Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio Diseño de Aplicaciones

2022

Resumen

El presente trabajo tiene como finalidad la elaboración de una aplicación, desarrollada en C#, aplicando las técnicas de diseño, manejando un control de versiones, utilizando el manejo de errores y cuidando la calidad del código tal como se vio a lo largo del curso.

Índice

Descripción del trabajo y el sistema

La empresa Threat Level Midnight Entertainment desea lanzarse como competidora de los grandes proveedores de streaming de series y películas, para lo cual ha solicitado que se desarrolle una POC (Prueba de Concepto) de su nueva plataforma.

Este sistema fue desarrollado como una aplicación de escritorio (Windows Forms), en el lenguaje C# y utilizando .NET Framework, la aplicación le permite al usuario:

Registro e inicio de sesión de un usuario: los usuarios de la aplicación pueden registrarse como tales introduciendo su nombre, apellido, correo electrónico y creando una contraseña. Luego de registrados, los usuarios pueden pasar a iniciar sesión.

Creación de usuario administrador: el sistema cuenta con un usuario administrador el cual se va a encargar de mantener las películas y los géneros. Las credenciales para acceder como este usuario son las siguientes, mail: admin@admin.com y contraseña: adminadmin.

Administración de los perfiles de usuarios: Cuando un usuario se registra en la plataforma este debe crear su perfil, para esto debe escoger un alias y un pin, el primer perfil creado va a ser el owner de la cuenta. El perfil owner es el encargado de crear nuevos perfiles, hasta tres perfiles más, y de definir si alguno de los perfiles es un perfil infantil, este último tipo de perfil no va a tener pin. Se decidió de todas maneras al momento de crear el perfil infantil solicitarle un pin por si en algún momento se decide implementar que un usuario infantil pase a ser uno regular.

Administración de películas: como se mencionó anteriormente el usuario administrador es el encargado de la gestión del contenido, esto quiere decir que puede crear y eliminar películas.

Administración de géneros: al igual que con las películas el usuario administrador es el encargado de administrar los géneros. Un género puede ser eliminado únicamente si no tiene ninguna película asociada.

Calificación de películas: En la pantalla principal debe verse un listado de películas. El usuario podrá acceder a su vista de detalle para calificarla. Una película puede ser calificada con -1 como voto negativo, +1 como voto positivo y ++1 como voto muy positivo.

Marcar película como vista: Al acceder al detalle de la película se podrá marcar como vista, simulando la reproducción de la película.

Ranking y personalización: los usuarios deben poder acceder a una vista de las películas al momento de ingresar a la página principal. De esta manera los usuarios van a poder seleccionar diferentes vistas para poder seleccionar qué película ver. El ranking por puntaje no se realiza por perfil, se toman en cuenta las votaciones de todos los perfiles.

Control de contenido apto para todo público: para los perfiles infantiles se deben filtrar las películas que no sean aptas para todo público y solo desplegar las que sí sean.

Alta, baja y modificación de actores o directores: debe ser posible dar de alta actores o directores.

Asociar o desasociar actores o directores de una película: debe ser posible asociar varios directores a una película así también como uno o varios actores.

Listado de películas vistas por un perfil: cada perfil debe poder ver una lista de las películas que ya vió en orden de visualización.

Búsqueda de películas por uno o varios actores: debe ser posible buscar películas dado uno o más actores ordenados alfabéticamente o por año de estreno.

Búsqueda de películas por uno o varios directores: debe ser posible buscar películas dado uno o más directores ordenados alfabéticamente o por año de estreno.

Descripción de diseño

Diagrama de paquetes

La solución desarrollada consta de cinco paquetes principales, estos paquetes son: LogicaNegocio, Dominio, TLMEGraficos, RepositorioBDD y PruebasUnitarias. Cada uno de estos paquetes contiene clases con una función específica, que se explicaran a continuación.

