UNIVERSIDAD VERACRUZANA

Table of Contents

TECNOLOGÍAS PARA LA CONSTRUCCIÓN DE SOFTWARE

ACTIVIDAD: Entrega semanal de avances

Realizado por: Rodrigo Iván Ahumada Rodríguez (S21013886) Marquez Rodríguez Seth (S23014042)

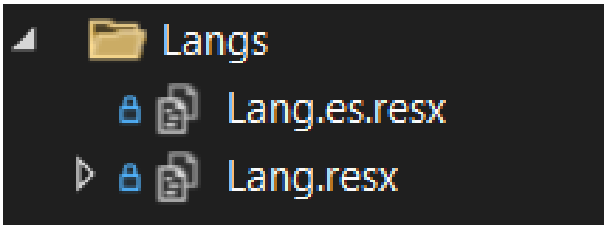
Docente: Perez Arriaga Juan Carlos

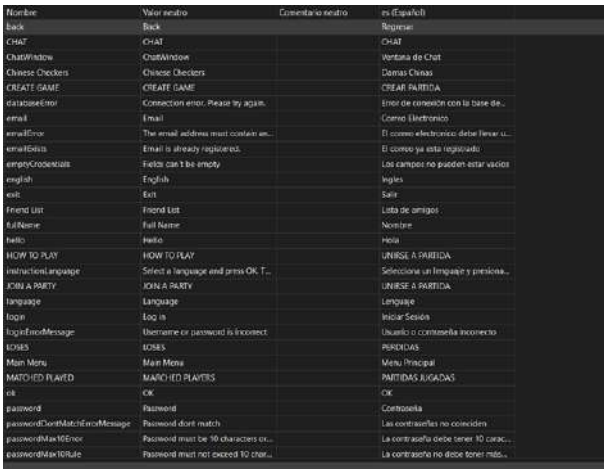
Fecha de entrega: Xalapa, Ver., 21 de Octubre de 2025

Primer reporte de Avance del proyecto final: Juego Damas Chinas

Recursos implementados

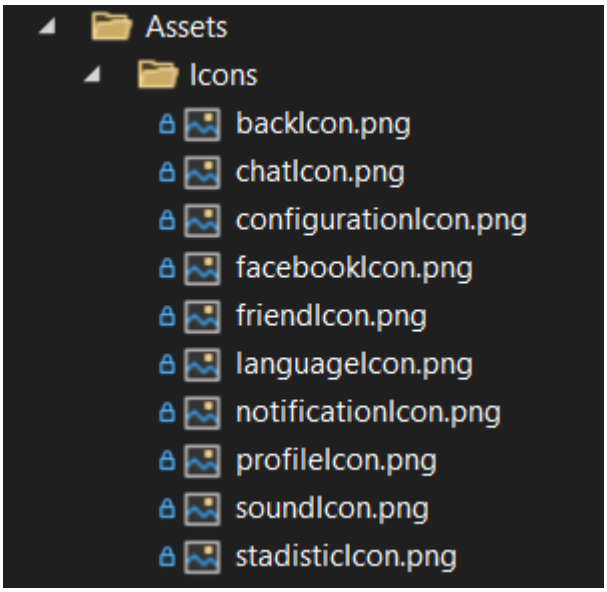
1. **Recurso de idiomas (Lang)** Se implementó internacionalización estática para inglés y español, lo que permite mostrar textos de la interfaz en ambos idiomas.





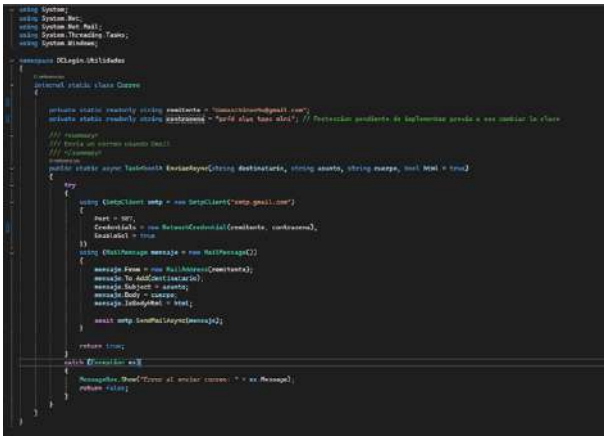
Nombre	Valor neutro	Comentario neutro	es (Español)
Back	Back		Regresar
CHAT	CHAT		CHAT
ChatWindow	ChatWindow		Ventana de Chat
Chinese Checkers	Chinese Checkers		Damas Chinas
CREATE GAME	CREATE GAME		CREAR PARTIDA
databaseError	Connection error. Please try again.		Error de conexión con la base de...
email	Email		Correo Electrónico
emailError	The email address must contain an...		El correo electrónico debe llevar u...
emailExists	Email is already registered.		El correo ya esta registrado
emptyCredentials	Fields can't be empty		Los campos no pueden estar vacios
english	English		Ingles
exit	Exit		Salir
Friend List	Friend List		Lista de amigos
fullName	Full Name		Nombre
hello	Hello		Hola
HOW TO PLAY	HOW TO PLAY		UNIRSE A PARTIDA
instructionLanguage	Select a language and press OK. T...		Selecciona un lenguaje y presiona...
JOIN A PARTY	JOIN A PARTY		UNIRSE A PARTIDA
language	Language		Lenguaje
login	Log in		Iniciar Sesión
loginErrorMessage	Username or password is incorrect		Usuario o contraseña incorrecto
LOSES	LOSES		PERDIDAS
Main Menu	Main Menu		Menu Principal
MATCHED PLAYED	MATCHED PLAYERS		PARTIDAS JUGADAS
ok	OK		OK
password	Password		Contraseña
passwordDontMatchErrorMessage	Password dont match		Las contraseñas no coinciden
passwordMax10Error	Password must be 10 characters or...		La contraseña debe tener 10 carac...
passwordMax10Rule	Password must not exceed 10 char...		La contraseña no debe tener más...

1. **Carpeta Assets/Icons** Contiene los íconos para elementos gráficos de la interfaz, como chat, perfil, notificaciones, sonido, idioma, estadísticas, etc.

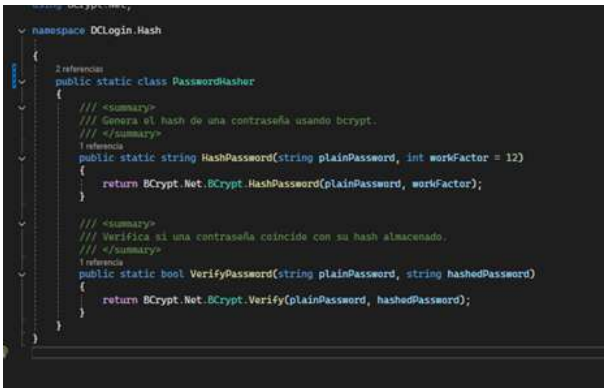


1. **Utilidades** Contiene utilidades generales que serán implementadas a lo largo del código facilitando la reutilización y estandarización.

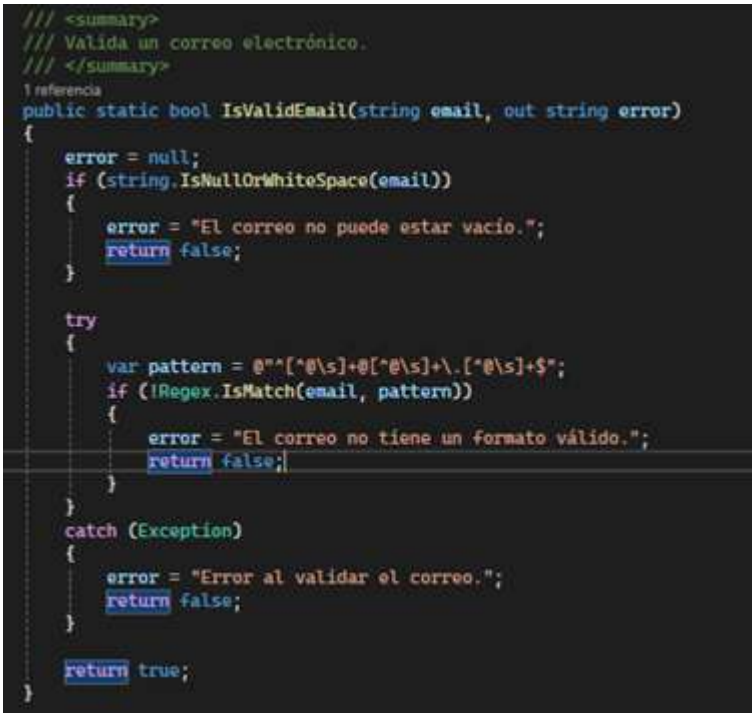
a. **Correo**: módulo para facilitar el envío de correos.



a. **Hasher.cs**: encriptación de contraseñas con BCrypt.



a. **Validator.cs**: utilidades de validación general.
    i. Correo



```
/// <summary>
/// Valida un correo electrónico.
/// </summary>
1 referencia
public static bool IsValidEmail(string email, out string error)
{
    error = null;
    if (string.IsNullOrWhiteSpace(email))
    {
        error = "El correo no puede estar vacío.";
        return false;
    }

    try
    {
        var pattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";
        if (!Regex.IsMatch(email, pattern))
        {
            error = "El correo no tiene un formato válido.";
            return false;
        }
    }
    catch (Exception)
    {
        error = "Error al validar el correo.";
        return false;
    }

    return true;
}
```

i. Password

```
/// <summary>
/// Valida contraseña (máximo 10 caracteres (segura)).
/// </summary>
2 referencias
public static bool IsValidPassword(string password, out string error)
{
    error = null;

    if (string.IsNullOrWhiteSpace(password))
    {
        error = "La contraseña no puede estar vacía.";
        return false;
    }

    if (password.Length > 10)
    {
        error = "La contraseña debe tener como máximo 10 caracteres.";
        return false;
    }

    if (!Regex.IsMatch(password, @"[A-Z]"))
    {
        error = "La contraseña debe contener al menos una letra mayúscula.";
        return false;
    }

    if (!Regex.IsMatch(password, @"\d"))
    {
        error = "La contraseña debe contener al menos un número.";
        return false;
    }

    if (!Regex.IsMatch(password, @"[\W_]")) // \W = no alfanumérico
    {
        error = "La contraseña debe contener al menos un carácter especial.";
        return false;
    }

    return true;
}
```

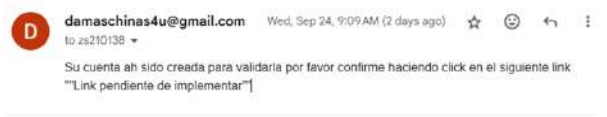
i. Usuario

```
/// <summary>
/// Valida un nombre de usuario (máximo 10 caracteres, máximo 2 espacios, sin caracteres especiales).
/// </summary>
1 referencia
public static bool IsValidUsername(string username, out string error)
{
    error = null;

    if (string.IsNullOrWhiteSpace(username))
    {
        error = "El nombre de usuario no puede estar vacío.";
        return false;
    }

    if (username.Length > 10)
    {
        error = "El nombre de usuario debe tener como máximo 10 caracteres.";
        return false;
    }

    // Solo letras, números y espacios
    if (!Regex.IsMatch(username, @"^[A-Za-z0-9 ]+$"))
    {
        error = "El nombre de usuario solo puede contener letras, números y espacios.";
        return false;
    }

    // Máximo 2 espacios
    int spaceCount = username.Split(' ').Length - 1;
    if (spaceCount > 2)
    {
        error = "El nombre de usuario puede tener como máximo 2 espacios.";
        return false;
    }

    return true;
}
```

Ventanas desarrolladas

1. **Login.xaml** Funcionalidad: Permite el acceso al sistema para usuarios registrados. Características: Validación de correo y contraseña con BCrypt. Internacionalización: Inglés y español. Estado: Funcional.

1. **SignIn.xaml** Funcionalidad: Registro de nuevos usuarios. Características: Validación de correo y contraseñas seguras. Notificación por correo en Gmail. Internacionalización: Inglés y español. Estado: Funcional.



1. **MainMenuRegisteredPlayer.xaml** Menú principal para usuarios registrados (partidas, amigos, perfil, configuración, chat y estadísticas). Internacionalización: Inglés y español.

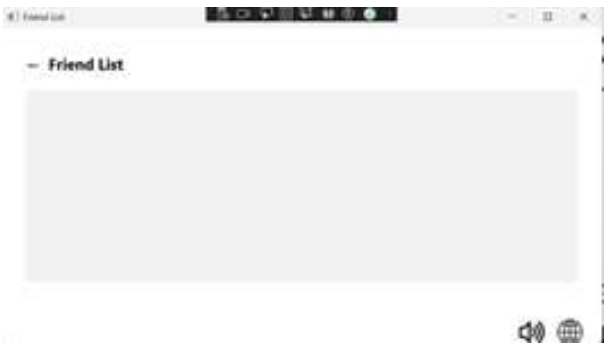
1. **MainMenuGuestPlayer.xaml** Menú principal simplificado para invitados. Acceso limitado a

partidas y opciones básicas.

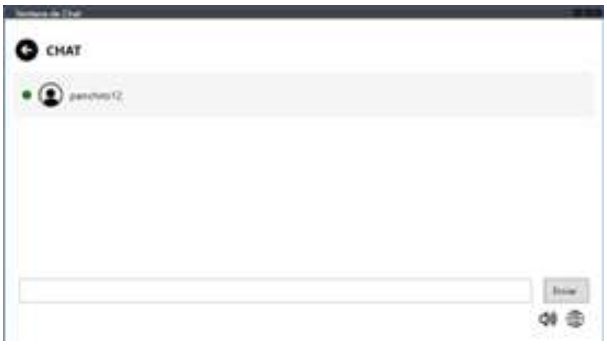
1. **PlayerProfile.xaml** Vista de perfil con estadísticas, logros e información del usuario.

1. **GuestProfile.xaml** Perfil básico para invitados (nombre temporal, avatar por defecto).

1. **FriendsList.xaml** Lista de amigos: agregar, eliminar, estados de conexión, mensajes. Estado: En construcción.



1. **ChatWindow.xaml** Ventana de chat entre jugadores. Estado: En construcción.



[Chat Inglés] | *images/Window_ChatWindow_English.png*

1. **SelectLanguage.xaml** Selección de idioma (inglés/español, carga de diccionarios Lang). Estado: Funcional.

1. **MainWindow.xaml** Ventana base del proyecto en WPF. Punto de arranque de la aplicación.

Mapeo de Base de Datos con Entity Framework

Se creó la base de datos en SQL Server Management Studio y se conectó en Visual Studio usando Entity Framework. Se comprobó la conexión correcta mediante autenticación en SQL Server.

[Entity Mapeo] | *images/xxx.png*

Resumen de contribución por integrante

Integrante 1 – Rodrigo Iván Ahumada Rodríguez - Diseño e implementación de vistas. - Creación e integración de íconos. - Configuración de la conexión a BD. - Internacionalización (50%). Contribución estimada: 50%.

Integrante 2 – Marquez Rodríguez Seth - Desarrollo de la navegabilidad entre ventanas. - Implementación de la BD en SQL Server. - Implementación de utilidades: validación, encriptación, validadores. - Internacionalización (50%). Contribución estimada: 50%.

Nota: El equipo considera que ambas contribuciones son complementarias (uno enfocado en capa visual y BD, el otro en lógica de validación y soporte multilenguaje).

