

Universidad ORT Uruguay

Facultad de Ingeniería

https://github.com/ORT-DA1/239770_259432_186779

14/10/2021

Descripción general del trabajo y del sistema	3
Descripción y justificación de diseño	4
Diagrama(s) de paquetes	4
Diagramas de clases	4
Explicación de los mecanismos generales y descripción de las principales decisiones de diseño tomadas	4
Breve análisis de los criterios seguidos para asignar las responsabilidades	5
Cobertura de pruebas unitarias	6
Funcionalidad de alta, baja y modificación de contenidos	7
Anexos	9
ErrorProviders	9
MessageBoxes	10
Diagrama de Paquetes	11
Diagrama de Clases.	12
Cobertura Test	13
Cobertura Clases	13
Contenidos	14
Test de Contenidos	15
Excepciones en Test	16
Como se ven en la aplicación	17
Clase abstracta	18

Descripción general del trabajo y del sistema

(EL USUARIO ADMINISTRADOR ES admin Y LA CONTRASEÑA adminadmin)

El objetivo de este proyecto era el de desarrollar un sistema con capacidades de red social. Para ello, era necesario la existencia de distintos usuarios y funcionalidades que se asemejen a redes como Facebook, Twitter o Instagram. La base de este trabajo era implementar la metodología Test Driven Development (TDD), la cual consiste en realizar primero las pruebas unitarias de las clases para posteriormente construir, aprovechando las herramientas proporcionadas por el IDE (en nuestro caso Visual Studio), las clases que compondrían al sistema.

A su vez, la implementación de Windows Forms y sus User Controls serían los encargados de realizar el trabajo de la interfaz de usuario (UI). Se buscó que el sistema sea lo más estético y eficiente posible, con las herramientas que tenía el grupo. El TDD nos ayudó a reconocer las capacidades del IDE para brindar ayuda a los programadores a la hora de crear proyectos de trabajo.

Otro instrumento fundamental del trabajo fue la metodología GitFlow de Git, la cual nos permitió separar el trabajo ordenadamente, dando un lugar para que cada feature se implemente correctamente. Nos generó varias cosas positivas, pero también algún que otro traspié, ya que algunos archivos, cuya composición desconocemos, se modificaban en algunos commits y/o merges, lo cual causó algunas molestias y descontentos en el trabajo. Desafortunadamente, debido a problemas de tiempo no logramos dejar el sistema como nos hubiese gustado. Creemos que con un poco más de tiempo, y quizás algunas introducciones de los conceptos a implementar con mayor anticipación, se podrían haber logrado mucho mejores resultados. Sin embargo, el grupo aplicó TDD, uno de los conceptos centrales del proyecto, con éxito y se noto su uso y el resultado positivo que este brindó. Se logró cumplir con casi todos los objetivos principales.

Descripción y justificación de diseño

Diagrama(s) de paquetes

En el [diagrama de paquetes](#) se observa que el grupo realizó 3 paquetes principales para la elaboración del proyecto: uno para las clases y funciones lógicas del sistema, otra para la interfaz de usuario, con sus controladores y ventanas, y finalmente una para las pruebas unitarias, claves para la realización del proyecto con la metodología de TDD.

Diagramas de clases

El [diagrama de clases](#) representa todo el paquete de Business Logic. En él, se puede apreciar la utilización de una clase abstracta en “Publicaciones”, que funcionó como base para las clases de “Álbum”, “Estado” y “Escuchando”. Todas estas, como bien lo dice la clase abstracta, son publicaciones que puede realizar el usuario, por lo tanto tomamos como base las características principales, de cuando fue creado y por quien, para elaborar la esta clase.

Explicación de los mecanismos generales y descripción de las principales decisiones de diseño tomadas

El diseño que el grupo eligió seguir fue principalmente basado tanto en previos conocimientos y aplicaciones en otras materias, como también en lo aprendido durante este curso, evidenciado a través de la metodología TDD y la utilización de Windows Forms y User controls, ambos conceptos totalmente nuevos para el grupo en su totalidad.

