameworkCore;
5
6  namespace EntidadFinanciera2M6
7  {
8      3 referencias
      public partial class Form1 : Form
9      {
10         // Contexto y servicios para manejar la lógica de negocio
11         private readonly EntidadFinancieraContext _db;
12         private readonly ClienteService _clienteService;
13         private readonly TransaccionService _transaccionService;
14
15         1 referencia
16         public Form1()
17         {
18             InitializeComponent();
19             // Inicialización del contexto y los servicios
20             _db = new EntidadFinancieraContext();
21             _clienteService = new ClienteService(_db);
22             _transaccionService = new TransaccionService(_db);
23             // Cargar los datos al iniciar el formulario
24             _ = CargarDatosAsync();
25
26         // Carga los datos de clientes y cuentas en los DataGridView
27         5 referencias
28         private async Task CargarDatosAsync()
29         {
30             try
31             {
32                 // Obtener clientes y cuentas activas usando los servicios
33                 var clientes = await _clienteService.ObtenerClientesConCuentasAsync();
34                 var cuentas = await _clienteService.ObtenerCuentasActivasAsync();
35
36                 // Asignar los datos a los controles de la interfaz
37                 dgvClientes.DataSource = clientes;
38                 dgvCuentas.DataSource = cuentas.Select(c => new
39                 {
40                     c.CuentaId,
41                     c.NumeroCuenta,
42                     c.Saldo,
43                     c.Activa,
44                     c.ClienteId,

```

```
EntidadFinanciera2M6
EntidadFinanciera2M6.Form1
db

44      c.Cliente.Nombre
45      }).ToList();
46      }
47      catch (Exception ex)
48      {
49          // Manejo de errores: mostrar mensaje al usuario
50          MessageBox.Show($"Error al cargar los datos: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
51      }
52  }
53
54      // Evento para agregar un nuevo cliente
55      1 referencia
56      private async void btnAgregarCliente_Click(object sender, EventArgs e)
57      {
58          try
59          {
60              var form = new AgregarClienteForm();
61              if (form.ShowDialog() == DialogResult.OK)
62              {
63                  // Usar el servicio para agregar el cliente
64                  await _clienteService.AgregarClienteAsync(form.NuevoCliente);
65                  await CargarDatosAsync();
66              }
67          }
68          catch (Exception ex)
69          {
70              // Mostrar error si ocurre algún problema
71              MessageBox.Show($"Error al agregar cliente: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
72          }
73      }
74
75      // Evento para agregar una nueva cuenta a un cliente seleccionado
76      1 referencia
77      private async void btnAgregarCuenta_Click(object sender, EventArgs e)
78      {
79          try
80          {
81              if (dgvClientes.SelectedRows.Count == 0)
82              {
83                  MessageBox.Show("Seleccione un cliente primero", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Warning);
84                  return;
85              }
86
87              var clienteId = (int)dgvClientes.SelectedRows[0].Cells["ClienteId"].Value;
88              var form = new AgregarCuentasForm(clienteId);
```

```
EntidadFinanciera2M6
EntidadFinanciera2M6.Form1
db

86      var form = new AgregarCuentasForm(clienteId);
87
88      if (form.ShowDialog() == DialogResult.OK)
89      {
90          // Usar el servicio para agregar la cuenta
91          await _clienteService.AgregarCuentaAsync(form.NuevaCuenta);
92          await CargarDatosAsync();
93      }
94      catch (Exception ex)
95      {
96          MessageBox.Show($"Error al agregar cuenta: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
97      }
98  }
99
100      // Evento para desactivar una cuenta seleccionada
101      1 referencia
102      private async void btnDesactivarCuenta_Click(object sender, EventArgs e)
103      {
104          try
105          {
106              if (dgvCuentas.SelectedRows.Count == 0)
107              {
108                  MessageBox.Show("Seleccione una cuenta para desactivar", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Warning);
109                  return;
110              }
111
112              var cuentaId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;
113              // Usar el servicio para desactivar la cuenta
114              await _clienteService.DesactivarCuentaAsync(cuentaId);
115              await CargarDatosAsync();
116          }
117          catch (Exception ex)
118          {
119              MessageBox.Show($"Error al desactivar cuenta: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
120          }
121      }
122
123      // Evento para realizar una transferencia entre dos cuentas seleccionadas
124      1 referencia
125      private async void btnTransferencia_Click(object sender, EventArgs e)
126      {
127          try
128          {
129              if (dgvCuentas.SelectedRows.Count != 2)
```