Uso de Inteligencia Artificial (IA)

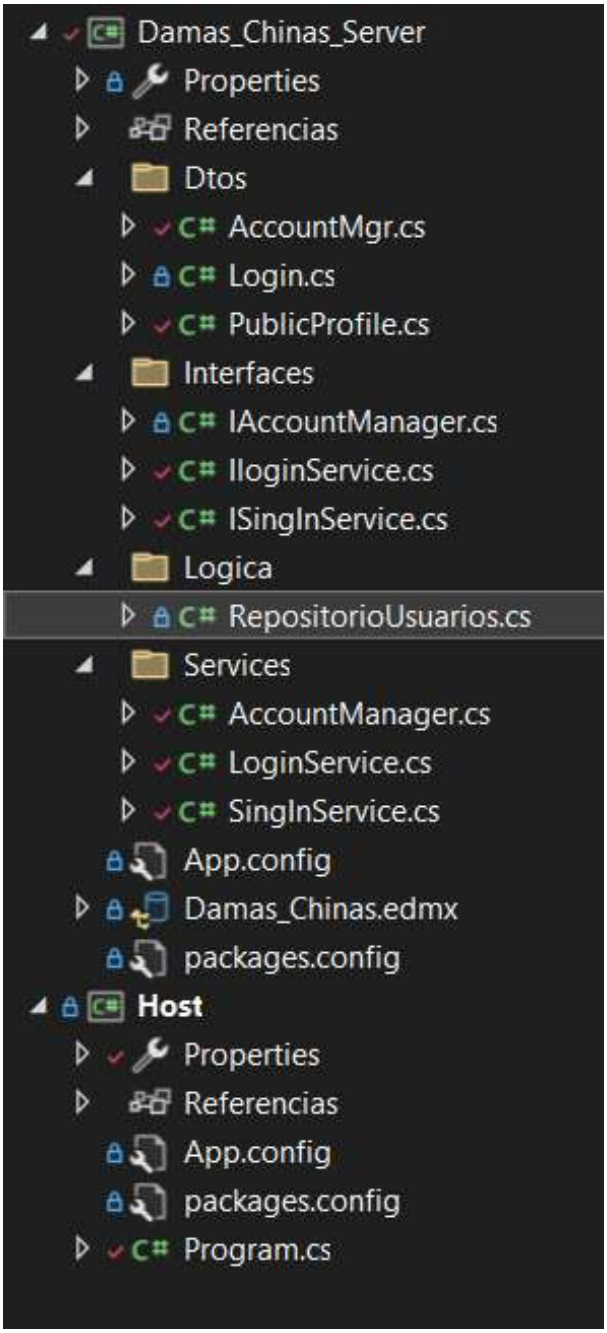
El equipo definió reglas para un uso responsable:

- Permitido: consultar sobre tecnologías y ventajas.
- Permitido: ejemplos generales de implementación.
- No permitido: pedir desarrollo completo de módulos.
- Permitido: compartir código propio para revisión y comentarios.
- Prohibido: usar código generado por IA que no se entienda.

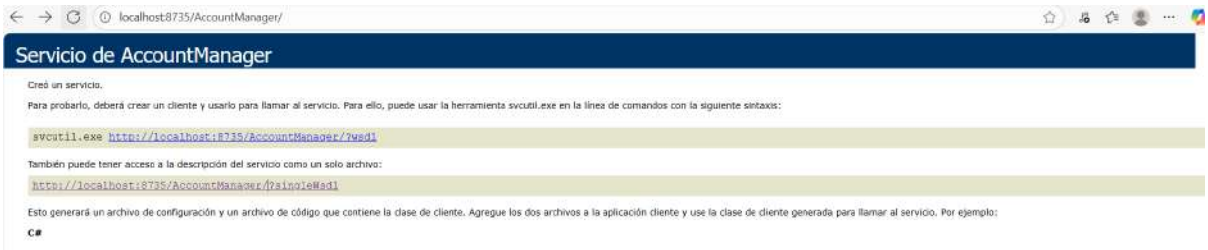
Segundo reporte de Avance del proyecto final: Juego Damas Chinas

Server

```
? LoginService: http://localhost:8739/LoginService/
? SignInService: http://localhost:8736/SignInService/
? AccountManager: http://localhost:8735/AccountManager/
```



AccountManager.cs



DataContracts

- **PublicProfile**
- Representa la información pública de un usuario.
- Campos:

- `Username` (string) – nombre de usuario del perfil.
- `Nombre` (string) – nombre propio del usuario.
- `LastName` (string) – apellido del usuario.
- `Correo` (string) – correo del usuario.
- `Telefono` (string) – teléfono registrado del usuario.
- **UsuarioInfo**
- Representa la información de un usuario dentro de una operación.
- Campos:
- `IdUsuario` (int) – identificador del usuario.
- `Username` (string) – nombre de usuario.
- `Correo` (string) – correo del usuario.
- `NombreCompleto` (string) – concatenación de nombre y apellido.
- **ResultadoOperacion**
- Representa el resultado de una operación de modificación de datos.
- Campos:
- `Exito` (bool) – indica si la operación fue exitosa.
- `Mensaje` (string) – descripción del resultado.
- `Usuario` (UsuarioInfo) – información del usuario afectado (opcional, puede ser null).

Servicios

- **IAccountManager** (ServiceContract)
- Define las operaciones expuestas por el servicio WCF **AccountManager**.
- Operaciones (OperationContract):
- `PublicProfile ObtenerPerfilPublico(int idUsuario)`
- `ResultadoOperacion CambiarUsername(int idUsuario, string nuevoUsername)`
- `ResultadoOperacion CambiarPassword(int idUsuario, string nuevaPassword)`

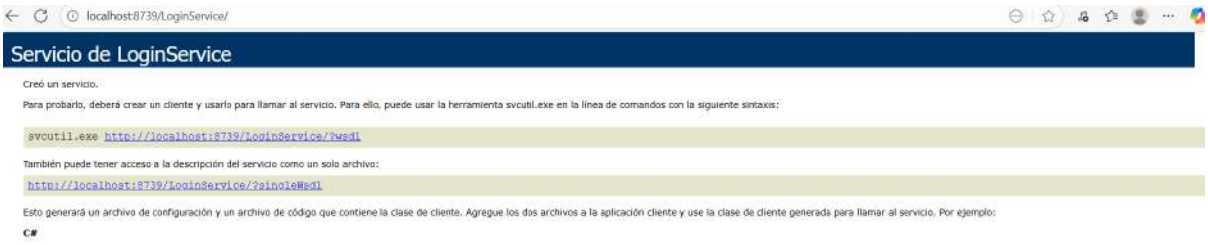
Operaciones / Funcionalidad

1. **ObtenerPerfilPublico(int idUsuario)**
    - Función: Devuelve la información pública de un usuario. Internamente delega la obtención a **RepositorioUsuarios.ObtenerPerfilPublico**.
    - Retorna: **PublicProfile** (DataContract)
    - Retorna **null** si el usuario no existe.
2. **CambiarUsername(int idUsuario, string nuevoUsername)**
    - Función: Modifica el nombre de usuario de un usuario. Delegado a **RepositorioUsuarios.CambiarUsername**, que valida el username.

- Retorna: **ResultadoOperacion** (DataContract)
- `Exito`, `Mensaje`, `Usuario` (actualmente null)

3. **CambiarPassword(int idUsuario, string nuevaPassword)**
   - Función: Modifica la contraseña de un usuario. Delegado a **RepositorioUsuarios.CambiarPassword**, que valida la contraseña.
   - Retorna: **ResultadoOperacion** (DataContract)
   - `Exito`, `Mensaje`, `Usuario` (actualmente null)

LoginService.cs



DataContracts

- **LoginResult**
- Representa el resultado de la validación de login de un usuario.
- Campos:
- `IdUsuario` (int) – identificador del usuario.
- `Username` (string) – nombre de usuario del perfil.
- `Success` (bool) – indica si la validación fue exitosa.

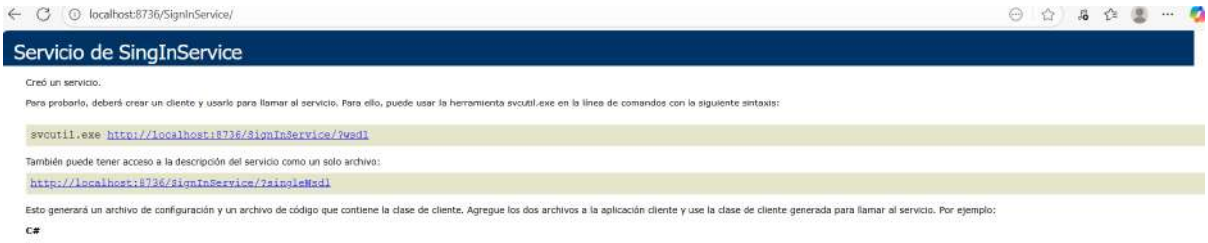
Servicios

- **ILoginService** (ServiceContract)
- Define las operaciones expuestas por el servicio WCF **LoginService**.
- Operaciones (OperationContract):
- `LoginResult ValidarLogin(string usuarioInput, string password)`

Operaciones / Funcionalidad

1. **ValidarLogin(string usuarioInput, string password)**
   - Función: Valida las credenciales de un usuario. Internamente delega la operación a **RepositorioUsuarios.ObtenerLoginResult**.
   - Retorna: **LoginResult** (DataContract)
   - `IdUsuario`, `Username`, `Success`
   - Validación mínima: Se asegura de que los parámetros no estén vacíos dentro del repositorio.

SingInService.cs



DataContracts

- **UsuarioInfo**
- Representa la información de un usuario dentro de una operación.
- Campos:
- `IdUsuario` (int) – identificador del usuario.
- `Username` (string) – nombre de usuario.
- `Correo` (string) – correo del usuario.
- `NombreCompleto` (string) – concatenación de nombre y apellido.
- **ResultadoOperacion**
- Representa el resultado de una operación sobre datos de usuario.
- Campos:
- `Exito` (bool) – indica si la operación fue exitosa.
- `Mensaje` (string) – descripción del resultado o error.
- `Usuario` (UsuarioInfo) – información del usuario afectado (opcional, puede ser null).

Servicios

- **ISingInService** (ServiceContract)
- Define las operaciones expuestas por el servicio WCF **SingInService**.
- Operaciones (OperationContract):
- `ResultadoOperacion CrearUsuario(string nombre, string apellido, string correo, string password, string username)`

Operaciones / Funcionalidad

1. **CrearUsuario(string nombre, string apellido, string correo, string password, string username)**
    - Función: Crea un nuevo usuario junto con su perfil asociado. Internamente delega la operación a **RepositorioUsuarios.CrearUsuario**, que realiza todas las validaciones de nombre, apellido, correo, username y contraseña.
    - Retorna: **ResultadoOperacion** (DataContract)

- `Exito` – true si se creó correctamente.
- `Mensaje` – mensaje de éxito o error.
- `Usuario` – **UsuarioInfo** con los datos del usuario creado.
- Envío de correo de bienvenida: Opcionalmente envía un email en segundo plano tras la creación del usuario.
- Validación: Todas las validaciones se realizan en **RepositorioUsuarios**, no en el servicio.

Logica

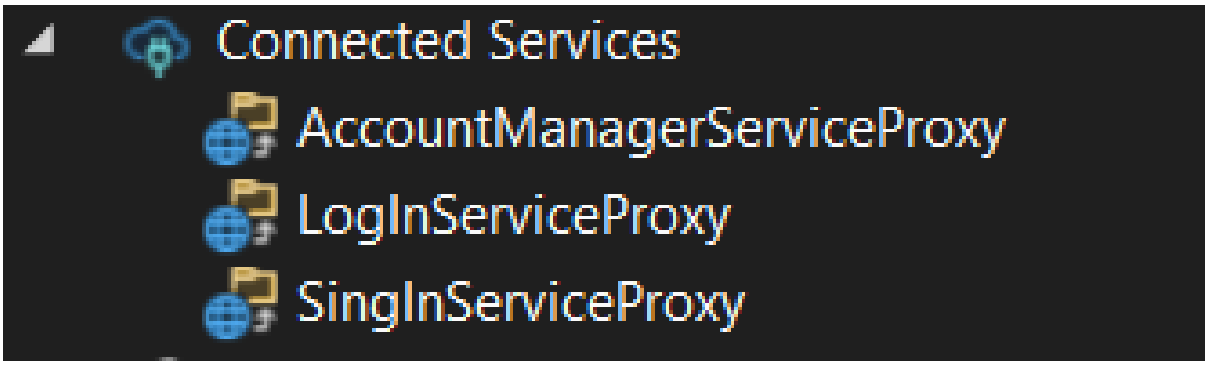
RepositorioUsuarios.cs

1. **CrearUsuario(string nombre, string apellido, string correo, string password, string username)**
   - Función: Crea un nuevo usuario junto con su perfil asociado en la base de datos. Antes de guardar, valida los datos usando la clase **Validator**:
   - Nombre y apellido → **Validator.ValidarNombre**
   - Correo → **Validator.ValidarCorreo**
   - Username → **Validator.ValidarUsername**
   - Contraseña → **Validator.ValidarPassword**
   - Retorna: usuarios (entidad creada con su perfil agregado).
   - Excepciones: Lanza excepción si ya existe el correo o el username, o si algún dato no cumple las reglas de validación.
2. **ObtenerLoginResult(string usuarioInput, string password)**
   - Función: Valida las credenciales de un usuario y retorna información básica para login. Realiza validación mínima de que los parámetros no estén vacíos.
   - Retorna: **LoginResult**
   - `IdUsuario`, `Username`, `Success`
   - Excepciones: Lanza excepción si **usuarioInput** o **password** están vacíos.
3. **ObtenerPerfilPublico(int idUsuario)**
   - Función: Obtiene la información pública de un usuario a partir de su id.
   - Retorna: **PublicProfile** (datos como username, nombre, apellido, correo y teléfono).
   - Retorna **null** si el usuario no existe.
4. **CambiarUsername(int idUsuario, string nuevoUsername)**
   - Función: Actualiza el nombre de usuario de un perfil. Antes de modificarlo, valida el username usando:
   - **Validator.ValidarUsername**
   - Retorna: **bool** – `true` si la operación fue exitosa.
   - Excepciones: Lanza excepción si el username ya existe, el perfil no se encuentra, o el

username no cumple las reglas de validación.

5. **CambiarPassword(int idUsuario, string nuevaPassword)**
    - Función: Actualiza la contraseña de un usuario. Antes de modificarla, valida la contraseña usando:
    - **Validator.ValidarPassword**
    - Retorna: **bool** – `true` si la operación fue exitosa.
    - Excepciones: Lanza excepción si el usuario no existe o la contraseña no cumple las reglas de validación.