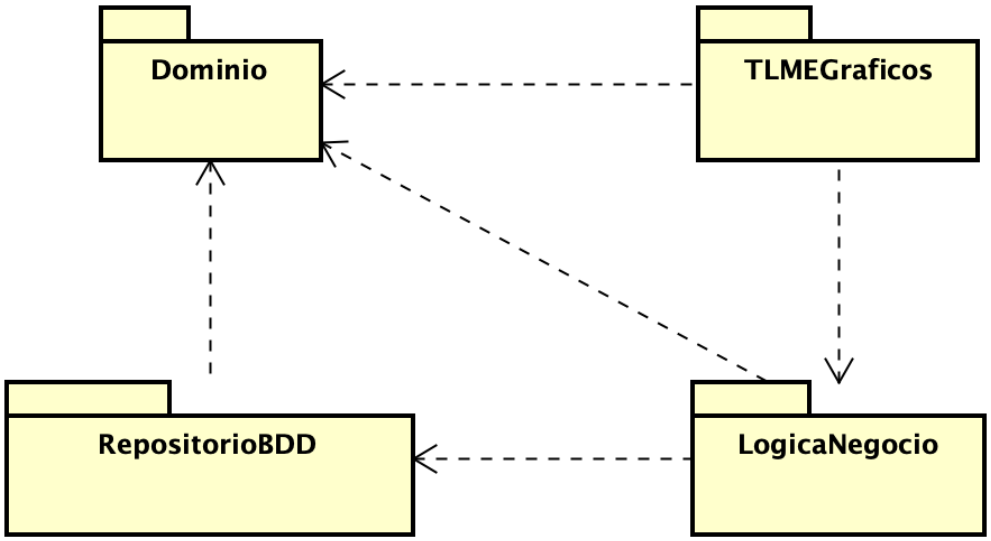


Imagen 1.1: Diagrama de Paquetes

LogicaNegocio: se encarga de manejar toda la lógica de negocio que se va a necesitar a lo largo de la utilización del sistema. En este paquete podemos encontrar todas las funcionalidades que se refieren al manejo de listas que se persistirán en la memoria.

TLMEGraficos: es donde se encuentran las interfaces de usuario. Estas le comunican la información y capturan la información ingresada por el usuario en un proceso determinado. El diseño de la interfaz tiene como objetivo facilitar al usuario la ejecución de una tarea. La interfaz es la parte visual de la aplicación. Esta comunica al usuario con toda la lógica del programa.

Tests: en este paquete se encuentran las clases con las pruebas unitarias para cada una de las clases de los paquetes ya nombrados. El objetivo de este paquete es aplicar la técnica TDD, con el proposito de facilitar el desarrollo del sistema.

RepositorioEnMememoria: este paquete se encarga principalmente de la persistencia en memoria de los datos ingresados por medio de la interfaz gráfica. A su vez cada entidad del domino tiene su respectiva clase dentro del paquete RepositorioEnMemoria, con la finalidad de persistir la información de las instancias de cada clase.

RepositorioBDD: este paquete se encarga principalmente de la persistencia en una base de datos de los datos ingresados por medio de la interfaz gráfica. A su vez cada entidad del domino tiene su respectiva clase dentro del paquete RepositorioBDD, con la finalidad de persistir la información de las instancias de cada clase.

Dominio: es donde se encuentran las clases que representan a las entidades fuertes de la solución. En esta clase se encuentran todos los métodos de validación de cada uno de los objetos que se utilizan en el sistema, métodos públicos y atributos.

Diagramas de Clases

A continuación se describen las clases del sistema, sus atributos, operaciones y las relaciones entre los objetos mediante el siguiente diagrama de clases.