```

129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166

```

```

{
    MessageBox.Show("Seleccione exactamente 2 cuentas", "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    return;
}

var cuentaOrigenId = (int)dgvCuentas.SelectedRows[0].Cells["CuentaId"].Value;
var cuentaDestinoId = (int)dgvCuentas.SelectedRows[1].Cells["CuentaId"].Value;

var form = new TransferenciaForms(cuentaOrigenId, cuentaDestinoId);
if (form.ShowDialog() == DialogResult.OK)
{
    // Usar el servicio de transacciones para realizar la transferencia
    await _transaccionService.RealizarTransferenciaAsync(cuentaOrigenId, cuentaDestinoId, form.Monto);
    MessageBox.Show("Transferencia realizada con éxito", "Éxito", MessageBoxButtons.OK, MessageBoxIcon.Information);
    await CargarDatosAsync();
}
}
catch (Exception ex)
{
    MessageBox.Show($"Error en la transferencia: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

// Evento para abrir el formulario secundario (Form2)
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    var form = new Form2();
    form.ShowDialog();
}

// Liberar recursos al cerrar el formulario
0 referencias
protected override void OnFormClosing(FormClosingEventArgs e)
{
    base.OnFormClosing(e);
    _db.Dispose();
}
}

```

¿Por qué se hizo?

- Form1 es el punto central de la interacción del usuario, por lo que mantener un código limpio y estructurado aquí es fundamental.
- Estas mejoras permiten que el código sea más legible, fácil de mantener y extensible para futuras funcionalidades.

5. AgregarClienteForm, AgregarCuentasForm, Form2 y TransferenciaForms

```

> AgregarClienteForm.cs
> AgregarCuentasForm.cs
> Form1.cs
> Form2.cs
> C# Program.cs
> TransferenciaForms.cs

```

Estos otros formularios se mantienen igual; solo era cuestión de documentar el código.

AgregarClienteForm

```
EntidadFinanciera2M6
EntidadFinanciera2M6.AgregarClienteForm
btnCancelar_Click(object sender, EventArgs e)

1  using EntidadFinanciera2M6.Models;
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10 using System.Windows.Forms;
11
12 namespace EntidadFinanciera2M6
13 {
14     /// <summary>
15     /// Formulario que permite agregar un nuevo cliente
16     /// </summary>
17
18     3 referencias
19     public partial class AgregarClienteForm : Form
20     {
21         /// <summary>
22         /// Cliente creado desde el formulario
23         /// </summary>
24
25         2 referencias
26         public Cliente NuevoCliente { get; private set; }
27
28         /// <summary>
29         /// Constructor del formulario AgregarClienteForm
30         /// </summary>
31         ///
32         1 referencia
33         public AgregarClienteForm()
34         {
35             InitializeComponent();
36         }
37
38         /// <summary>
39         /// Valida los campos y asigna el nuevo cliente si es válido
40         /// </summary>
41
42         1 referencia
43         private void btnAceptar_Click(object sender, EventArgs e)
44         {
45             if (string.IsNullOrEmpty(txtNombre.Text) || string.IsNullOrEmpty(txtIdentificacion.Text))
46             {
47                 MessageBox.Show("Todos los campos son requeridos");
48                 return;
49             }
50             NuevoCliente = new Cliente
51             {
52                 Nombre = txtNombre.Text,
53                 Identificacion = txtIdentificacion.Text
54             };
55             DialogResult = DialogResult.OK;
56             Close();
57         }
58
59         /// <summary>
60         /// Cierra el formulario sin guardar cambios
61         /// </summary>
62
63         1 referencia
64         private void btnCancelar_Click(object sender, EventArgs e)
65         {
66             DialogResult = DialogResult.Cancel;
67             Close();
68         }
69     }
70 }
```

```
EntidadFinanciera2M6
EntidadFinanciera2M6.AgregarClienteForm

42     {
43         MessageBox.Show("Todos los campos son requeridos");
44         return;
45     }
46     NuevoCliente = new Cliente
47     {
48         Nombre = txtNombre.Text,
49         Identificacion = txtIdentificacion.Text
50     };
51     DialogResult = DialogResult.OK;
52     Close();
53 }
54
55
56
57     /// <summary>
58     /// Cierra el formulario sin guardar cambios
59     /// </summary>
60
61     1 referencia
62     private void btnCancelar_Click(object sender, EventArgs e)
63     {
64         DialogResult = DialogResult.Cancel;
65         Close();
66     }
67 }
```

AgregarCuentasForm