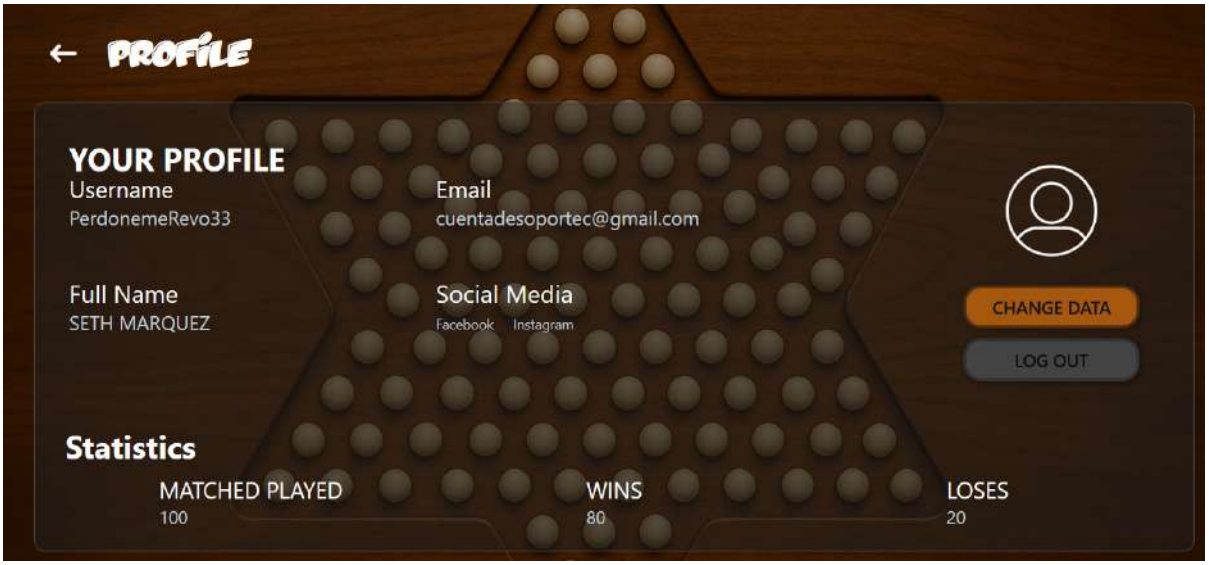
IMPLEMENTACION



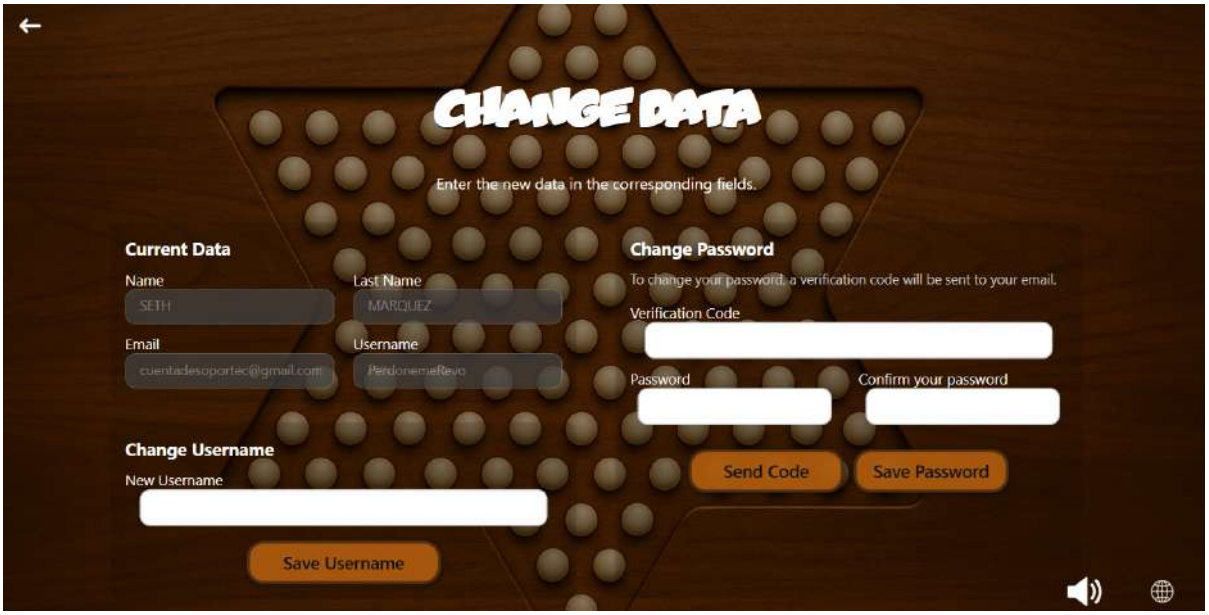
SingIn

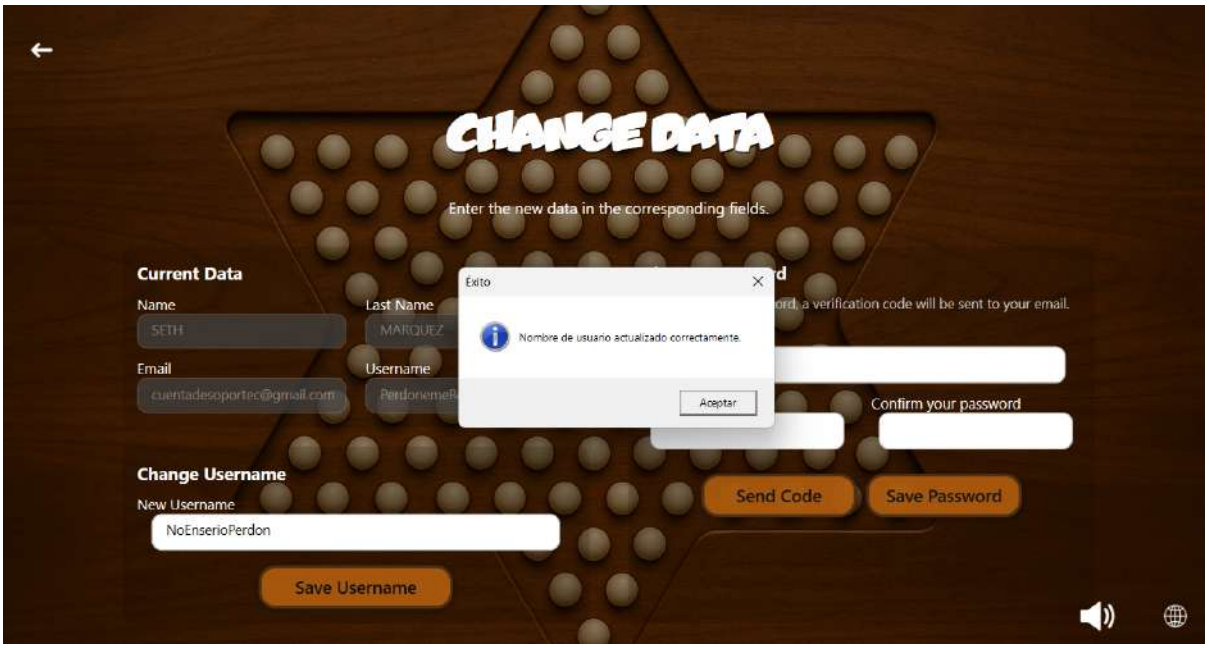
MenuRegisteredPlayer

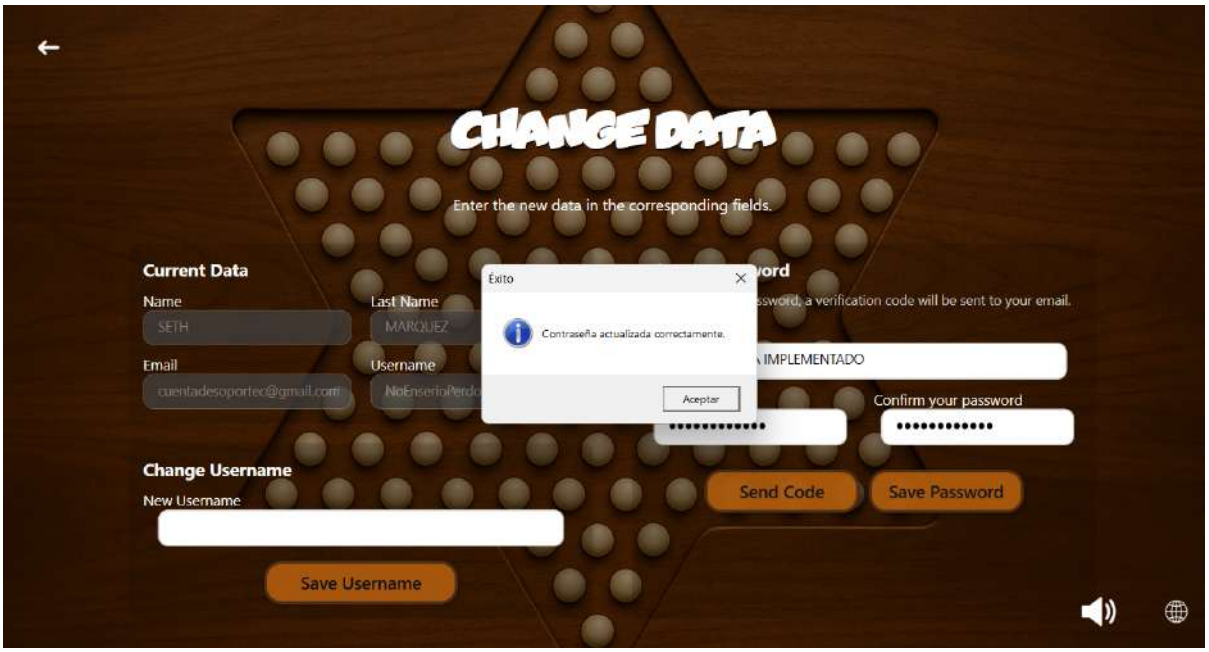
ProfilePlayer



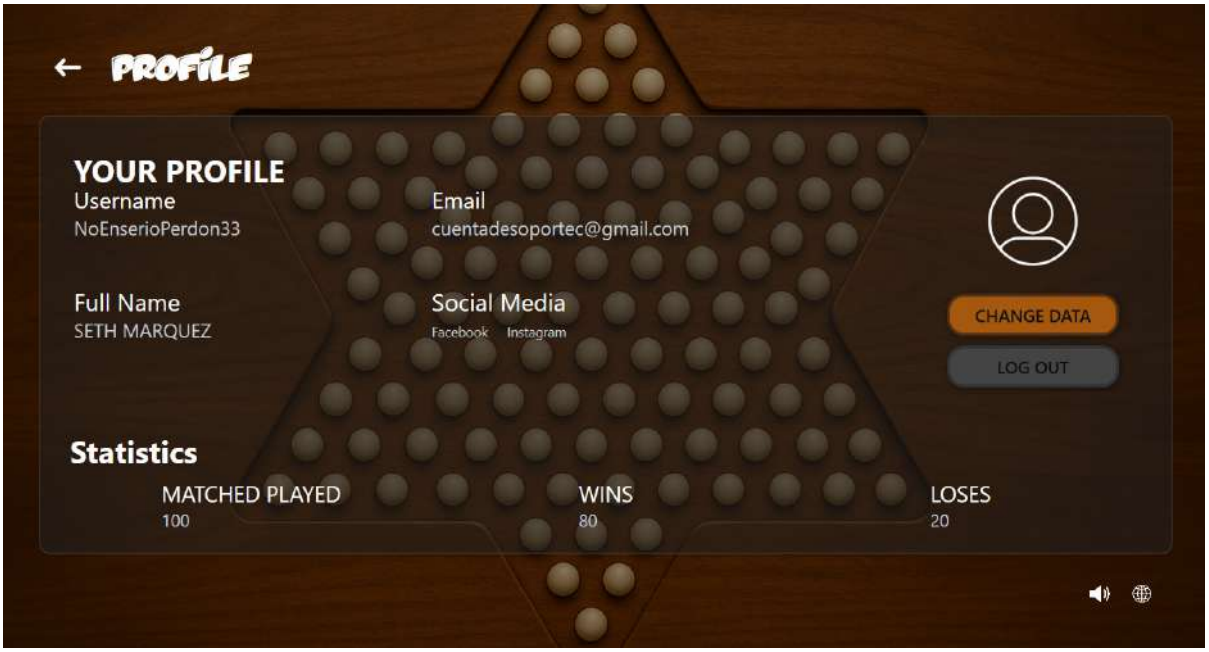
ChangeData


CHANGE DATA
Enter the new data in the corresponding fields.
Current Data
Name
SETH
Last Name
MARQUEZ
Email
cuentadesoportec@gmail.com
Username
PerdonemeRevo
Change Password
To change your password, a verification code will be sent to your email.
Verification Code
Password
Confirm your password
Send Code
Save Password
Change Username
New Username
Save Username


CHANGE DATA
Enter the new data in the corresponding fields.
Current Data
Name
SETH
Last Name
MARQUEZ
Email
cuentadesoportec@gmail.com
Username
Éxito
Nombre de usuario actualizado correctamente.
Aceptar
Confirm your password
Change Username
New Username
NoEnserioPerdon
Send Code
Save Password
Save Username


CHANGE DATA
Enter the new data in the corresponding fields.
Current Data
Name
SETH
Last Name
MARQUEZ
Email
cuentadesoportec@gmail.com
Username
Éxito
Contraseña actualizada correctamente.
Aceptar
IMPLEMENTADO
Confirm your password
Change Username
New Username
Send Code
Save Password
Save Username

Evidencia posterior al cambio



DOCUMENTACION

Elaboración de un documento de **AsciiDoc** para facilitar el control y registro de los cambios realizados con cada entrega.

Avance del Proyecto Final: Juego Damas Chinas (Entrega 2)

SETH

- **Creación del server:** 100%

- **Implementación de servicios en el cliente:** 100%
- **Creación de documento ASCII-DOC:** 60%
- **Conexión de Cliente con Host:** 100%

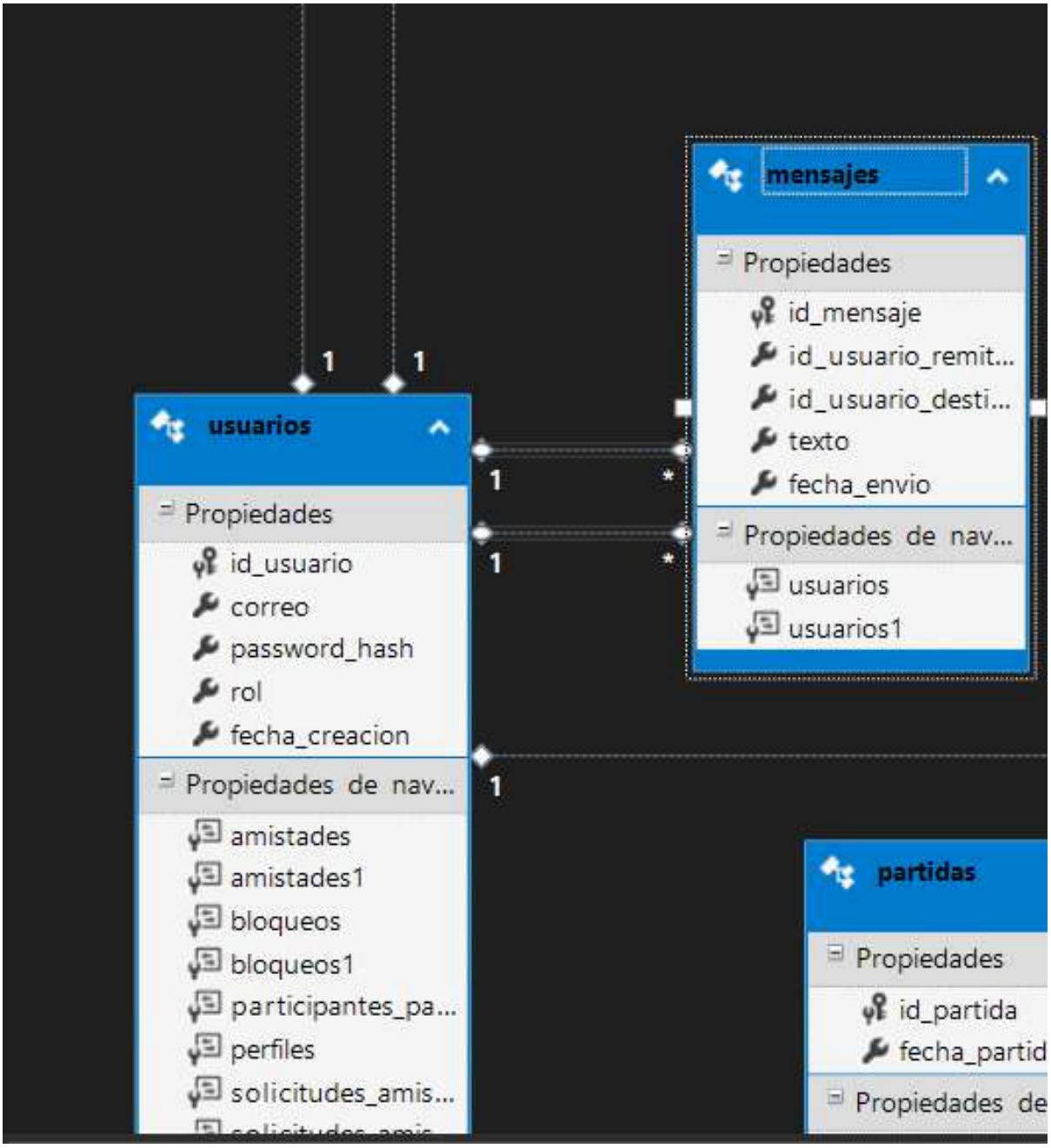
IVAN

- **GUI:** 100%

Tercer reporte de entrega

Modificaciones a la BD

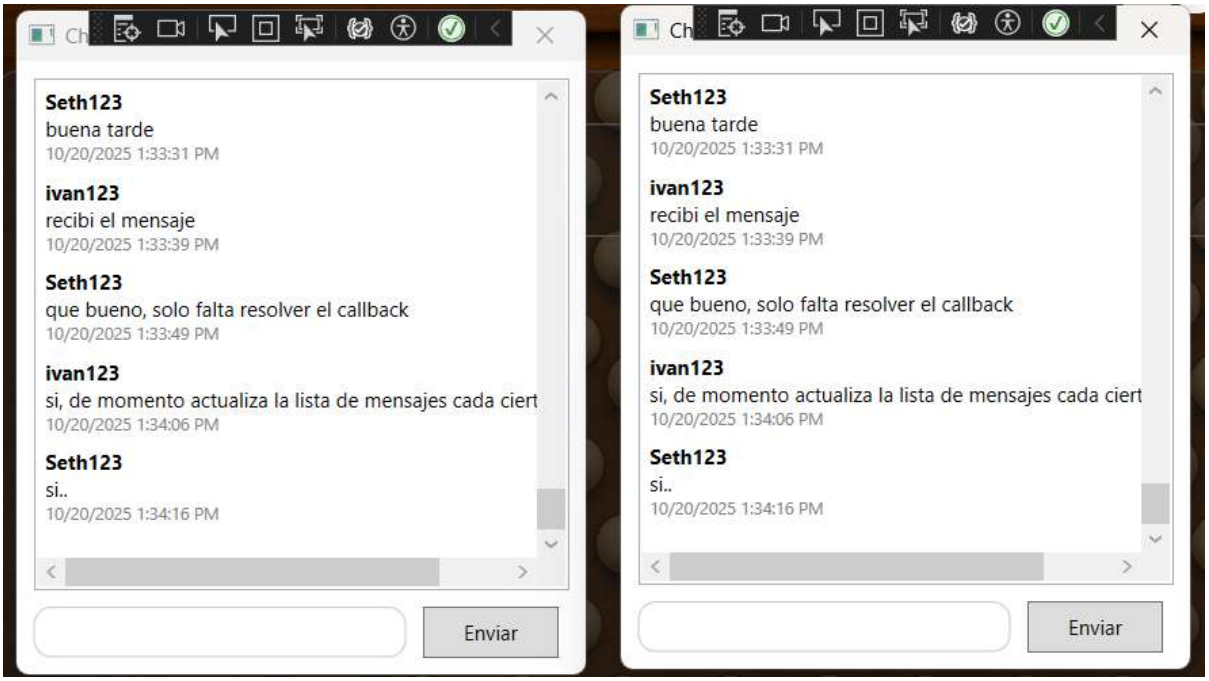
En busqueda de mantener la persistencia de los mensajes existentes en un chat de amigos se implemento la siguiente tabla en la base de datos, con la finalidad de no saturar la memoria del servidor para acceder a las platicas.