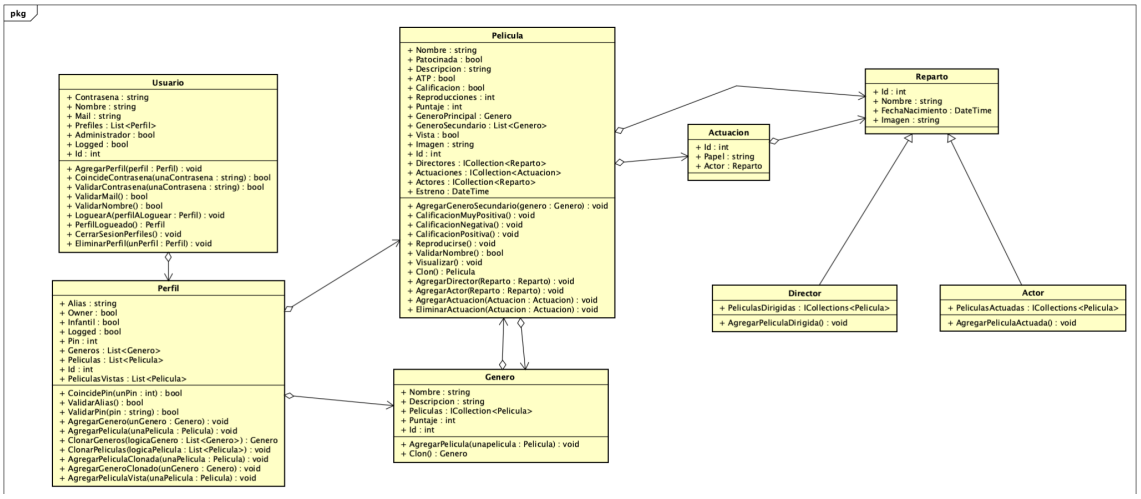


Imagen 2.1: Diagrama de Clases Dominio

El equipo entiende que el vínculo entre las clases Genero y Pelicula se podría haber realizado de una mejor manera, utilizando una clase que servirá como intermediario se obtendrá una cardinalidad para cada clase de 1::N, evitando tener una cardinalidad de N::N. Aunque consideramos que de la manera que se implementó es suficiente para cumplir con lo solicitado en la letra.