```
EntidadFinanciera2M6
EntidadFinanciera2M6.AgregarCuentasForm

1  using EntidadFinanciera2M6.Models;
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10 using System.Windows.Forms;
11
12 namespace EntidadFinanciera2M6
13 {
14     /// <summary>
15     /// Formulario para agregar una nueva cuenta a un cliente
16     /// </summary>
17
18     3 referencias
19     public partial class AgregarCuentasForm : Form
20     {
21         /// <summary>
22         /// Cuenta creada desde el formulario
23         /// </summary>
24         2 referencias
25         public Cuenta NuevaCuenta { get; private set; }
26         private int _clienteId;
27
28         /// <summary>
29         /// Constructor del formulario, recibe el ID del cliente
30         /// </summary>
31
32         1 referencia
33         public AgregarCuentasForm(int clienteId)
34         {
35             InitializeComponent();
36             _clienteId = clienteId;
37         }
38
39         /// <summary>
40         /// Crea la cuenta si el número es válido y cierra el formulario
41         /// </summary>
42
43         1 referencia
44         private void btnAceptar_Click(object sender, EventArgs e)
45         {
46             if (string.IsNullOrEmpty(txtNumCuenta.Text))
```

```
EntidadFinanciera2M6  EntidadFinanciera2M6.AgregarCuentasForm
42  if (string.IsNullOrEmpty(txtNumCuenta.Text))
43  {
44      MessageBox.Show("El numero de cuenta es requerido");
45      return;
46  }
47  else
48  {
49      NuevaCuenta = new Cuenta
50      {
51          NumeroCuenta = txtNumCuenta.Text,
52          Saldo = numSaldoInicial.Value,
53          ClienteId = _clienteId,
54          Activa = true
55      };
56  }
57  DialogResult = DialogResult.OK;
58  Close();
59  }
60
61  /// <summary>
62  /// Cancela la operación y cierra el formulario
63  /// </summary>
64  1 referencia
65  private void btnCancelar_Click(object sender, EventArgs e)
66  {
67      DialogResult = DialogResult.Cancel;
68      Close();
69  }
70
71 }
```

Form2

```
EntidadFinanciera2M6
EntidadFinanciera2M6.Form2

1  using EntidadFinanciera2M6.Data;
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10 using System.Windows.Forms;
11
12 namespace EntidadFinanciera2M6
13 {
14     /// <summary>
15     /// Formulario que muestra todas las transacciones registradas
16     /// </summary>
17
18     3 referencias
19     public partial class Form2 : Form
20     {
21         /// <summary>
22         /// Constructor que inicializa componentes y carga transacciones
23         /// </summary>
24         ///
25         private EntidadFinancieraContext ef = new EntidadFinancieraContext();
26         1 referencia
27         public Form2()
28         {
29             InitializeComponent();
30             Cargar();
31         }
32
33         /// <summary>
34         /// Carga y muestra las transacciones en el DataGridView
35         /// </summary>
36         1 referencia
37         private void Cargar()
38         {
39             dataGridView1.DataSource = ef.Transacciones.ToList();
40         }
41     }
42 }
```


TransferenciaForms

```
EntidadFinanciera2M6
EntidadFinanciera2M6.TransferenciaForms
btnAceptar_Click(object sender, EventArgs e)

1  using EntidadFinanciera2M6.Data;
2  using Microsoft.EntityFrameworkCore;
3  using System;
4  using System.Collections.Generic;
5  using System.ComponentModel;
6  using System.Data;
7  using System.Drawing;
8  using System.Linq;
9  using System.Text;
10 using System.Threading.Tasks;
11 using System.Windows.Forms;
12
13 namespace EntidadFinanciera2M6
14 {
15     /// <summary>
16     /// Formulario que permite ingresar el monto para realizar una transferencia entre cuentas
17     /// </summary>
18
19     3 referencias
20     public partial class TransferenciaForms : Form
21     {
22         /// <summary>
23         /// Monto a transferir ingresado por el usuario
24         /// </summary>
25
26         2 referencias
27         public decimal Monto { get; set; }
28         private int _cuentaOrigenId;
29         private int _cuentaDestinoId;
30         private EntidadFinancieraContext _Ef;
31
32         /// <summary>
33         /// Constructor que recibe las cuentas involucradas y carga sus datos
34         /// </summary>
35
36         1 referencia
37         public TransferenciaForms(int cuentaOrigenId, int cuentaDestinoId)
38         {
39             _cuentaOrigenId = cuentaOrigenId;
40             _cuentaDestinoId = cuentaDestinoId;
41             _Ef = new EntidadFinancieraContext();
42             InitializeComponent();
43             CargarCuentas();
44         }
45     }
46 }
```