Cliente

CHAT

Funcionalidad: Guardar Mensajes entre amigos. Características: Muestra de forma dinamica la recepcion de mensajes manteninedo chats individuales entre amigos. Internacionalización: Inglés y español. Estado: En proceso, la mensajeria funciona pero el callback que actualiza al llegar un nuevo mensaje aun no esta del todo funcional



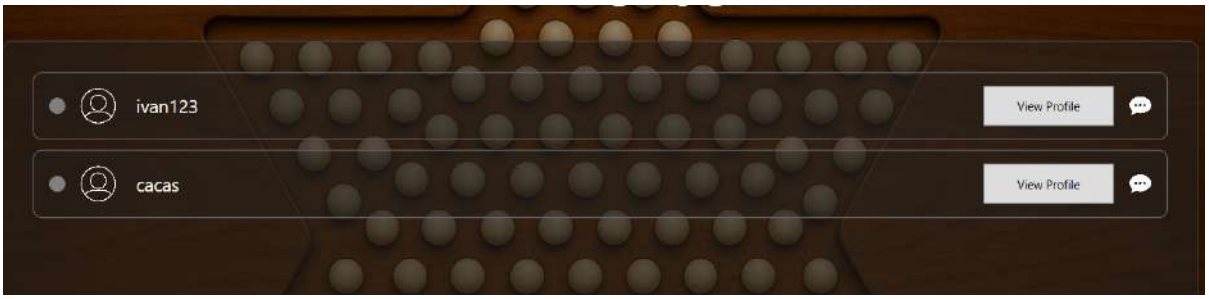
Fucionalidad

```
<ListBox x:Name="MessagesList" ItemsSource="{Binding Messages}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Margin="0,2">
                <TextBlock Text="{Binding UsernameDestino}" FontWeight="Bold"/>
                <TextBlock Text="{Binding Texto}" TextWrapping="Wrap"/>
                <TextBlock Text="{Binding FechaEnvio}" FontSize="10" Foreground="Gray"/>
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

AL iniciar la ventana de chat, se recurre a la interfaz **MesageService** misma que se encarga de consultar los mensajes existentes entre 2 usuarios y agregarlos a una coleccion de mensajes misma que sera mostrada automaticamente por la page.

```
1 referencia
public ChatWindow(int miId, string friendUsername)
{
    InitializeComponent();

    _miIdUsuario = miId;
    _friendUsername = friendUsername;
    _myUsername = ObtenerMiUsername();

    DataContext = this;

    var callback = new MensajeriaCallback(this);
    var context = new InstanceContext(callback);

    var binding = new NetTcpBinding
    {
        Security = { Mode = SecurityMode.None },
        ReceiveTimeout = TimeSpan.MaxValue
    };

    var endpoint = new EndpointAddress("net.tcp://localhost:8755/MensajeriaService/");
    var factory = new DuplexChannelFactory<IMensajeriaService>(context, binding, endpoint);
    _client = factory.CreateChannel();


    _client.RegistrarCliente(_myUsername);


    CargarHistorial();


    _refreshTimer = new DispatcherTimer();
    _refreshTimer.Interval = TimeSpan.FromSeconds(5);
    _refreshTimer.Tick += (s, e) => CargarHistorial();
    _refreshTimer.Start();
}
```

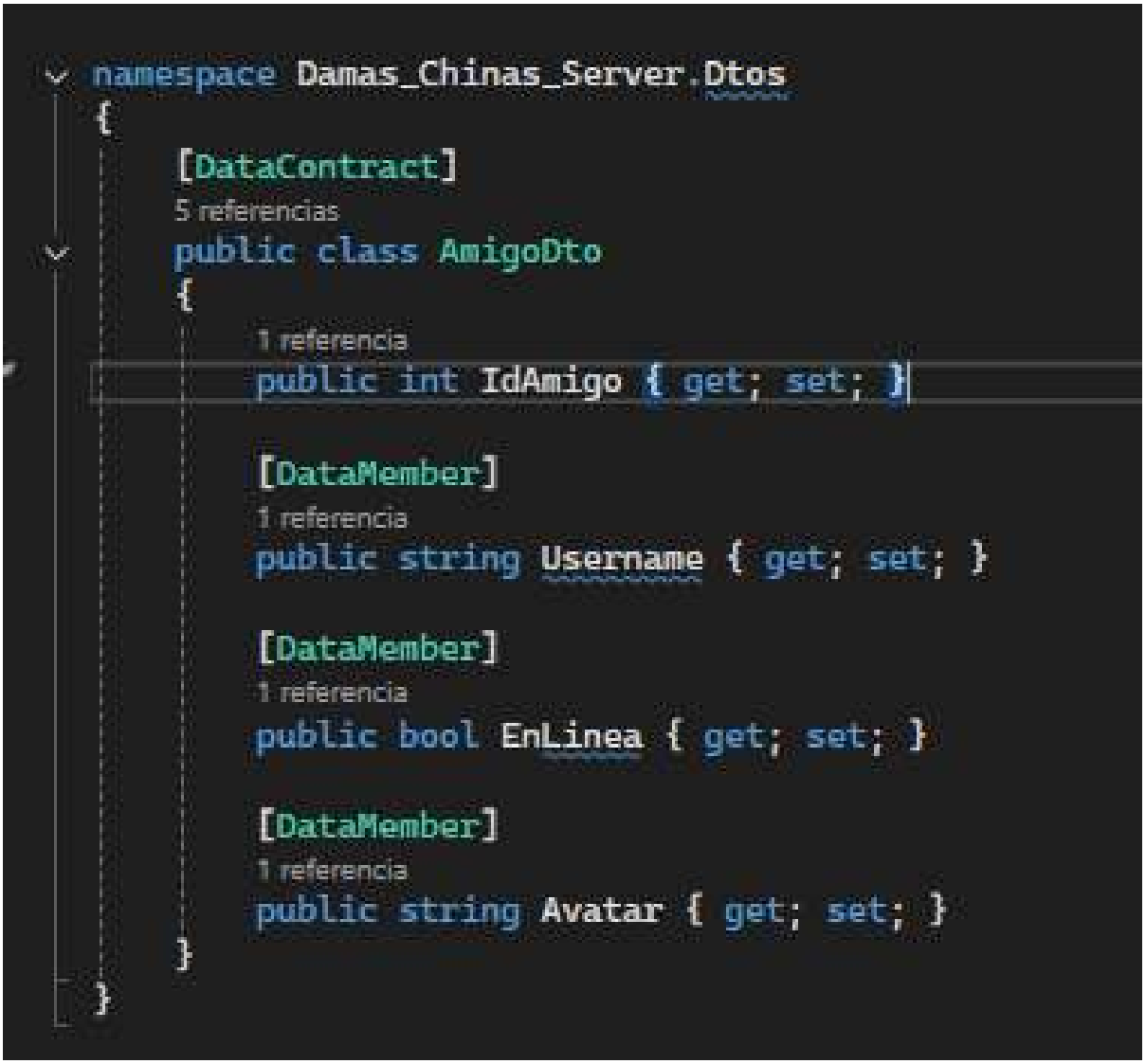
Friends



Funcionalidad:

- Mostrar la lista de amigos
- Permite la apertura de un chat con amigos
- Nos permite consultar el perfil publico de nuestros amigos

Características: Al entrar en la ventana se recupera la lista de amigos del usuario y los despliega por medio de una lista de amigos provista por el servidor



```
namespace Damas_Chinas_Server.Dtos
{
    [DataContract]
    5 referencias
    public class AmigoDto
    {
        1 referencia
        public int IdAmigo { get; set; }

        [DataMember]
        1 referencia
        public string Username { get; set; }

        [DataMember]
        1 referencia
        public bool EnLinea { get; set; }

        [DataMember]
        1 referencia
        public string Avatar { get; set; }
    }
}
```

Y lo muestra por medio de un ItemControl Para mantener la estetica de nuestras ventanas y permiten la creacion de botones dinamicos para cada uno de los amigos.

```xaml
<!-- ItemsControl +-->
<ItemsControl x:Name="FriendsList" ItemsSource="{Binding AmigosList}" Margin="10">
    <ItemsControl.ItemTemplate>
        <DataTemplate>
            <Border Background="#44222222"
                    BorderBrush="#666"
                    BorderThickness="1"
                    CornerRadius="8"
                    Padding="10"
                    Margin="0,0,0,10">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="*"/>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="Auto"/>
                    </Grid.ColumnDefinitions>

                    <!-- Falta Implementar -->
                    <Ellipse Width="16" Height="16"
                             Fill="Gray"
                             VerticalAlignment="Center"
                             Margin="5,0,10,0"
                             IsEnabled="False"/>

                    <!-- Avatar -->
                    <Image Grid.Column="1"
                           Source="{Binding Avatar}"
                           Width="45" Height="45"
                           Stretch="Uniform"/>

                    <!-- Username -->
                    <TextBlock Grid.Column="2"
                               Text="{Binding Username}"
                               Foreground="White"
                               FontSize="18"
                               VerticalAlignment="Center"
                               Margin="15,0,0,0"/>

                    <!-- Ver perfil (Boton) -->
                    <Button Grid.Column="3"
                            Content="View Profile"
                            Width="130" Height="40"
                            Margin="10,0,10,0"
                            Click="OnViewProfileClick"/>

                    <!-- Chat (Boton) -->
                    <Button Grid.Column="4"
                            Background="Transparent"
                            BorderThickness="0"
                            Cursor="Hand"
                            Click="OnChatClick">
                        <Image Source="pack://application:,,,/DamasChinas_Client.UI;component/Assets/Icons/chatIcon.png"
                               Width="30" Height="30"
                               Stretch="Uniform"/>
                    </Button>
```

Servidor

Evidencia Servicios Corriendo

```
? LoginService: http://localhost:8739/LoginService/
? SignInService: http://localhost:8736/SignInService/
? AccountManager: http://localhost:8735/AccountManager/
? SaludoService (NetTcp): net.tcp://localhost:8755/MensajeriaService/
? AmistadService: http://localhost:8741/AmistadService/
```

AmistadService

Funcionalidades y estado.

Este servicio es el encargado de todo lo relacionado a la gestion de amigos.

- Mostrar Lista de Amigos (Funcional)
- Agregar amigo (pendiente de implementacion)
- Eliminar amigo (pendiente de implementacion)
- Bloquear usuario (pendiente de implementacion)

```
using Damas_Chinas_Server.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace Damas_Chinas_Server
{
    1 referencia
    public class AmistadService : IAmistadService
    {
        private readonly RepositorioAmistades repo = new RepositorioAmistades();

        1 referencia
        public List<AmigoDto> ObtenerAmigos(int idUsuario)
        {
            return repo.ObtenerAmigos(idUsuario);
        }
    }
}
```

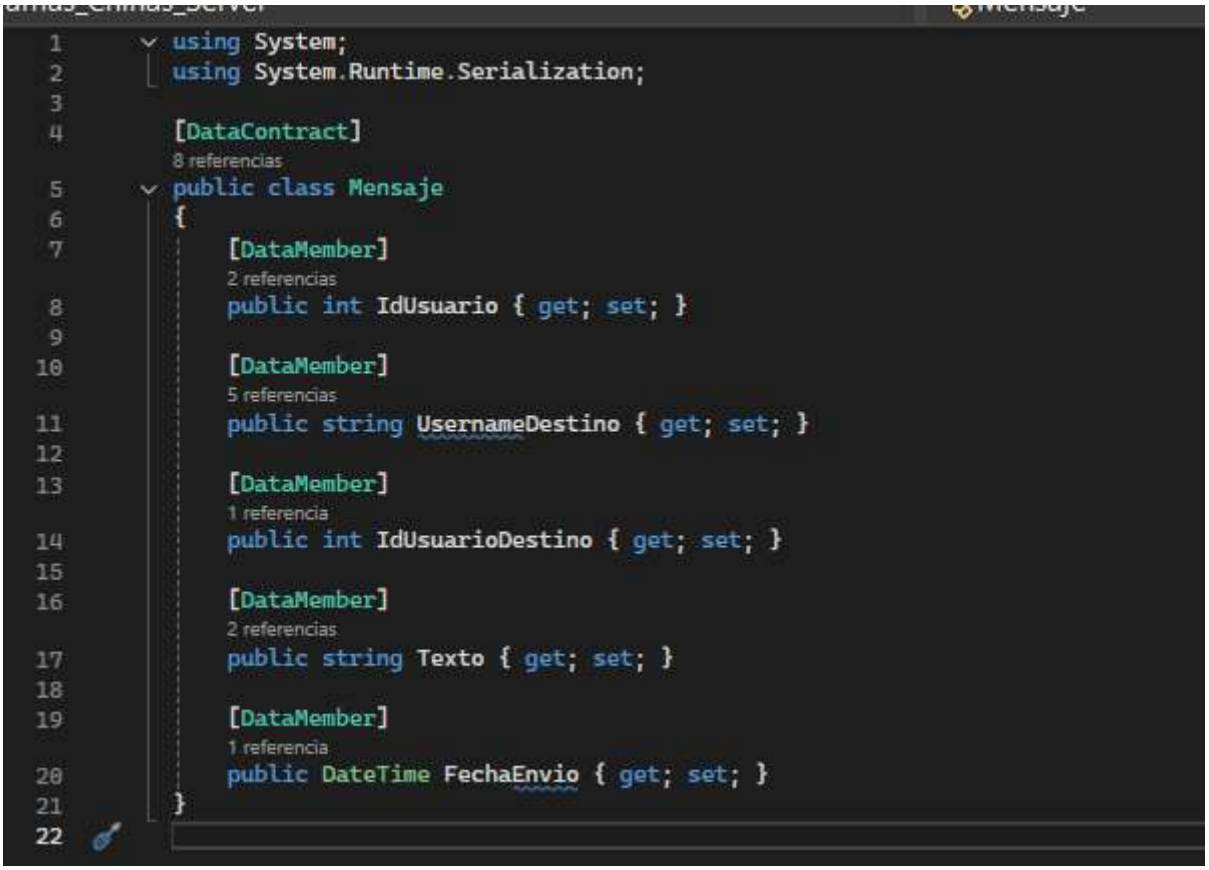
```
using Damas_Chinas_Server.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace Damas_Chinas_Server
{
    [ServiceContract]
    2 referencias
    public interface IAmistadService
    {
        [OperationContract]
        1 referencia
        List<AmigoDto> ObtenerAmigos(int idUsuario);
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;


namespace Damas_Chinas_Server.Dtos
{
    [DataContract]
    5 referencias
    public class AmigoDto
    {
        1 referencia
        public int IdAmigo { get; set; }

        [DataMember]
        1 referencia
        public string Username { get; set; }

        [DataMember]
        1 referencia
        public bool EnLinea { get; set; }

        [DataMember]
        1 referencia
        public string Avatar { get; set; }
    }
}
```

MessageService

Este servicio es el encargado de todo lo relacionado a la entrega de mensajes entre amigos para el chat

DTO



```
using System;
using System.Runtime.Serialization;

[DataContract]
8 referencias
public class Mensaje
{
    [DataMember]
    2 referencias
    public int IdUsuario { get; set; }