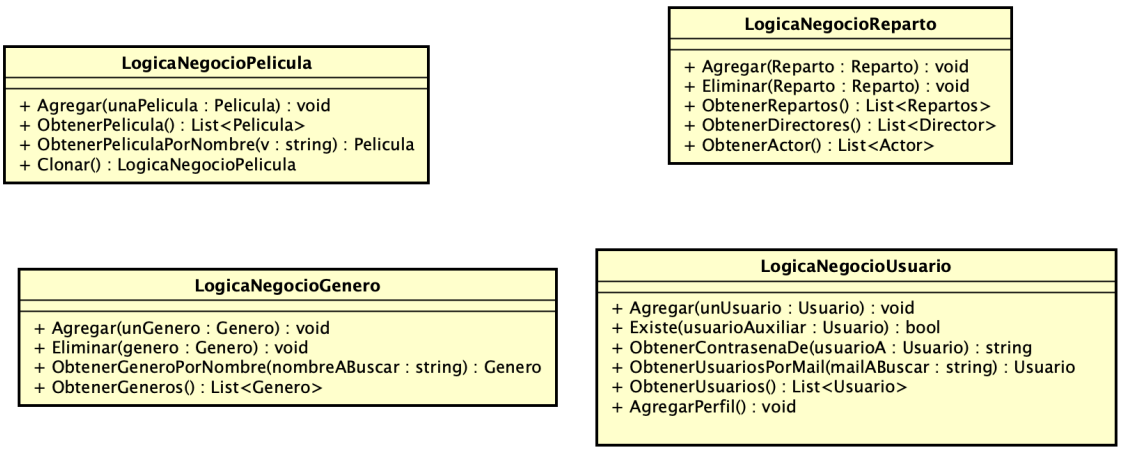


Imagen 2.2: Diagrama de Clases LogicaNegocio

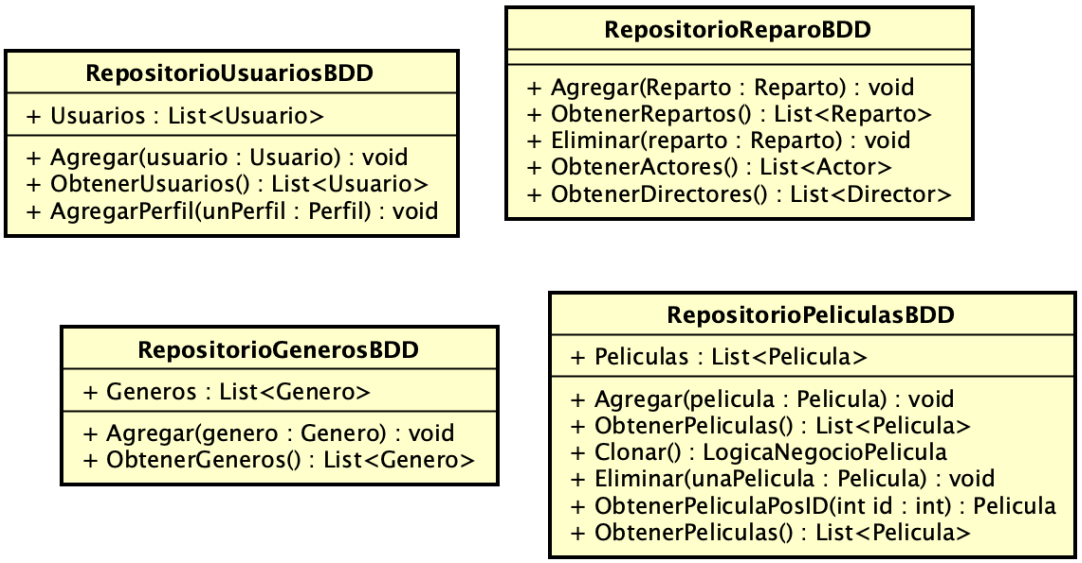


Imagen 2.2: Diagrama de Clases RepositorioEnBDD

Solución de Persistencia

Considerando que el diseño debe contemplar el modelado de una solución con una persistencia adecuada para el problema utilizando Entity Framework (Code First), a continuación, se detallará la solución implementada.

En la siguiente figura, se observa un modelo de tablas de la solución de persistencia, utilizada al desarrollar la aplicación.

Modelo de Tablas

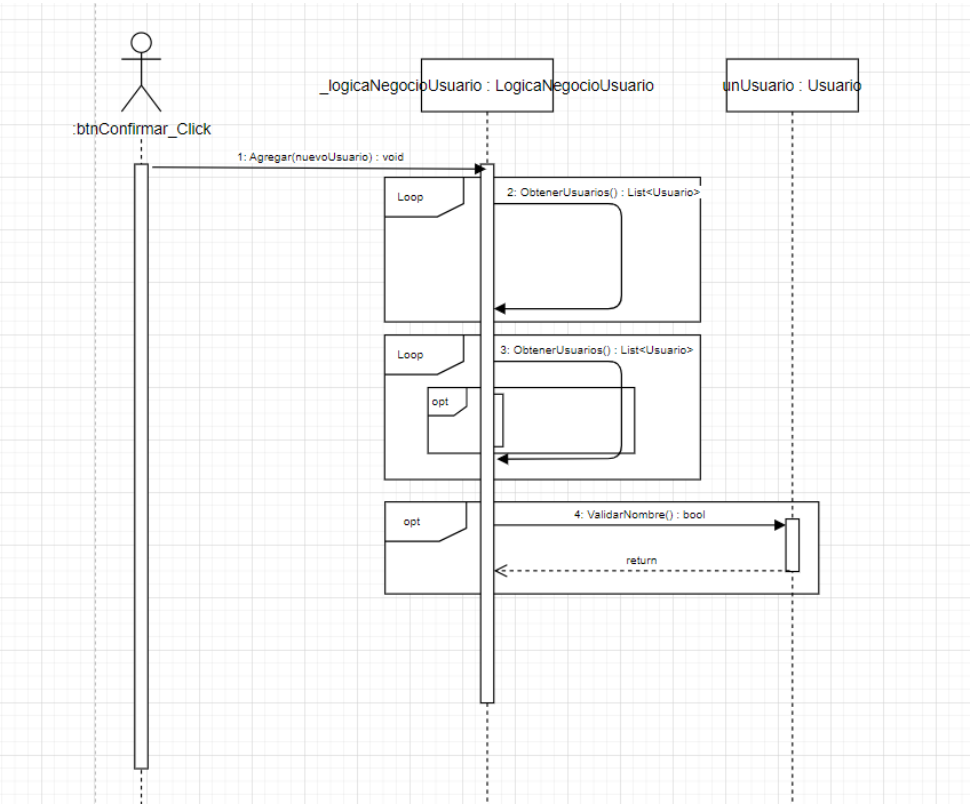
Nuestra solución cuenta con siete tablas, en donde cada una de ellas representa a una entidad del dominio. Cuando nos referimos a tabla nos estamos también refiriendo a la clase en nuestro dominio.