Se decidió por separar los elementos principales en clases, dentro de los cuales pudimos notar que Album, Estado y Escuchando caían dentro de la temática de publicaciones del usuario, por lo que se decidió crear una [clase abstracta](#) con el fin de tener un mejor manejo de estos elementos importantes para el desarrollo del proyecto.

Luego, una vez realizado el boceto del sistema, el grupo procedió a realizar los tests unitarios, guiados por la metodología TDD, concluyendo con las clases del sistema.

A la hora de realizar la interfaz de usuario, se escogió crear una serie de ventanas del tipo Windows Form para tener las principales funcionalidades del sistema.

Además, el grupo optó por usar, en la medida de lo posible, elementos de Material Design, los cuales se obtuvieron al descargar una extensión en Visual Studio, como fue visto en clase. Algunos componentes nos fueron contraproducentes, ya que no logramos acceder a algunas características, por lo que se manejaron componentes nativos de Windows Forms. Se necesitaba un método para que los usuarios se registren dentro del sistema, y posteriormente inicien sesión. Esto se realizó en una

ventana con ambas opciones como botones, “Iniciar Sesión” y “Registrar Usuario”, las cuales a su vez creaban una ventana para cada una respectivamente. Luego, dentro del inicio de la aplicación, optamos por utilizar los componentes conocidos como User Controls. En un comienzo intentamos evitarlos, porque era una tecnología con la cual no nos sentíamos cómodos, pero luego, motivo de necesidad y de descubrir nuevas herramientas, investigamos acerca de su uso y funcionalidad. Esto dio sus frutos inmediatamente, ya que fue vital para el desarrollo de muchas funcionalidades, como la creación de juegos y de álbumes. Hubiese sido fundamental implementarlos antes, ya que perdimos preciado tiempo intentando mediante otros medios en vano. Creemos que igualmente nos faltó un poco de manejo con ellos, debido a que en ciertas ocasiones optamos por ocultar elementos, en lugar de implementar distintas alternativas que quizás no eran posibles con el conocimiento del grupo.

Para el manejo de errores, se utilizaron elementos suministrados por las Windows Forms, como los [errorProviders](#) y las [Message Boxes](#). Creemos fundamental que el usuario incurra en la menor cantidad de errores o inputs incorrectos posibles, y estos elementos nos ayudaron considerablemente a mostrarle al usuario donde están los errores, así como también una explicación, situación en la que muchas veces nos encontramos como usuarios nosotros mismos, de porque estaba incurriendo en dicho error.

Con el fin de mantener una interfaz prolija y eficiente, se intentó considerablemente separar la interfaz de la lógica. Esto se llevó a cabo mediante dos clases, “Lógica” y “FuncionesLista”, que intentaron encargarse de manejar todos los procesos que eran “pesados” en cuanto a lógica, como también una clase encargada de almacenar todos los usuarios y juegos para no tener que acceder a clases ya sobrecargadas de funciones y atributos.

Breve análisis de los criterios seguidos para asignar las responsabilidades

Las responsabilidades de clases que usamos es que tengan constructores que son los responsables de inicializar los valores de los atributos durante el momento que se crea el objeto.

Luego responsabilizamos a las clases a tener métodos para cambiar los valores de los atributos y los denominamos Set seguidos de otro nombre representativo de que atributo estaban modificando. Otra responsabilidad es saber cual es el valor que tiene el atributo a través de una función Get seguido del valor que queremos obtener.

Todas las funciones tienen una responsabilidad única, o generan un cambio u obtienen un valor.

Cobertura de pruebas unitarias

Las pruebas unitarias son una forma de comprobar que una determinada parte de código funciona correctamente. Las pruebas unitarias se ejecutan de forma automática y se repiten tantas veces como uno quiera. Por lo que las pruebas tienen que ser eficaces y rápidas ya que si no enlentecen el trabajo. Para asegurarnos que los cambios realizados no modifica ningún proceso no deseado fue que optamos por correr todas las pruebas cada vez que modificamos código. Esto nos ayudó a ver los grados de dependencias y a no generar pruebas largas y lentas de ejecutar. Las pruebas también nos ayudaron a la hora de ver que cantidad de código estaba cubierto para saber si estaba todo testeado y funcionando como nosotros deseábamos.