    [DataMember]
    5 referencias
    public string UsernameDestino { get; set; }

    [DataMember]
    1 referencia
    public int IdUsuarioDestino { get; set; }

    [DataMember]
    2 referencias
    public string Texto { get; set; }

    [DataMember]
    1 referencia
    public DateTime FechaEnvio { get; set; }
}
```

ServiceContract

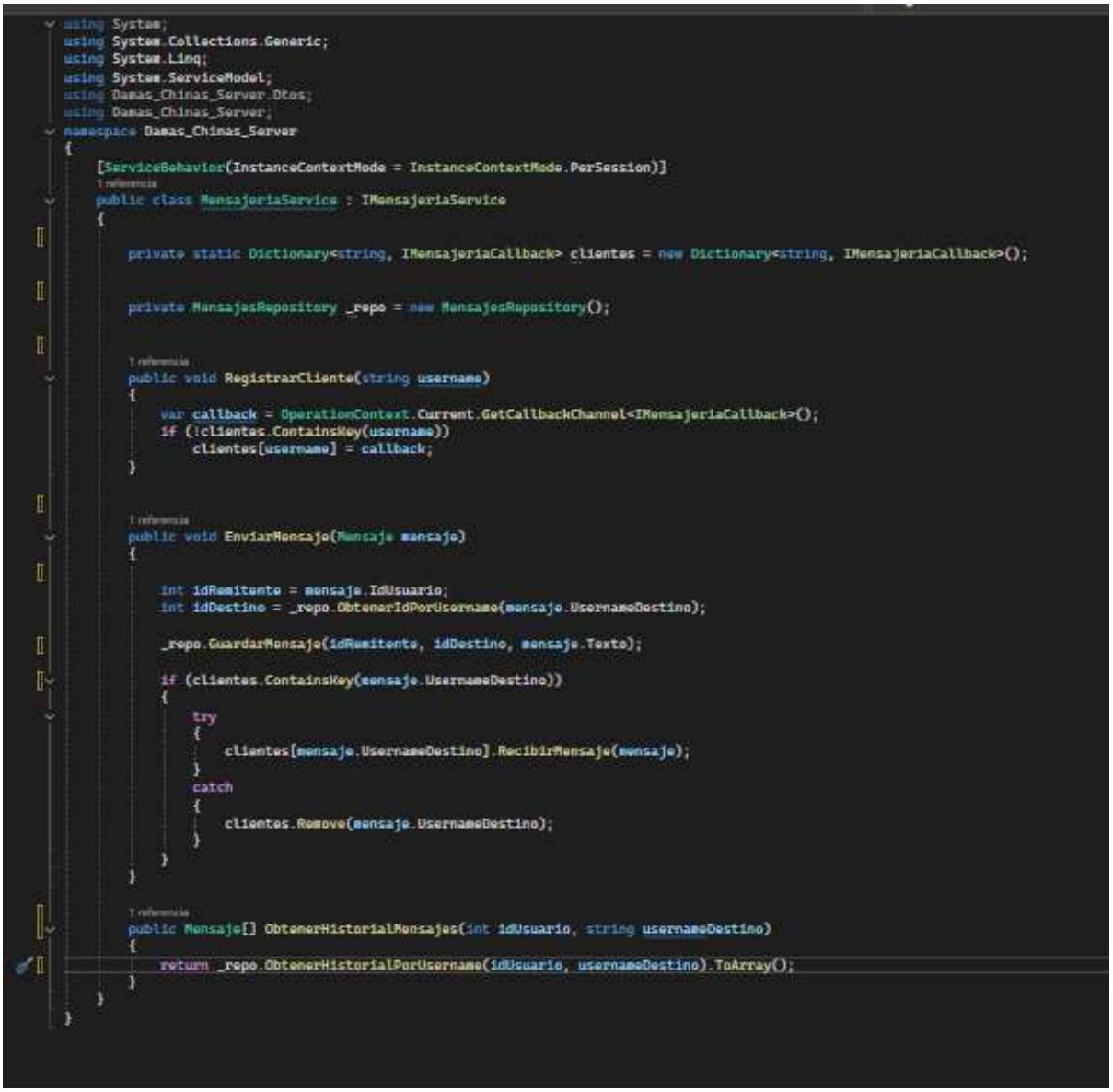
```cs
using System;
using System.Runtime.Serialization;

[DataContract]
8 referencias
public class Mensaje
{
    [DataMember]
    2 referencias
    public int IdUsuario { get; set; }

    [DataMember]
    5 referencias
    public string UsernameDestino { get; set; }

    [DataMember]
    1 referencia
    public int IdUsuarioDestino { get; set; }

    [DataMember]
    2 referencias
    public string Texto { get; set; }

    [DataMember]
    1 referencia
    public DateTime FechaEnvio { get; set; }
}
```

Service

```cs
namespace Damas_Chinas_Server
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
    1 referencia
    public class MensajeriaService : IMensajeriaService
    {

        private static Dictionary<string, IMensajeriaCallback> clientes = new Dictionary<string, IMensajeriaCallback>();


        private MensajesRepository _repo = new MensajesRepository();


        1 referencia
        public void RegistrarCliente(string username)
        {
            var callback = OperationContext.Current.GetCallbackChannel<IMensajeriaCallback>();
            if (!clientes.ContainsKey(username))
                clientes[username] = callback;
        }


        1 referencia
        public void EnviarMensaje(Mensaje mensaje)
        {

            int idRemitente = mensaje.IdUsuario;
            int idDestino = _repo.ObtenerIdPorUsername(mensaje.UsernameDestino);

            _repo.GuardarMensaje(idRemitente, idDestino, mensaje.Texto);

            if (clientes.ContainsKey(mensaje.UsernameDestino))
            {
                try
                {
                    clientes[mensaje.UsernameDestino].RecibirMensaje(mensaje);
                }
                catch
                {
                    clientes.Remove(mensaje.UsernameDestino);
                }
            }
        }

        1 referencia
        public Mensaje[] ObtenerHistorialMensajes(int idUsuario, string usernameDestino)
        {
            return _repo.ObtenerHistorialPorUsername(idUsuario, usernameDestino).ToArray();
        }
    }
}
```

UtilidadesImplementadas

encargadas de mediar la logica entre el servidor y la implementacion de entity framework



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using Damas_Chinas_Server.Dtos;
using Damas_Chinas_Server;
namespace Damas_Chinas_Server
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
    1 referencia
    public class MensajeriaService : IMensajeriaService
    {

        private static Dictionary<string, IMensajeriaCallback> clientes = new Dictionary<string, IMensajeriaCallback>();


        private MensajesRepository _repo = new MensajesRepository();


        1 referencia
        public void RegistrarCliente(string username)
        {
            var callback = OperationContext.Current.GetCallbackChannel<IMensajeriaCallback>();
            if (!clientes.ContainsKey(username))
                clientes[username] = callback;
        }


        1 referencia
        public void EnviarMensaje(Mensaje mensaje)
        {

            int idRemitente = mensaje.IdUsuario;
            int idDestino = _repo.ObtenerIdPorUsername(mensaje.UsernameDestino);

            _repo.GuardarMensaje(idRemitente, idDestino, mensaje.Texto);

            if (clientes.ContainsKey(mensaje.UsernameDestino))
            {
                try
                {
                    clientes[mensaje.UsernameDestino].RecibirMensaje(mensaje);
                }
                catch
                {
                    clientes.Remove(mensaje.UsernameDestino);
                }
            }
        }

        1 referencia
        public Mensaje[] ObtenerHistorialMensajes(int idUsuario, string usernameDestino)
        {
            return _repo.ObtenerHistorialPorUsername(idUsuario, usernameDestino).ToArray();
        }
    }
}
```

LOBBY

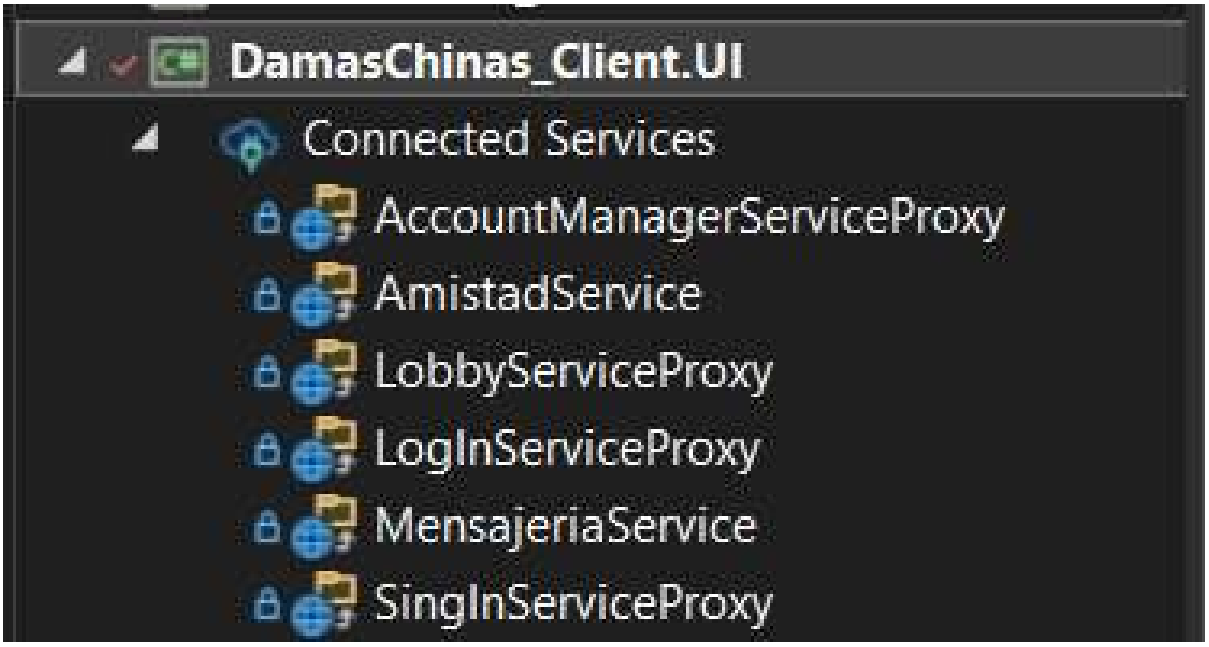
Funcionalidad: Creación, unión y gestión dinámica de lobbies multijugador.

Características: Permite la creación de lobbies privados o públicos por parte del usuario anfitrión, la unión de otros jugadores mediante un código de invitación, y la comunicación en tiempo real entre los miembros de un mismo lobby.

Internacionalización: Inglés y español. Estado: **En funcionamiento.** El cliente y servidor ya se comunican correctamente mediante WCF, el **LobbyManager** del cliente está implementado y los servicios del servidor están en ejecución estable.

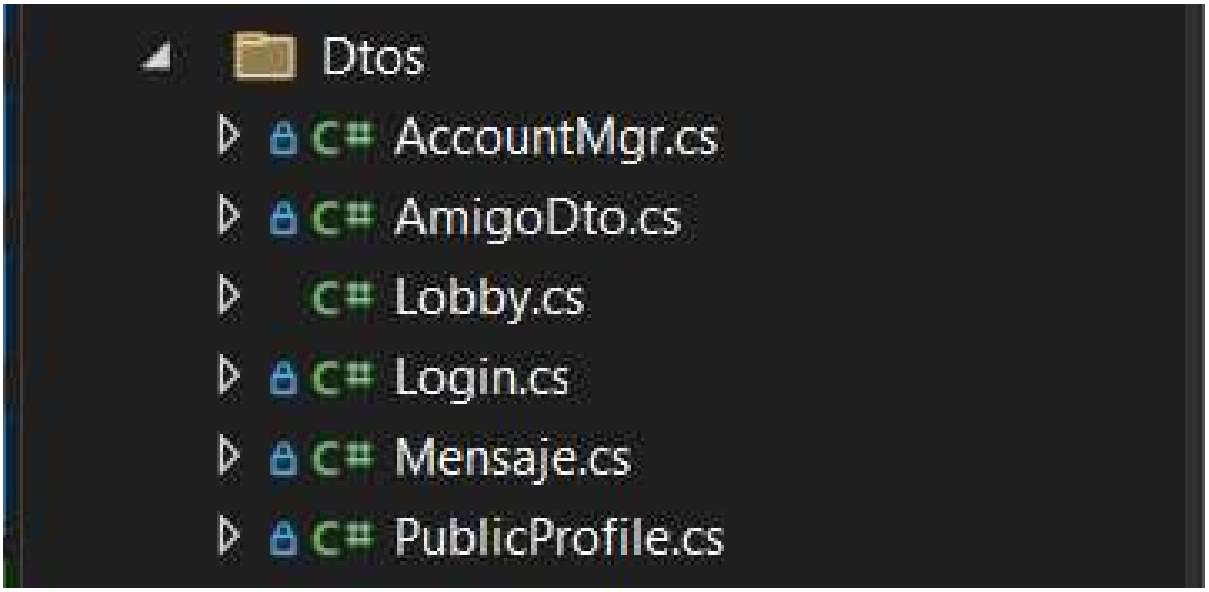
Descripción de la implementación

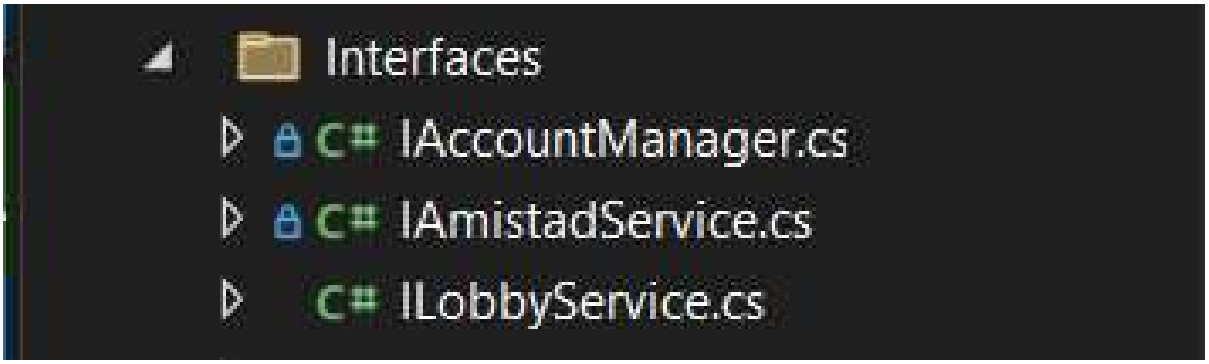
La estructura del sistema se divide en **Cliente (UI)** y **Servidor (WCF Services)**. El **Cliente** utiliza el proxy `LobbyServiceProxy` generado automáticamente desde el servicio del servidor para gestionar la comunicación, mientras que el **Servidor** expone la lógica de negocio a través del contrato `ILobbyService` y el **callback contract** `ILobbyCallback`.

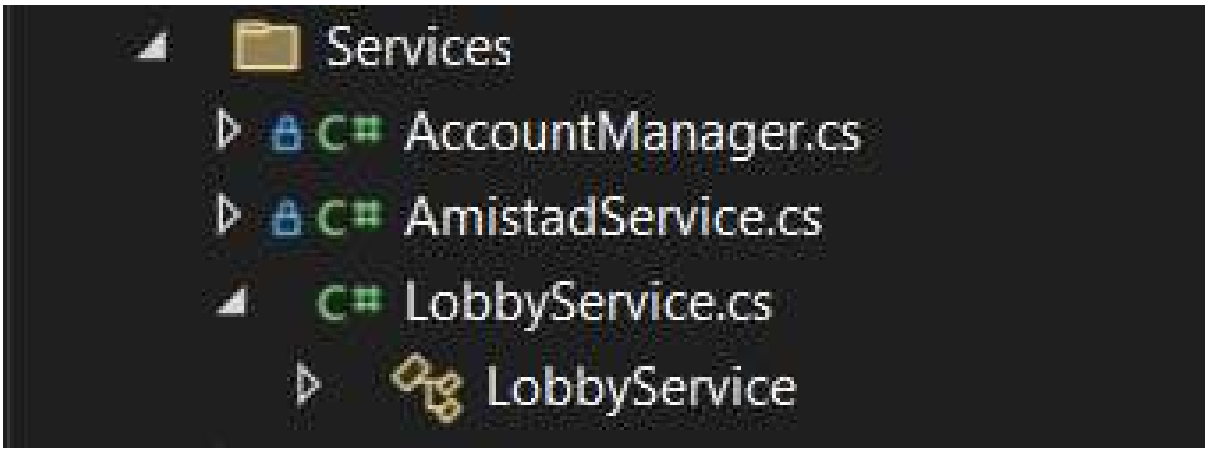


Estructura del Servidor

- **Carpeta Dtos:** contiene las clases `Lobby`, que representan la sesión multijugador y los jugadores conectados.
- **Carpeta Interfaces:** define `ILobbyService` (contrato principal) e `ILobbyCallback` (canal de retorno que notifica eventos a los clientes).
- **Carpeta Services:** contiene `LobbyService.cs`, que implementa toda la lógica del sistema de lobbies.







ServiceContract y Callback

El servidor implementa las siguientes operaciones mediante WCF:

- `CreateLobby(hostUserId, hostUsername, isPrivate)` → crea un nuevo lobby y registra al host.
- `JoinLobby(code, userId, username)` → permite a un jugador unirse mediante el código.
- `LeaveLobby(code, userId)` → remueve al jugador del lobby.
- `SendLobbyMessage(code, userId, username, message)` → envía mensajes a todos los miembros conectados.
- `StartGame(code)` → notifica a todos los jugadores que la partida ha comenzado.