Diagramas de Secuencia

Utilizaremos diagramas de secuencia, diagrama que nos permite modelar la interacción entre objetos en un sistema y plasmar cómo estos intercambian mensajes en un orden determinado, para especificar tres funcionalidades del sistema. Las funcionalidades que consideramos representar son: agregar un usuario, eliminar una película de los perfiles y modificar un director a actor.

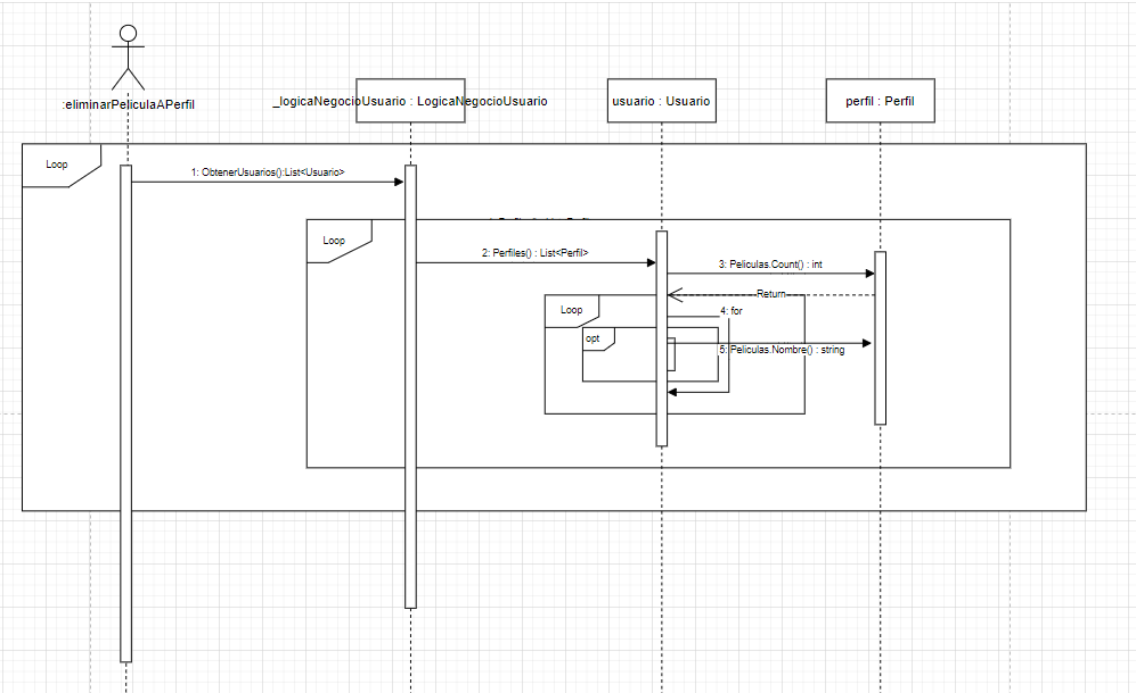
Agregar Usuario

Agregar un usuario comienza a la hora que el usuario entra a la interfaz de registrarse. Una vez verificada que las contraseñas coinciden, se crea un usuario nuevo y se valida. Cuando el usuario ya está validado es que comienza la funcionalidad de agregar usuario. Un objeto de la clase LogicaNegocioUsuario invoca el método Agregar(Usuario usuario). Dentro de LogicaNegocioUsuario se realiza un loop que recorre todos los usuarios y se verifica que el nombre no exista dentro de los usuarios. Luego se realiza otro loop que verifica que el mail no exista dentro de los usuarios y por último se invoca al método ValidarNombre() de la clase Usuario