Como podemos ver en la [imagen](#), obtuvimos un 100% de cobertura en el código. Esto quiere decir que el código fue testeado al menos una vez y ya que los resultados de los test son todos correctos ([imagen](#)), podemos decir que el código actúa en función a lo que nosotros queríamos realizar. El tener un 100% de cobertura y que los test han sido todos positivos no nos alcanzó para asegurarnos que el funcionamiento era el correcto por lo que hicimos más cantidad de pruebas para verificar si realmente se cubrían los casos bordes.

En cuanto a la cobertura en las clases de los test no es algo importante ya que sabemos que se lanzan y se capturan excepciones(por lo que no se finalizara de recorrer el test) y por que lo que nos interesa que se testee es el código “funcional”.

Cada clase creada se generó posteriormente a la creación de su test el cual lleva el nombre de la clase seguido de la palabra “Test”. Esta práctica consta en crear pruebas, luego escribir el código para que pase la misma y por último refactorizar el código.

Esto nos fue útil ya que no se nos acumuló nunca código que funcionaba mal ni tuvimos que cambiar las funciones ya que cuando finalizamos de escribirlas sabíamos si cumplían la función deseada o no. Hay test muy simples como get y set de distintos atributos y luego otros más complejos que buscan valores en una lista o dan listas ordenadas entre otros. [Ver imágenes de test](#)

Funcionalidad de alta, baja y modificación de contenidos

Estado:

Su constructor pide como parámetros la persona que crea este tipo de publicación y el texto que quiere asignarle. Este, no puede superar los 260 caracteres, debe contener más de 10 y no puede ser nulo. En caso que alguno de los requisitos no se cumpla se lanza una excepción de 'PublicacionException' o de 'ArgumentNullException'. Luego de creado el Estado, este se agrega a la lista de estados que el usuario tiene como atributo del cual se elimina en caso de que el usuario lo desee. El mismo no brinda la opción de ser modificado por lo que se puede crear o eliminar. También posee una funcionalidad que al convertir el objeto Estado a texto (toString) que devuelve un texto para mostrar. ([Imagen](#))

Escuchando:

El constructor pide como parámetros la persona que crea este tipo de publicación, el nombre, el artista y el álbum de la canción que está escuchando el usuario. Esta clase no tiene ninguna excepción. Luego de creado el Escuchando, este se agrega a la lista de escuchadas que el usuario tiene como atributo del cual se elimina en caso de que el usuario lo desee. El mismo no brinda la opción de ser modificado por lo que se puede crear o eliminar. También posee una funcionalidad que al convertir el objeto Escuchando a texto (toString) que devuelve un texto para mostrar. ([Imagen](#))

Álbum:

Su constructor pide como parámetros la persona que crea este tipo de publicación y el nombre que quiere asignarle. En álbum se pueden agregar hasta un máximo de 10 imágenes que pueden eliminarse en cualquier momento. En caso que se agregue una 11 imagen se lanza una excepción de '[PublicacionException](#)' pero nunca se llevará a cabo ya que no está la opción. Luego de creado el Álbum, este se agrega a la lista de álbumes que el usuario tiene como atributo del cual se elimina en caso de que el usuario lo desee. Este puede ser modificado agregando o quitando imágenes. También posee una funcionalidad que al convertir el objeto Estado a texto (toString) devuelve un texto para mostrar. ([Imagen](#))

Jugadas:

Se generan a partir de la clase Juego, ya que es quien tiene un atributo que es una lista de Jugadas. Cada jugada tiene una fecha de creación, un usuario que es quien jugó, un puntaje y el nombre del juego. A su vez la clase Jugada posee 2 funciones una para compararse con otra y luego la función toString.

En los test fueron probadas las excepciones y las funciones de cada clase, como vemos en la [imagenes](#).