El `ILobbyCallback` define los métodos que el servidor invoca en los clientes: - `OnMemberJoined(LobbyMember member)` - `OnMemberLeft(int userId)` - `OnMessageReceived(int userId, string username, string message, string utcIso)` - `OnLobbyClosed(string reason)` - `OnGameStarted(string code)`

Estructura del Cliente

El cliente cuenta con una capa de utilidades (`Utilities`) que maneja la conexión con los servicios del servidor y los eventos recibidos por medio de callbacks.

- **LobbyManager.cs**: clase encargada de gestionar la conexión con el servicio `LobbyService`, utilizando el proxy generado por **Connected Services**.
- Implementa la interfaz `ILobbyServiceCallback` para recibir las notificaciones del servidor en

tiempo real.

- Expone eventos públicos (`MemberJoined`, `MemberLeft`, `MessageReceived`, `LobbyClosed`, `GameStarted`) a los que la UI puede suscribirse para actualizar las vistas.

Flujo de comunicación

1. El usuario crea o se une a un lobby desde la interfaz WPF.
2. `LobbyManager` crea una instancia de `LobbyServiceClient` con un `InstanceContext` para manejar los callbacks.
3. El servicio `LobbyService` en el servidor agrega el usuario al lobby y notifica al resto mediante `OnMemberJoined`.
4. La UI del cliente actualiza la vista de jugadores conectados automáticamente mediante eventos suscritos.

Avance del Proyecto Final: Juego Damas Chinas (Entrega 3)

SETH

- **Creacion e implementacion de el servicio de mensajes:** 100%
- **Creacion del servicio de amigos** 100%
- **Creacion del servicio de mensajes** 100%
- **Creación de documento ASCII-DOC:** 50%
- **Implemtacion de servicio amigos en cliente** 100%
- ***Implementacion de servicio de mensajes en cliente** 100%
- **Modificacion de la base de datos** 50%

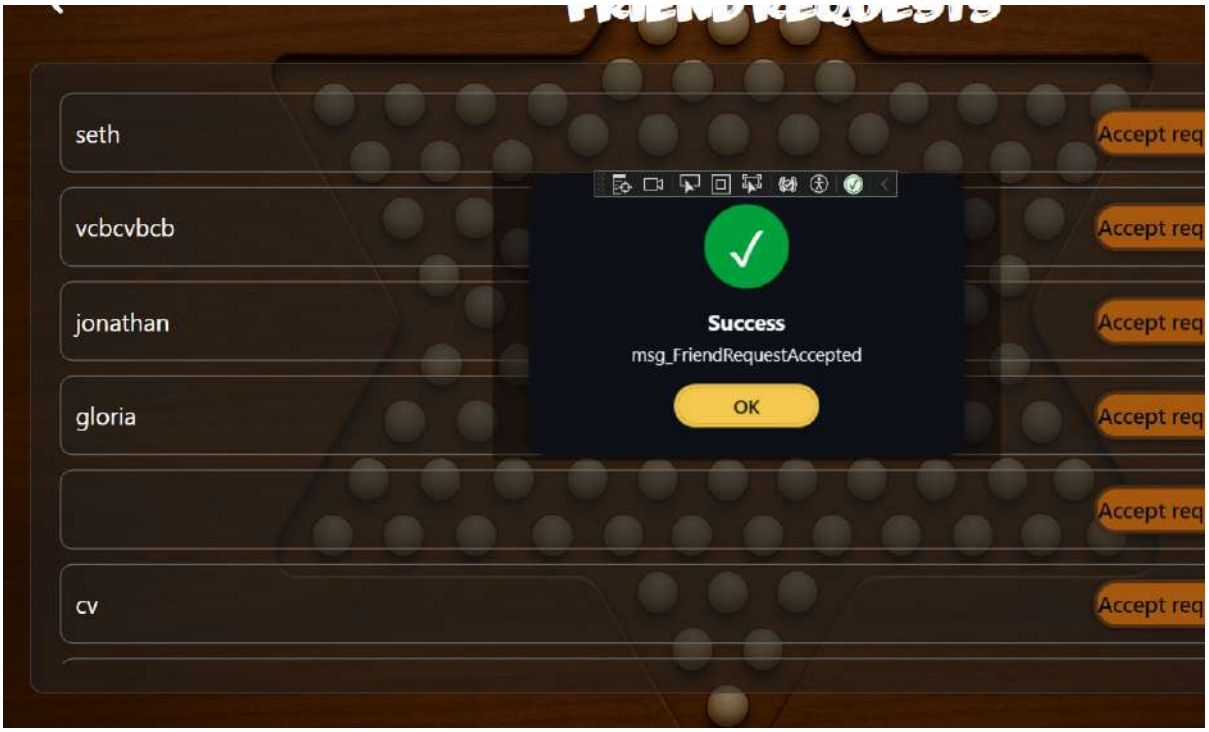
IVAN

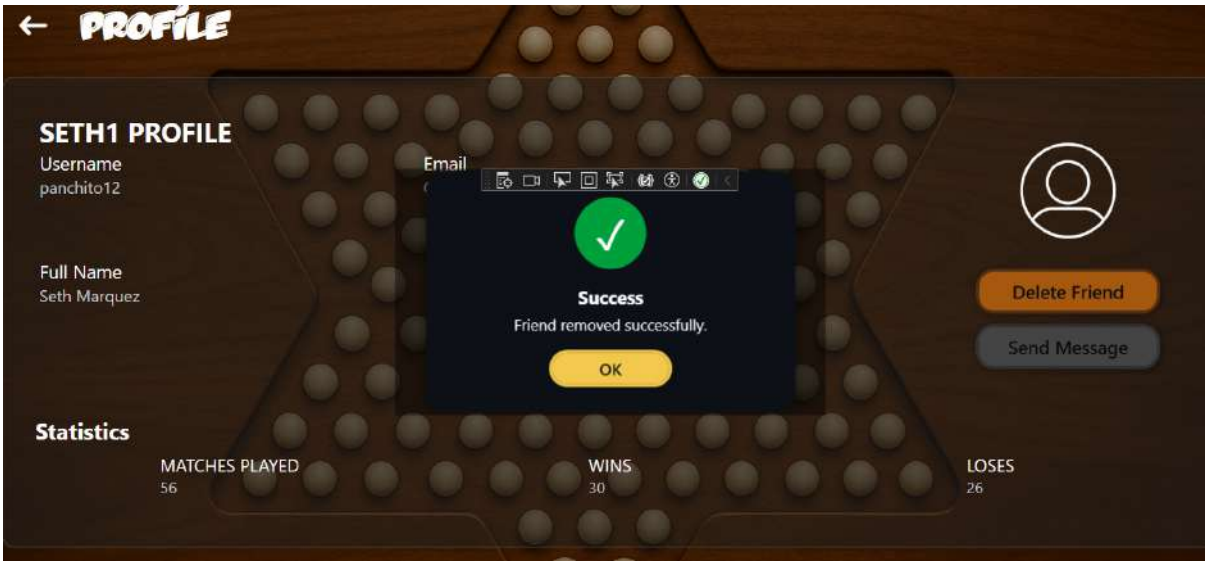
- **GUI:** 100%
- **Creacion e implementación del servicio de Lobby de juego** 100%
- **Creación de documento ASCII-DOC:** 50%
- **Implementacion de servicio de lobby en cliente** 100%
- **Modificacion de la base de datos** 50%

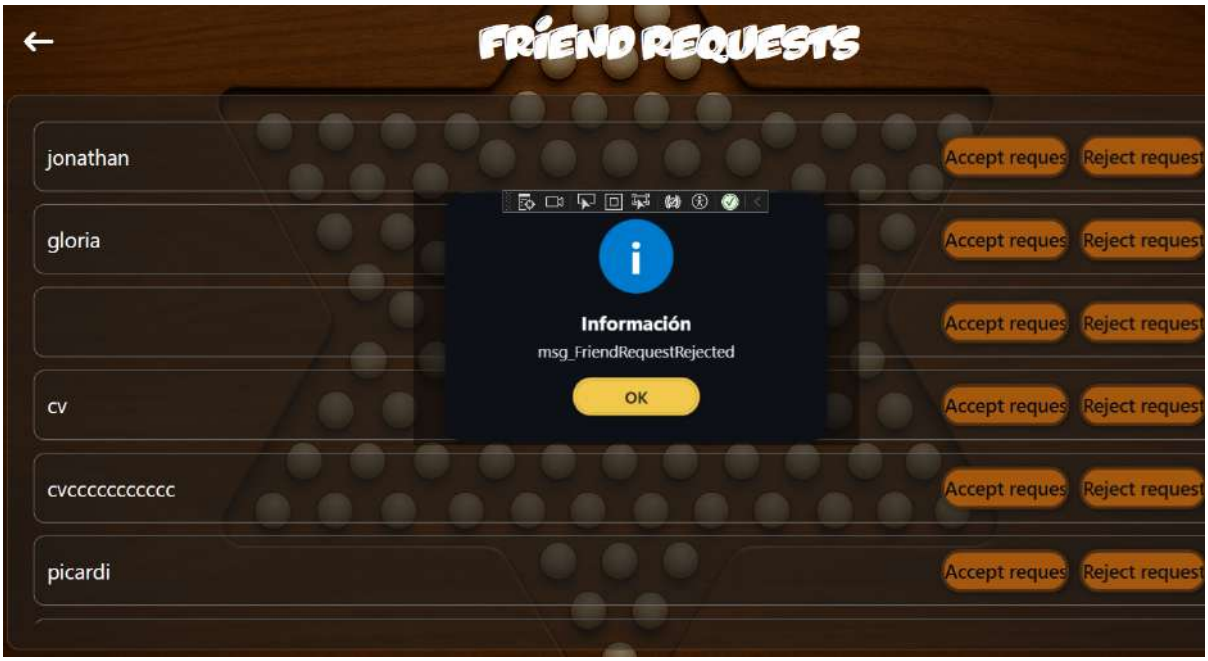
Cuarto reporte de avance del proyecto final: Juego Damas Chinas

Servidor

Servicio de amigos







El repositorio **FriendRepository** centraliza toda la lógica de relaciones entre usuarios del sistema:

- Maneja solicitudes de amistad y su ciclo completo.
- Gestiona creación y eliminación de amistades.
- Administra bloqueos con limpieza automática de relaciones previas.
- Devuelve información lista para ser consumida por los servicios WCF.
- Garantiza que todas las operaciones sean consistentes y seguras dentro del servidor.

Obtener solicitudes de amistad pendientes

Método: `GetFriendRequests(string username)`

- Obtiene el ID del usuario receptor.
- Busca solicitudes con estado **"pendiente"**.

- Carga los perfiles asociados mediante `Include()`.
- Construye objetos `FriendDto` con información lista para la UI:
- Id del emisor.
- Username del emisor.
- Avatar.
- Estado de conexión usando `SessionManager.IsOnline`.

Función: Devuelve todas las solicitudes de amistad entrantes que aún no han sido atendidas.

Enviar una solicitud de amistad

Método: `SendFriendRequest(string senderUsername, string receiverUsername)`

Realiza varias validaciones antes de permitir crear una solicitud:

- Evita que un usuario se envíe solicitud a sí mismo.
- Verifica que no sean ya amigos.
- Verifica que no exista un bloqueo entre ambos.
- Comprueba que no exista ya una solicitud pendiente.

Si todas las validaciones son correctas, crea un nuevo registro en `solicitudes_amistad` con estado **"pendiente"**.

Función: Registra una nueva solicitud asegurando consistencia y evitando duplicados.

Actualizar el estado de una solicitud

Método: `UpdateFriendRequestStatus(string receiverUsername, string senderUsername, bool accept)`

Gestión completa de solicitudes pendientes:

- Localiza la solicitud entre emisor y receptor.
- Si `accept == true`:
- Revisa que no exista un bloqueo activo.
- Verifica si ya existía una amistad (para evitar duplicados).
- Crea la amistad si es necesario.
- Marca la solicitud como **"aceptada"**.
- Si `accept == false`:
- Marca la solicitud como **"rechazada"**.

Función: Permite aceptar o rechazar solicitudes mediante una sola operación.

Crear amistad directamente

Método: `AddFriend(int idUsuario1, int idUsuario2)`

- Evita que un usuario se agregue a sí mismo.
- Verifica que ambos usuarios existan en la base de datos.
- Revisa que no exista ya una amistad previa.
- Crea el registro en la tabla `amistades`.

Función: Crea una relación de amistad válida entre usuarios.

Eliminar una amistad

Método: `DeleteFriend(int idUsuario1, int idUsuario2)`

- Busca una amistad entre ambos usuarios en cualquier dirección.
- Si existe, elimina la relación de la base.

Función: Permite terminar una relación de amistad.

Bloquear o desbloquear usuarios

Método: `UpdateBlockStatus(string blockerUsername, string blockedUsername, bool block)`

Cuando `block == true`: - Verifica si el bloqueo ya existe. - Elimina cualquier amistad previa entre ambos usuarios. - Elimina solicitudes de amistad enviadas entre ellos. - Crea un nuevo registro en la tabla `bloqueos`.

Cuando `block == false`: - Verifica que exista un bloqueo activo. - Lo elimina de la base de datos.

Función: Gestiona bloqueos de usuarios, impidiendo interacciones futuras hasta que el bloqueo sea retirado.

Solicitud de codigo sing in

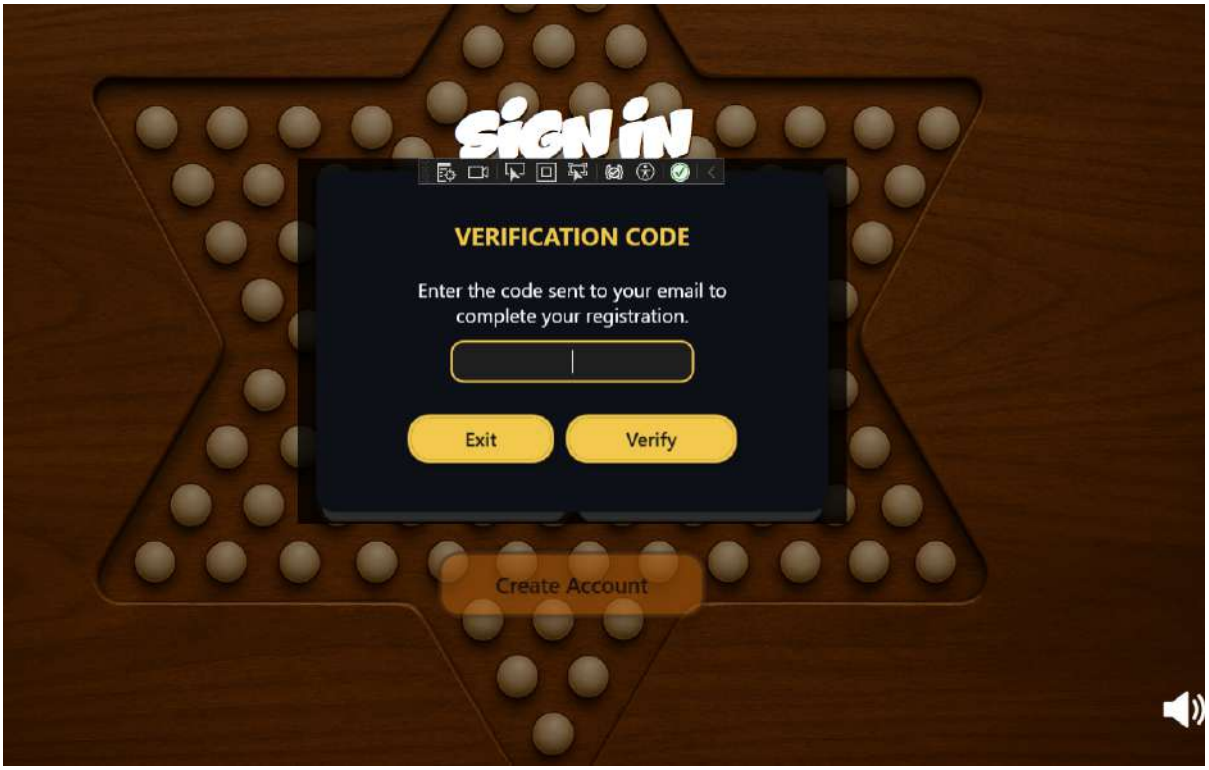
El sistema implementa un mecanismo de verificación por correo para asegurar que cada usuario realmente es dueño del correo con el que intenta registrarse. Este proceso añade una capa esencial de seguridad y evita la creación de cuentas fraudulentas.