```
EntidadFinanciera2M6
EntidadFinanciera2M6.TransferenciaForms
btnAceptar_Click(object sender, EventArgs e)

42     /// <summary>
43     /// Muestra la información de las cuentas en pantalla
44     /// </summary>
45
46     1 referencia
47     private void CargarCuentas()
48     {
49         var cuentaOrigen = _Ef.Cuentas.
50             Include(c => c.Cliente).
51             First(c => c.CuentaId == _cuentaOrigenId);
52
53         var cuentaDestino = _Ef.Cuentas.
54             Include(c => c.Cliente).
55             First(c => c.CuentaId == _cuentaDestinoId);
56
57         lblCuentaOrigen.Text = $"Cuenta Origen: {cuentaOrigen.Cliente.Nombre} - {cuentaOrigen.NumeroCuenta}";
58         lblCuentaDestino.Text = $"Cuenta Destino: {cuentaDestino.Cliente.Nombre} - {cuentaDestino.NumeroCuenta}";
59         lblSaldo.Text = $"Saldo Disponible {cuentaOrigen.Saldo:c}";
60     }
61
62     /// <summary>
63     /// Valida el monto ingresado y finaliza la operación si es válido
64     /// </summary>
65
66     1 referencia
67     private void btnAceptar_Click(object sender, EventArgs e)
68     {
69         if (numMonto.Value > 0)
70         {
71             Monto = numMonto.Value;
72             DialogResult = DialogResult.OK;
73             Close();
74         }
75         else
76             MessageBox.Show("El monto tiene que ser mayor a 0");
77     }
78 }
```

Conclusión

A lo largo del desarrollo de este sistema para la gestión de una entidad financiera, hemos logrado cumplir satisfactoriamente con todos los objetivos propuestos en la planificación inicial.

El sistema cuenta con una estructura bien organizada que incluye múltiples formularios dedicados a distintas funcionalidades específicas, entre las cuales destacan: el registro de clientes (`AgregarClienteForm`), la creación de cuentas bancarias (`AgregarCuentasForm`), la visualización de transacciones (`Form2`), y la gestión de transferencias entre cuentas (`TransferenciaForms`). Cada uno de estos formularios fue diseñado y programado teniendo en cuenta buenas prácticas de programación, principios de encapsulamiento, control de errores y validaciones que garantizan la integridad de los datos introducidos.

Además, se ha realizado un proceso de documentación interna del código a través de comentarios explicativos que detallan la función y propósito de cada parte importante del programa. Esta documentación no solo facilita la comprensión del código por parte de otros desarrolladores, sino que también mejora la mantenibilidad del sistema a futuro.

Desde el punto de vista técnico, implementamos un contexto de base de datos mediante Entity Framework, lo cual nos permitió conectar el sistema a una base de datos relacional de forma robusta, moderna y eficiente. También se hicieron comprobaciones de entrada para evitar errores comunes como campos vacíos o valores inválidos, y se gestionaron correctamente los `DialogResult` de cada formulario para una comunicación clara entre ventanas.

En cuanto a la funcionalidad, el sistema permite:

- Registrar nuevos clientes con datos obligatorios.
- Asignar cuentas a cada cliente con saldos iniciales.
- Consultar todas las transacciones registradas mediante un `DataGridView`.
- Realizar transferencias entre cuentas con validaciones de monto y mensajes de retroalimentación para el usuario.

Durante el proceso de desarrollo no solo pusimos en práctica nuestros conocimientos en programación con C# y Windows Forms, sino que también fortalecimos nuestras habilidades de análisis de requerimientos, diseño de interfaces, trabajo con bases de datos y escritura de código limpio, legible y reutilizable.

En conclusión, el proyecto demuestra que se han cumplido todos los requisitos funcionales y técnicos establecidos. Hemos creado una aplicación completa y operativa que representa un sistema financiero básico pero eficaz, capaz de ser ampliado en el futuro con nuevas funcionalidades como historial detallado de clientes, reportes financieros, autenticación de usuarios, entre otros. Esta experiencia nos ha brindado una visión práctica del desarrollo de software real y reafirma nuestra capacidad para abordar proyectos más complejos en el ámbito profesional.