Anexos

ErrorProviders

VentanaRegistroUsuario

VOLVER



ELEGIR FOTO

Nombre de usuario

Contraseña

Nombre

Apellido

Fecha de Nacimiento:

jueves , 14 de octubre de 2021

Direccion:

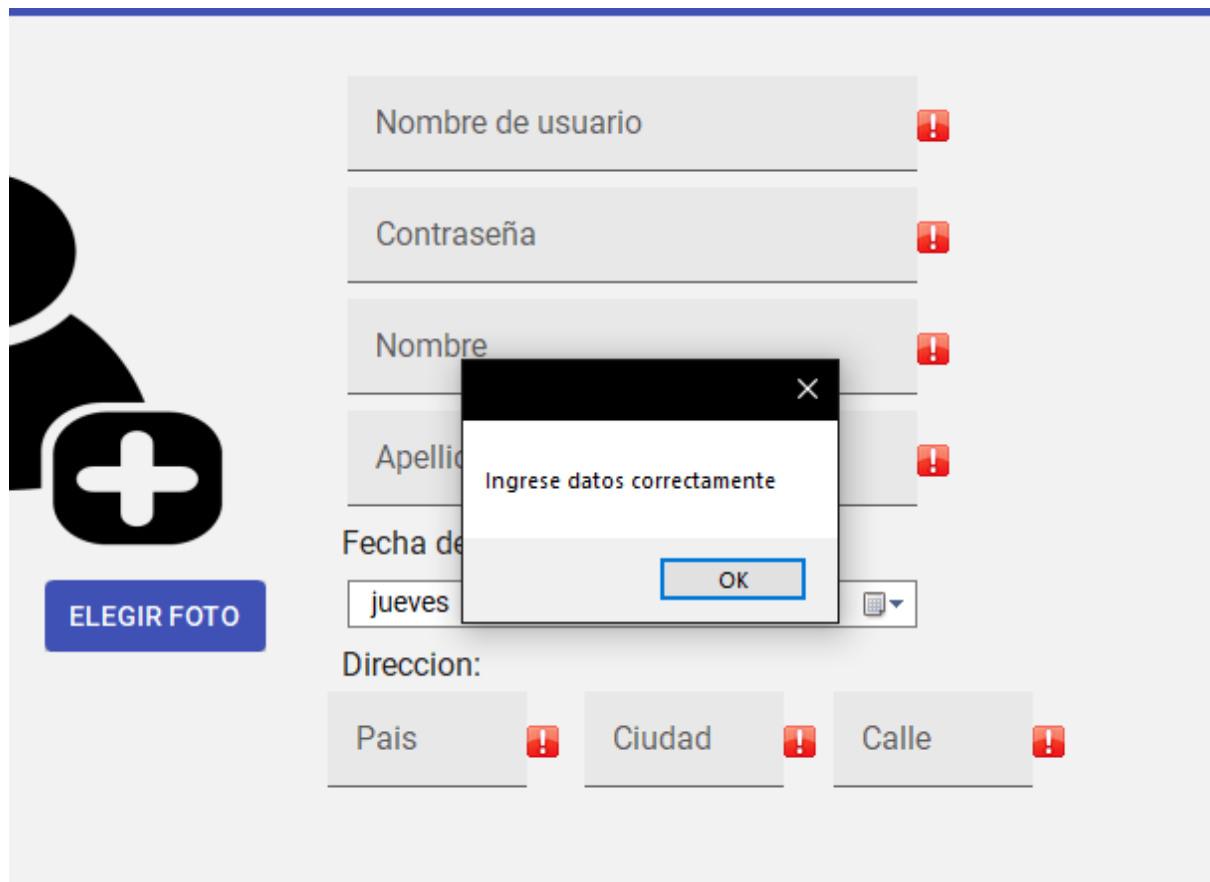
Pais

Ciudad

Calle

GUARDAR

MessageBoxes



The image shows a user registration form with several input fields. Each field has a red exclamation mark icon to its right, indicating a validation error. The fields are: 'Nombre de usuario', 'Contraseña', 'Nombre', 'Apellido', 'Fecha de nacimiento' (with 'jueves' entered), 'Direccion:' (a sub-header for three fields: 'Pais', 'Ciudad', and 'Calle'), and 'ELEGIR FOTO' (a button). A modal dialog box is open in the center, displaying the message 'Ingrese datos correctamente' and an 'OK' button.