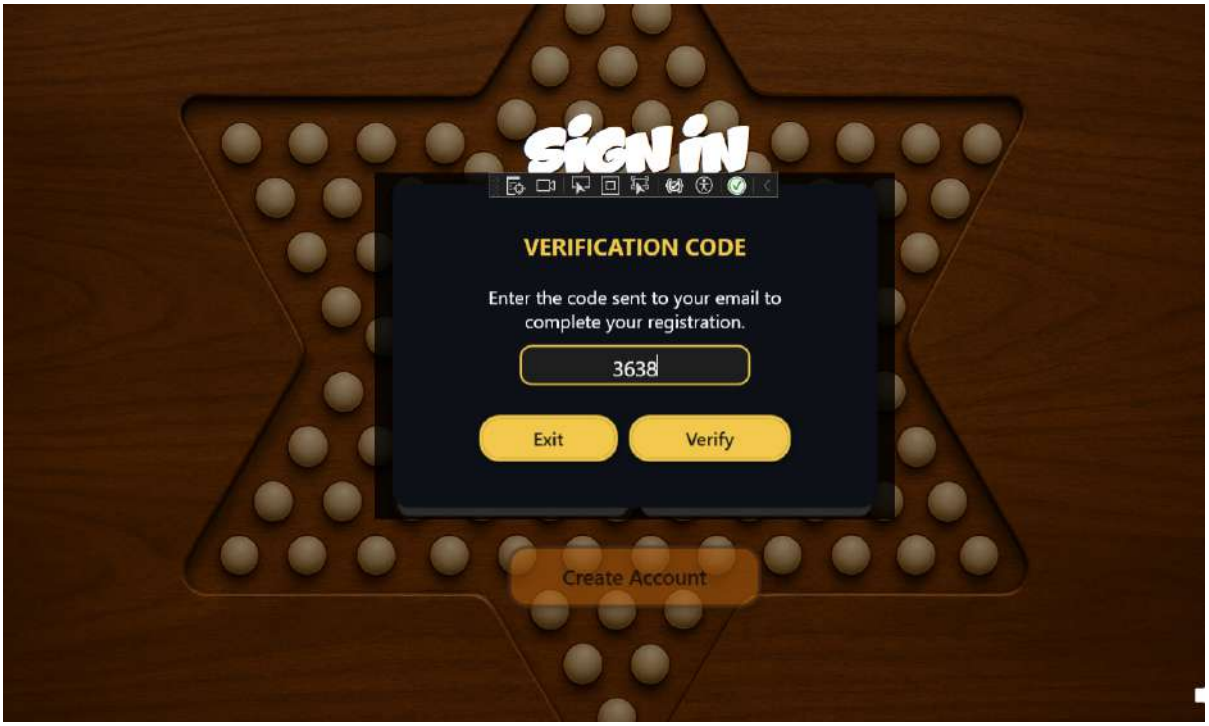
Cuando el usuario solicita el código, el servidor genera un número único y lo guarda temporalmente junto con la hora exacta en que fue creado. Este código se envía al correo del usuario y tiene una vigencia limitada de 5 minutos. Durante este tiempo, el usuario debe introducirlo en la aplicación para continuar con su registro.

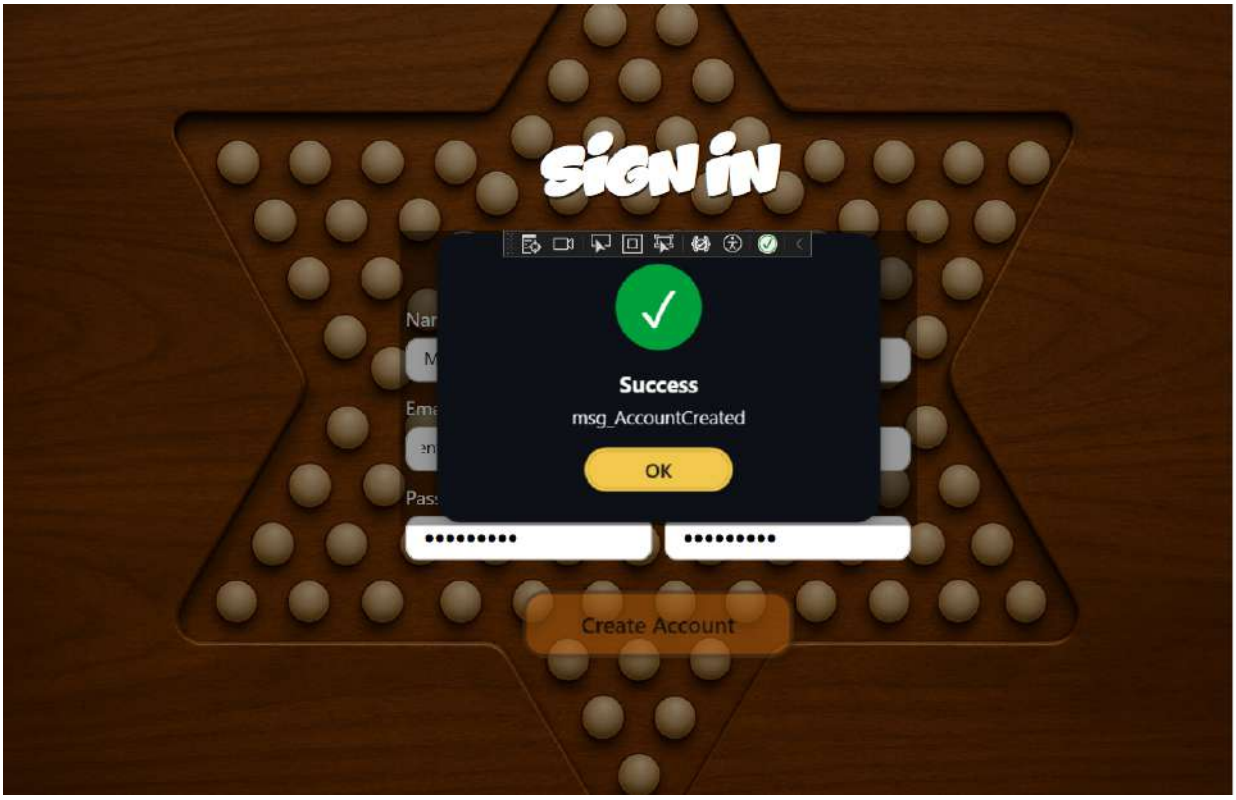
Al recibir el código ingresado, el servidor realiza tres validaciones: 1. **Existencia:** confirma que haya un código generado para ese correo. 2. **Exactitud:** verifica que el código ingresado coincida con el enviado. 3. **Validez temporal:** comprueba que no hayan pasado más de 5 minutos desde su creación.

Si cualquiera de estas condiciones falla, el servidor rechaza el registro por motivos de seguridad. Si todo es correcto, el código se elimina y el usuario queda autorizado para completar su creación de cuenta.

Este mecanismo garantiza que los correos registrados sean reales, dificulta intentos de suplantación y refuerza la integridad del proceso de registro.






SIGN IN
Success
msg_AccountCreated
OK
Create Account


7:40 Lun, 17 de nov
Internet
Totalplay-AFA4_2.
Bluetooth
Linterna
No interrumpir
Activado
Gmail • cuentadesoport... • ahora
damaschinas4u
Código de verificación
Tu código de verificación es: 3638
Este código expirará en 5 minutos.
Archivar
Marcar como leído
Responder
Cerrar todo

Cliente

Creación y Refactorización de Nuevas Páginas XAML

A continuación se enlistan **todas las ventanas XAML** que fueron creadas, actualizadas o refactorizadas durante esta semana:

- Friends.xaml
- ChatWindow.xaml
- PreLobby.xaml
- ChangeData.xaml
- SignIn.xaml
- MenuRegisteredPlayer.xaml
- ProfilePlayer.xaml
- Componentes de Popups (PopupError.xaml, PopupWarning.xaml, PopupSuccess.xaml)
- LoadedBar.xaml (nuevo componente)
- Archivos de estilos globales (Buttons.xaml, TextBoxes.xaml, Cards.xaml, Colors.xaml, GeneralTheme.xaml)

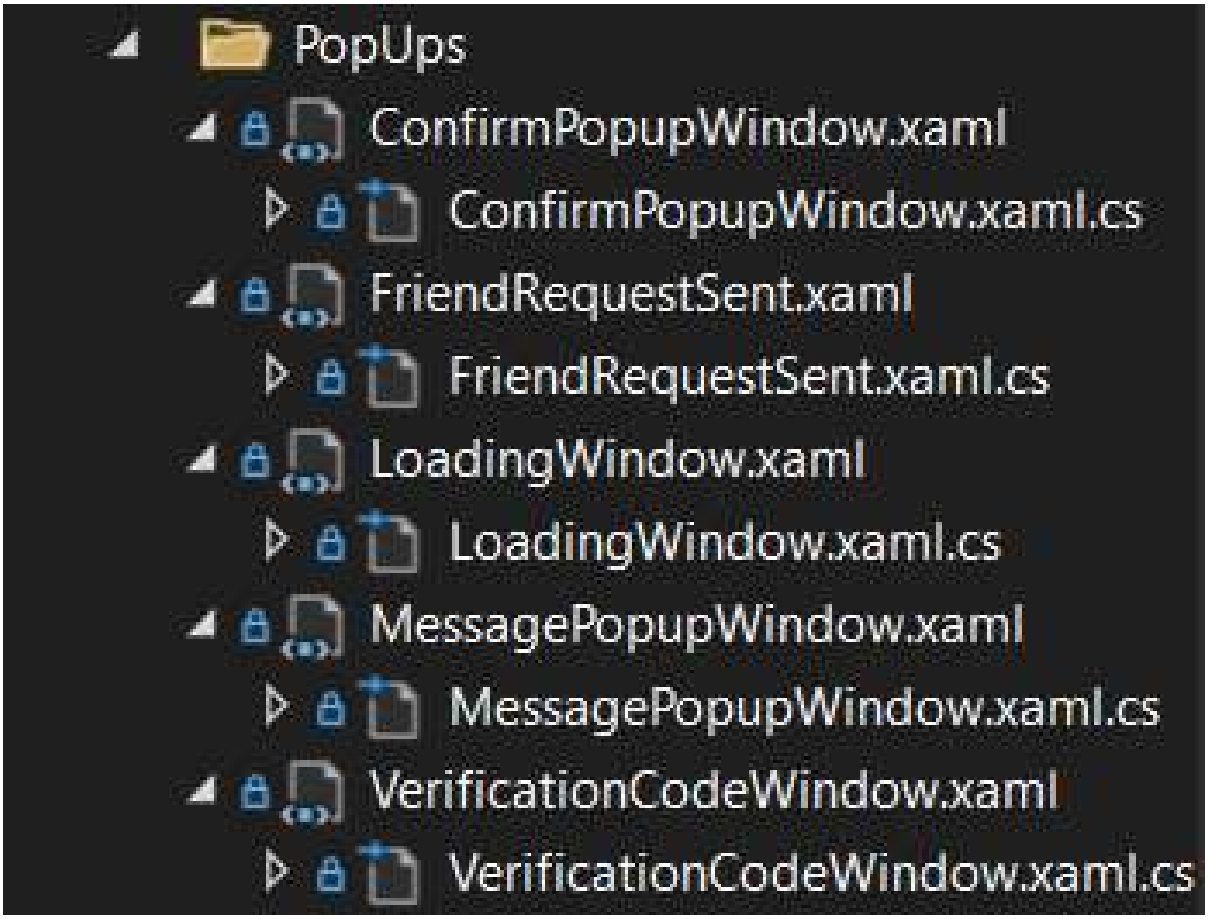
Durante esta entrega, todas estas vistas recibieron mejoras orientadas a la estandarización visual y técnica del proyecto. Se eliminaron textos hardcodeados para reemplazarlos por claves internacionalizadas, se reorganizó la estructura visual siguiendo el estándar interno, se aplicaron nuevos estilos globales unificados, se corrigieron bindings, se integró el nuevo sistema de popups, se añadió soporte para la barra de carga (LoadedBar) en vistas clave y se ordenaron los recursos utilizados por cada ventana. Estos cambios garantizan coherencia gráfica, mantenibilidad, escalabilidad y una experiencia de usuario más fluida.

Implementación de Popups Reutilizables

Durante esta semana se consolidó un sistema de ventanas emergentes (popups) completamente modular, orientado a sustituir cualquier uso directo de `MessageBox` o mensajes hardcodeados. Los popups fueron integrados con los estilos globales del proyecto, con el sistema de internacionalización y con el controlador unificado de mensajes.

Las ventanas emergentes creadas o ajustadas durante esta entrega son las siguientes:

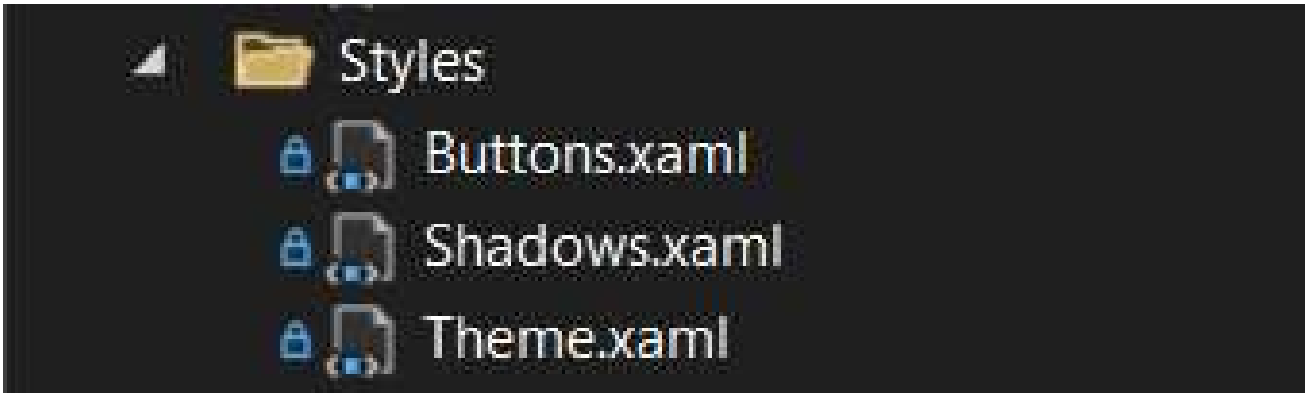
- ConfirmPopupWindow.xaml
- FriendRequestSent.xaml
- LoadingWindow.xaml
- MessagePopupWindow.xaml
- VerificationCodeWindow.xaml



Este sistema unificado permite que cualquier ventana del proyecto utilice popups estilizados, sin hardcode y totalmente coherentes con el estándar visual y funcional del proyecto.

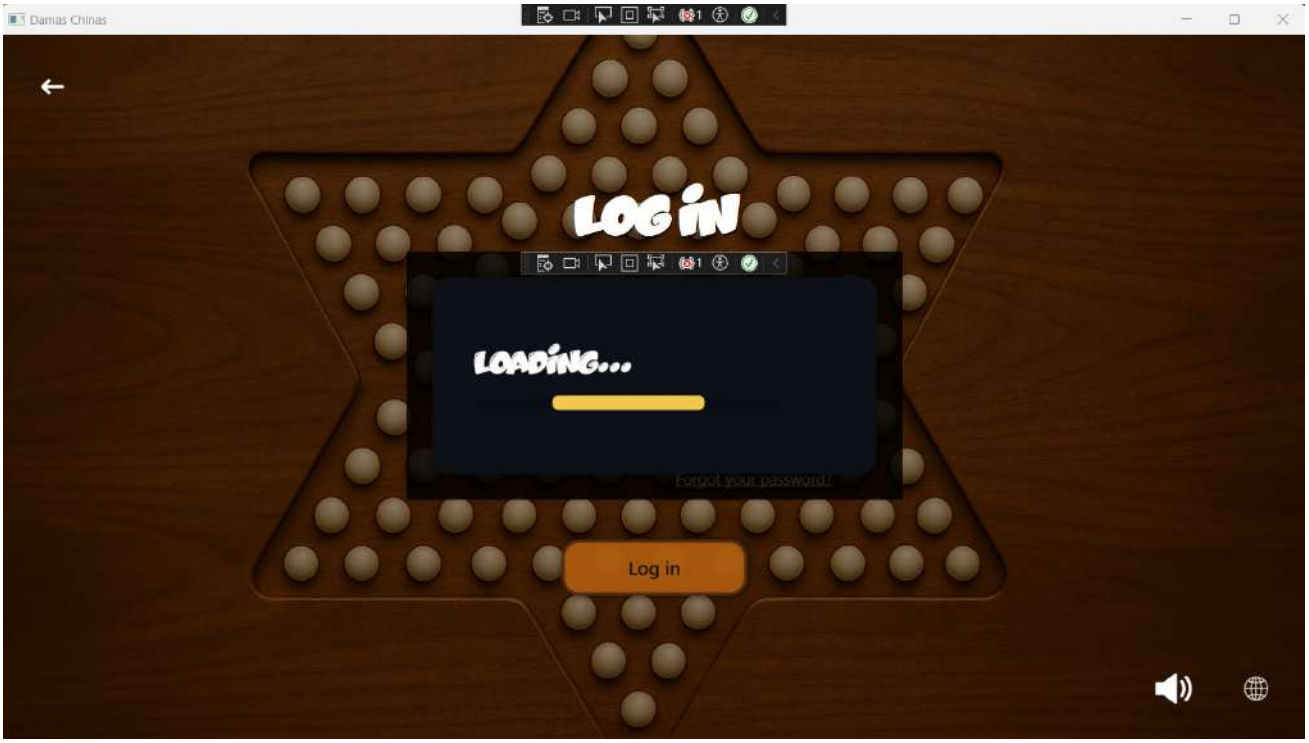
Nuevos Estilos Globales (WPF)

Uno de los avances más importantes fue la creación y reorganización de estilos globales para toda la UI: botones, tarjetas, inputs, temas y colores.