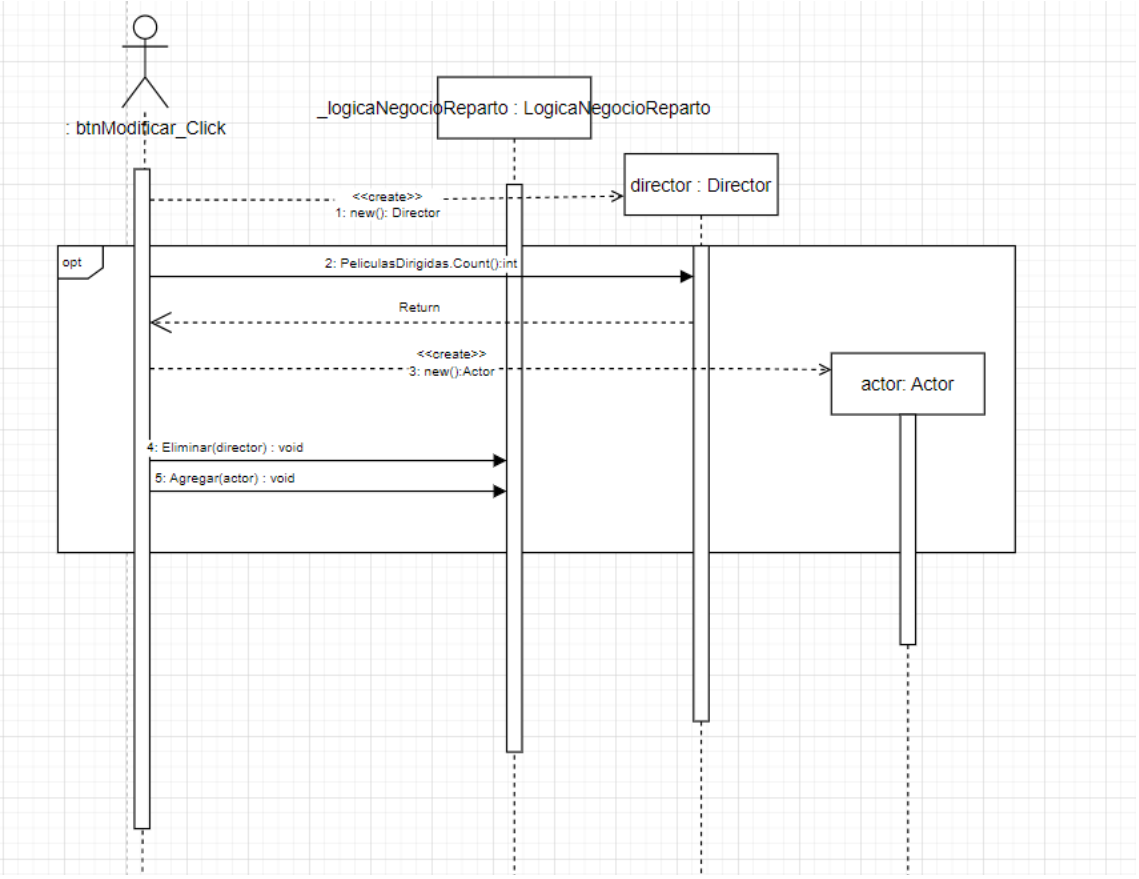
Eliminar Pelicula

Eliminar una película comienza a la hora que el usuario entra a la interfaz de eliminar película. Una vez seleccionada la película y al clickear el botón eliminar, la película debe ser eliminada de todos los perfiles. Para ello se realiza un loop entre todos los usuarios de LogicaNegocioUsuario, luego un segundo loop dentro de los perfiles de cada Usuario y finalmente un último loop que recorre todas las películas de los perfiles y se elimina.