LoadedBar Utility (Barra de Carga)

Se implementó la barra **LoadedBar**, utilizada para representar operaciones del servidor que requieren tiempo (por ejemplo, carga de inicio de sesión o registro).



Esta utilidad es completamente modular y reutilizable.

Sistema de Internacionalización Sin Hardcode

Se completó la integración del sistema estandarizado que reemplaza completamente el uso de textos directos (hardcode) tanto en la interfaz como en la lógica del cliente. El flujo completo de un mensaje ahora funciona así:

MessageCode (Servidor) → MessageTranslator (Cliente) → MessageHelper → Diccionario de recursos → Popup/UI

Este flujo garantiza coherencia, escalabilidad y soporte multilenguaje en toda la aplicación.

MessageCode

El servidor nunca envía textos. Cada operación devuelve únicamente un `MessageCode`, lo que evita inconsistencias y permite que el cliente controle completamente la presentación de los mensajes.

MessageTranslator

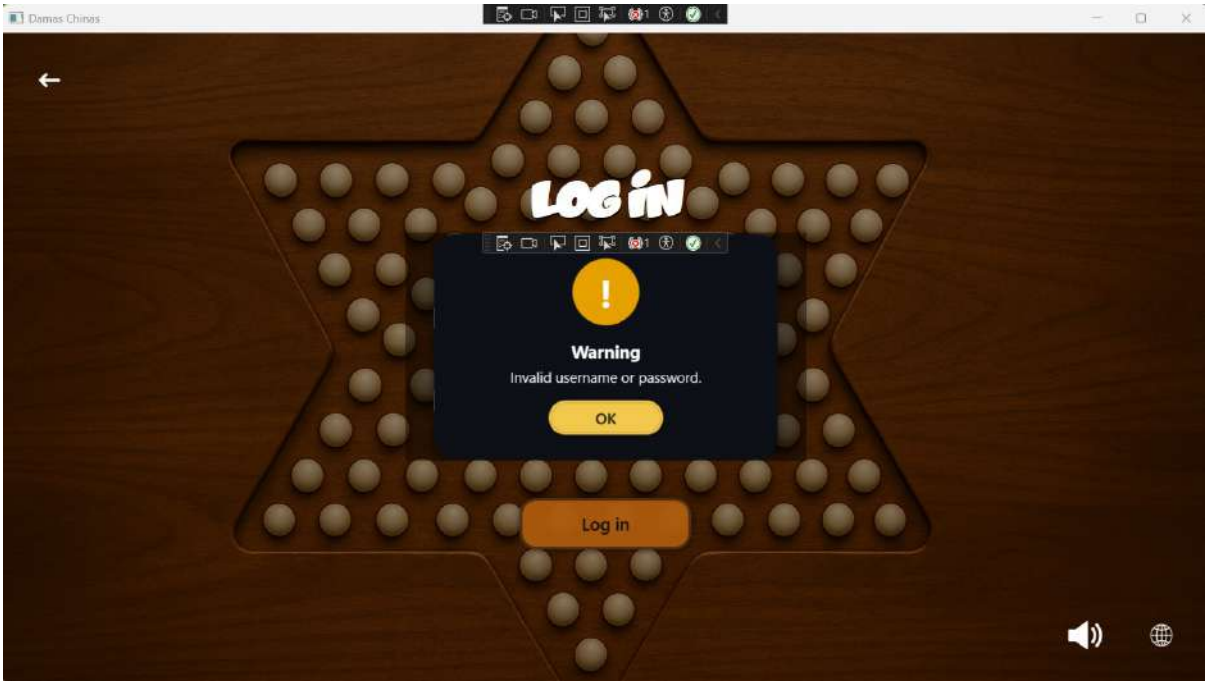
El `MessageTranslator` es el componente responsable de convertir un `MessageCode` en el texto correspondiente según el idioma activo del sistema. Este traductor consulta los diccionarios de recursos (`Lang.en.xaml`, `Lang.es.xaml`) y retorna el texto final que debe mostrarse. Gracias a este componente, ninguna vista ni lógica del cliente necesita conocer claves ni diccionario: solo recibe la traducción ya lista.

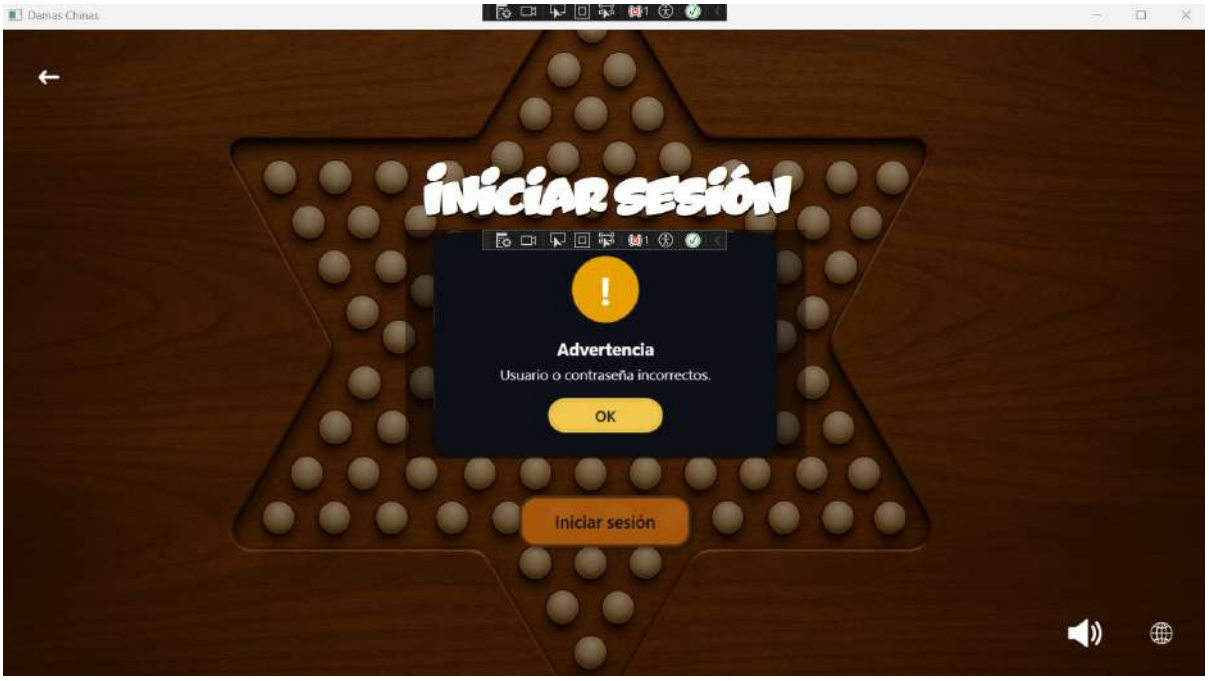
MessageHelper

El `MessageHelper` es la capa que usa la traducción generada por `MessageTranslator` y decide **cómo mostrarla** en la interfaz. Es el encargado de:

- Mostrar el popup correspondiente (éxito, advertencia o error).
- Aplicar estilos globales.
- Centralizar toda la lógica de visualización de mensajes.
- Evitar duplicación de código en cada ventana.
- Asegurar que todos los mensajes pasen por el mismo proceso estándar.

En otras palabras, **MessageTranslator resuelve qué mensaje mostrar y MessageHelper decide cómo mostrarlo**.





Diccionarios de idioma reorganizados

Los archivos de idioma fueron reorganizados y estructurados por claves relacionadas con los códigos del servidor. Esto permite incorporar fácilmente nuevos mensajes o modificar textos sin tocar la lógica del cliente.

Con este sistema, toda la interfaz queda completamente preparada para soportar cualquier idioma y garantiza que todos los mensajes visibles cumplan con el estándar del proyecto, sin hardcode y con una experiencia consistente en todas las vistas.

Corrección Formal del Manejo de Excepciones

Se reemplazó por completo:

- `catch(Exception)`
- Mensajes escritos directamente en UI
- MessageBox directos

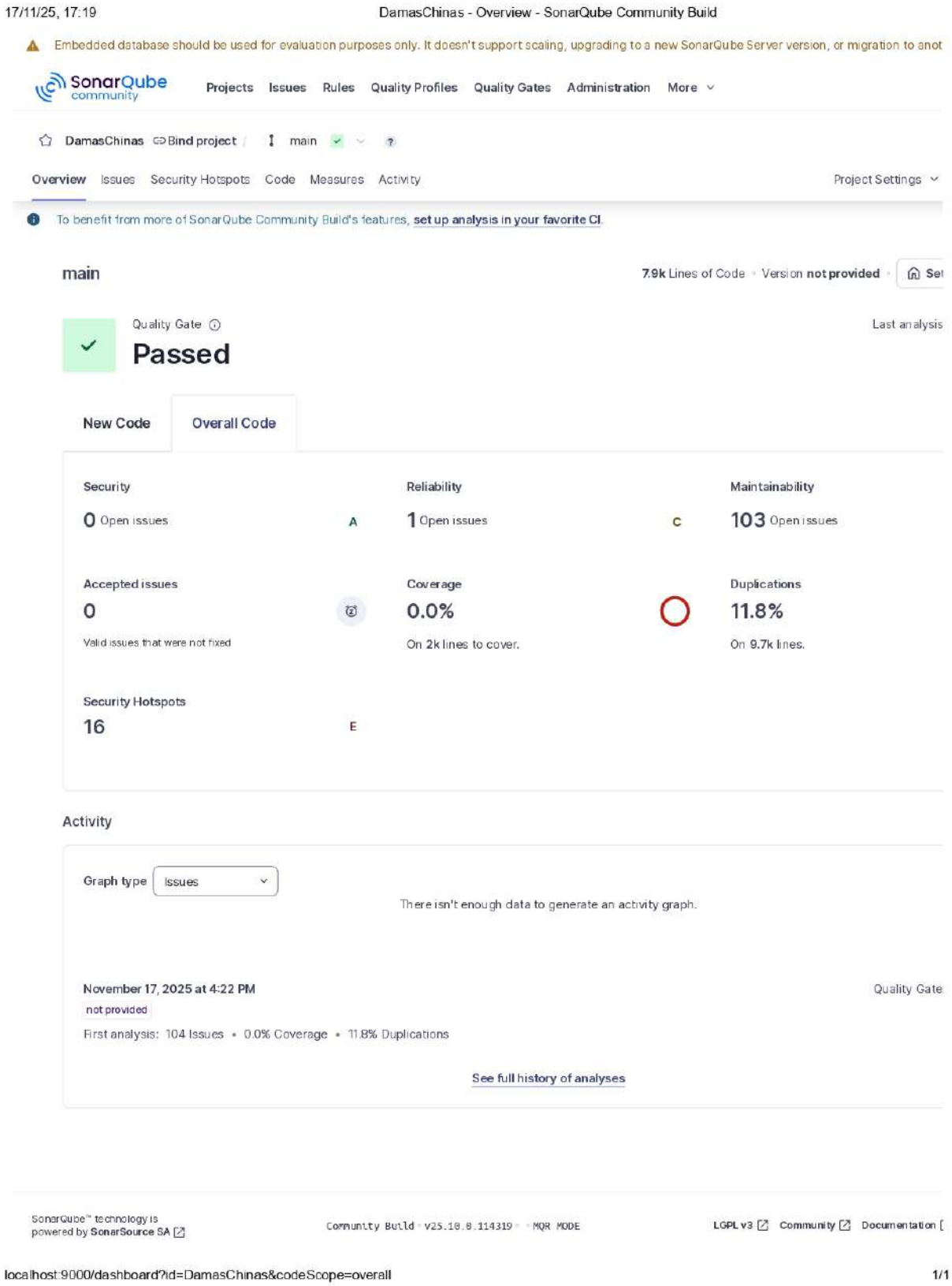
por un sistema basado en:

- `OperationResult`
- `MessageHelper`
- `MessageTranslator`
- `TryExecuteAction()`

Preparación del Entorno de Análisis con SonarQube

Durante esta semana no se aplicaron correcciones derivadas de SonarQube; sin embargo, se realizó toda la preparación necesaria para habilitar el análisis de calidad del código del proyecto. Esto incluyó la instalación de SonarQube Community Edition, su configuración local, la habilitación del servicio en el sistema y la integración del proyecto Damas Chinas a través de una clave de proyecto y token de autenticación.

Asimismo, se instaló y configuró **SonarScanner for MSBuild**, permitiendo ejecutar análisis completos de la solución desde la consola de Visual Studio. Con esto, el proyecto queda listo para comenzar a detectar code smells, duplicaciones, problemas potenciales de mantenibilidad y violaciones al estándar de codificación en futuras iteraciones.



Resumen de contribución por integrante

Rodrigo Iván Ahumada Rodríguez

Durante esta semana se avanzó significativamente en la interfaz del cliente y en la estructura interna del proyecto. Se completó la creación y refactorización de múltiples ventanas XAML, asegurando el cumplimiento del estándar interno de diseño, eliminando hardcode visual y organizando los componentes para que su mantenimiento sea más claro y escalable. También se implementó un sistema completo de popups reutilizables, con soporte para internacionalización y estilos globales.

Asimismo, se integró la barra de carga (LoadedBar) como componente reutilizable para operaciones remotas, se reorganizó la estructura de recursos (imágenes, diccionarios y estilos) y se completó el sistema de internacionalización basado en MessageCode, MessageTranslator y MessageHelper. Se corrigió el manejo de mensajes visibles al usuario, centralizando todo en una lógica común.

Finalmente, se realizó toda la preparación del entorno de análisis de calidad utilizando SonarQube, incluyendo su instalación, configuración inicial, generación del token de proyecto y la integración con SonarScanner para ejecutar análisis locales. Esto deja el proyecto listo para comenzar con correcciones de calidad en futuras entregas.

- Refactor y creación de múltiples ventanas XAML → **100%**
- Implementación del sistema centralizado de popups → **80%**
- Integración del sistema de internacionalización sin hardcode → **100%**
- Implementación de LoadedBar como componente reutilizable → **100%**
- Reorganización de recursos y estilos globales → **100%**
- Manejo unificado de mensajes con MessageCode → **90%**
- Preparación completa del entorno de análisis SonarQube → **100%**

Seth Marquez Rodriguez

Seth Márquez Márquez

Durante esta semana se avanzó en la lógica del servidor, completando la implementación del sistema de solicitudes de amistad, la aceptación y rechazo de solicitudes, la gestión de amistades y el módulo de bloqueos entre usuarios. Todo este flujo fue desarrollado con validaciones completas que aseguran la integridad de los datos, evitan duplicados y previenen erroresdeinteracciones entre jugadores. Asimismo, se organizó y consolidó el repositorio `FriendRepository`, responsable de centralizar la lógica social del proyecto.

Además, se implementó el sistema de verificación por correo para el registro de nuevos usuarios, el cual incluye la generación del código, su almacenamiento temporal, validación con vigencia limitada y eliminación segura tras su uso. Este mecanismo incrementó la seguridad del proceso de registro y garantizó que solo correos válidos y comprobados puedan registrar cuentas.

Cabe mencionar que se trabajo en la implementacion de estas funcionalidades en el cliente siguiendo la base dejada por mi compañero.

En conjunto con lo anterior, se realizó una refactorización general del servidor, reduciendo complejidad ciclomática, ordenando rutas de servicios, mejorando la claridad del código y normalizando el manejo de excepciones. También se eliminaron estructuras innecesarias, se unificó la lógica distribuida en métodos extensos y se optimizaron los repositorios para una mayor consistencia y mantenibilidad. Con estos trabajos, la capa del servidor quedó más estable, limpia y preparada para futuras ampliaciones del sistema.

- Lógica completa del servidor (WCF) → **100%**
- Creacion e iplementacion de el modulo de control de amigos → **100%**
- Creacion e implementacion de la funcionalidad de codigos → **100%**
- Refactorización para reducir deuda tecnica y alto acomplamiento en el servidor → **100%**
- Elaboracion del reporte ASCI-DOC → **80%**
- Manejo unificado de mensajes con MessageCode → **10%**
- Implementación del sistema centralizado de popups → **80%**