Modificar un Director

El proceso comienza con la selección de un Director, una vez se selecciona el usuario modifica su rol, como información personal del Director y al clickear el botón modificar este cambia de rol a Actor. Para ello se genera un objeto de tipo Director, se verifica que no tenga ninguna película dirigida, de forma contraria se aborta el proceso. Una vez verificado que no tenga ninguna película dirigida se crea un Actor con toda la información del Director. Invocando el método Eliminar de LogicaNegocioReparto se elimina el Director de la Base de Datos y posteriormente se invoca el método Agregar de LogicaNegocioReparto y se agrega el Actor.


: btnModificar_Click
_logicaNegocioReparto : LogicaNegocioReparto
director : Director
<<create>>
1: new(): Director
opt
2: PeliculasDirigidas.Count():int
Return
<<create>>
3: new():Actor
actor: Actor
4: Eliminar(director) : void
5: Agregar(actor) : void

Excepciones

El sistema captura los diferentes errores que se pueden cometer al ingresar información mediante la interfaz gráfica, por lo que se crearon diferentes clases de excepciones. Para manejar campos de información incorrecta, excepciones y mensajes de éxito, decidimos mostrarlos en pantalla con el componente “Message Box”.

De esta forma ayudamos a mantener al usuario siempre informado de lo que está sucediendo. Esta decisión de trabajar con excepciones que están en dominio fue para desligar lo máximo posible la lógica de negocio con la interfaz, ya que si mañana cambiamos la misma, la idea es mantener lo más intacta la lógica.

```
try
{
    if (perfilLogeado.Infantil || perfilLogeado.CoincidePin(Int32.Parse(txtPin.Text)))
    {
        perfilLogeado.Logged = true;
        _formPrincipal.esconderBotonRegistrarPerfil();
        _formPrincipal.MostrarInicio();
    }
    else
    {
        MessageBox.Show("Pin incorrecto");
    }
}
catch (PerfilInvalidoExcepcion ex)
{
    MessageBox.Show(ex.Message);
}
```

Imagen 2.2: Ejemplo manejo de errores

Cobertura de pruebas unitarias

Como ya fue mencionado previamente la aplicación fue desarrollada siguiendo la metodología TDD (desarrollo guiado por pruebas). Esta técnica consiste en escribir antes los casos de prueba que la funcionalidad. Para esto, en primer lugar, se escribe una prueba y se verifica que la nueva prueba falle. Luego, se implementa el código que hace que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito.

Siguiendo esta práctica, logramos conseguir un porcentaje de cobertura de pruebas unitarias muy bueno.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
camilosacchi_CAMILOSACCH...	105	6.31%	1558	93.69%
repositorioenmemoria.dll	0	0.00%	36	100.00%
RepositorioEnMemoria	0	0.00%	36	100.00%
logicanegociotest.dll	11	4.14%	255	95.86%
logicanegocio.dll	12	6.59%	170	93.41%
LogicaNegocio	12	6.59%	170	93.41%
dominiotest.dll	46	6.20%	696	93.80%
dominio.dll	36	8.24%	401	91.76%
Dominio.Excepciones	0	0.00%	6	100.00%
Dominio	36	8.35%	395	91.65%

Imagen 3.1: Evidencia cobertura pruebas unitarias

Como conclusión se puede afirmar que la lógica del código se encuentra estable y verificada, y que funcionará en la mayoría de los casos.

Referencias bibliográficas

[1] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf

[2] R. C. Martin, Clean Code: *A Handbook of Agile Software Craft*, 2008.

ANEXO 1

Errores conocidos

Se entrega una versión con todas las funcionalidades solicitadas funcionando de manera correcta, pero sin persistir en base de datos. Esto no quiere decir que no se puedan persistir en la base de datos. En la base de datos se pueden guardar los Usuarios, Perfiles, Películas y Géneros pero no las funcionalidades no se comportan como deberían. Para comprobar esto se debe ir al Form1, cambiar RepositorioClaseImpl por RepositorioClaseBDD y cambiar las listas por ICollections<Entidad>. Este error ocurre dada el alto acoplamiento y la baja cohesión en el diseño de la solución.