Nombre de usuario !

Contraseña !

Nombre !

Apellido !

Fecha de nacimiento jueves !

Direccion:

Pais ! Ciudad ! Calle !

ELEGIR FOTO

Ingrese datos correctamente

OK

Diagrama de Paquetes

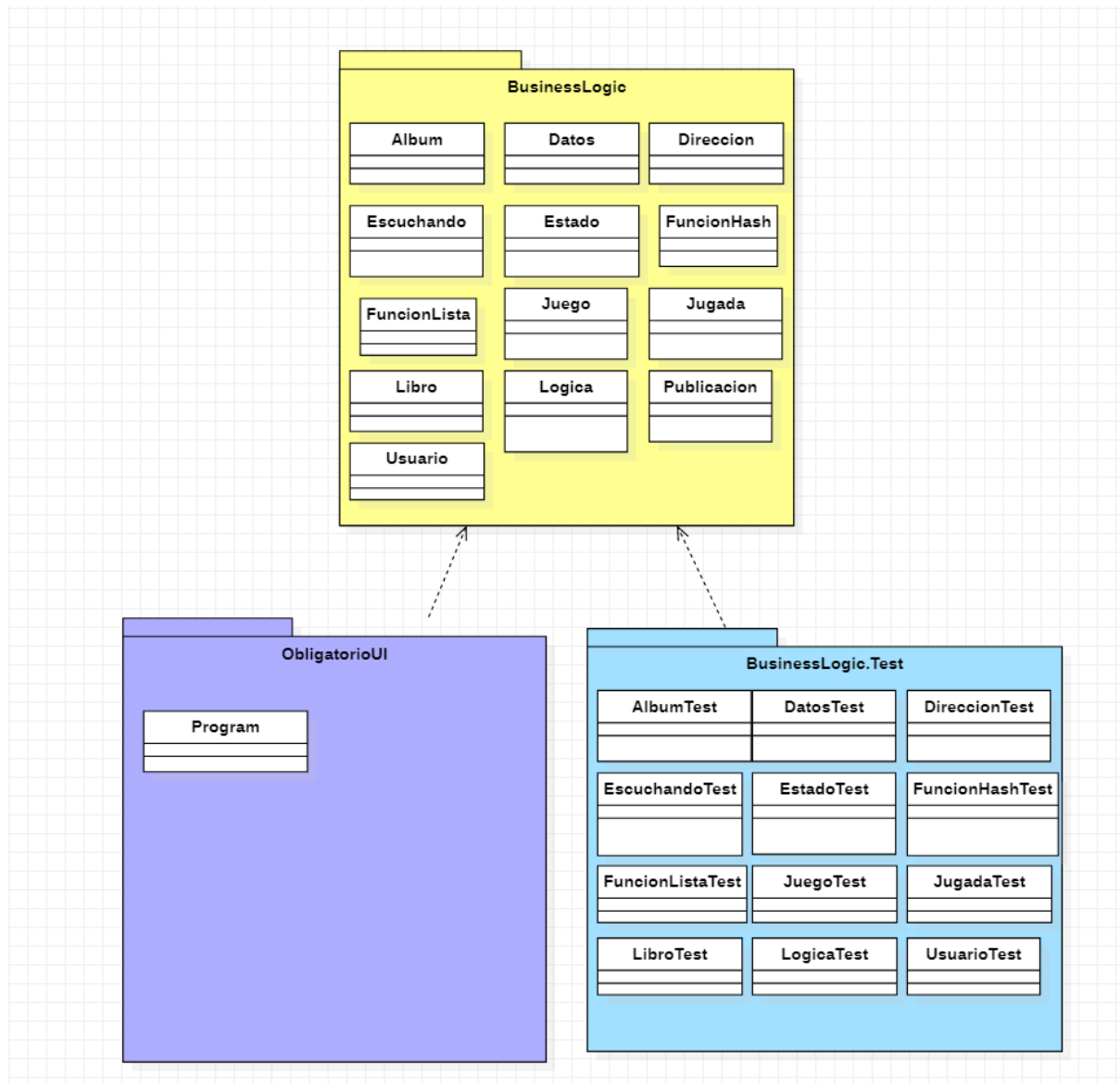
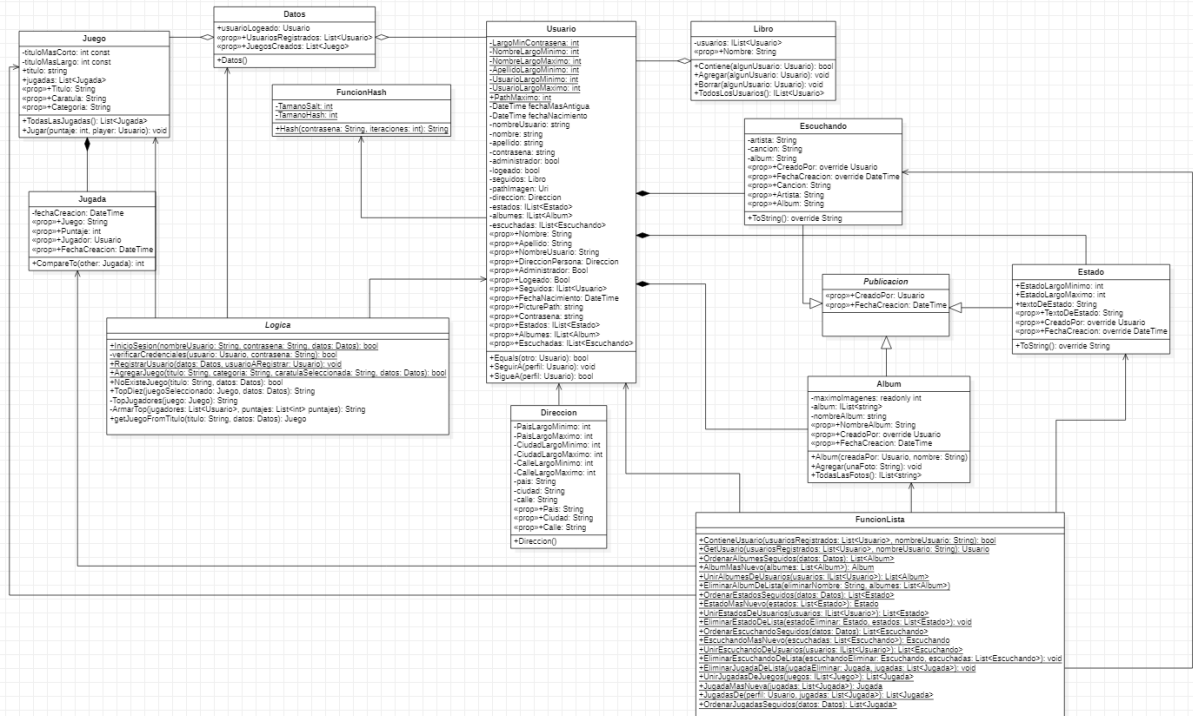


Diagrama de Clases.



Cobertura Test

— Dominio.Test	922	27	949	1675	97.1%	<div><div></div></div>
AlbumTest	55	1	56	106	98.2%	<div><div></div></div>
DatosTest	58	0	58	88	100%	<div><div></div></div>
DireccionTest	44	9	53	120	83%	<div><div></div></div>
EscuchandoTest	96	0	96	174	100%	<div><div></div></div>
EstadoTest	83	3	86	158	96.5%	<div><div></div></div>
FuncionHashTest	9	1	10	33	90%	<div><div></div></div>
FuncionListaTest	177	0	177	274	100%	<div><div></div></div>
JuegoTest	41	3	44	77	93.1%	<div><div></div></div>
JugadaTest	44	0	44	81	100%	<div><div></div></div>
LibroTest	77	0	77	125	100%	<div><div></div></div>
LogicaTest	68	0	68	108	100%	<div><div></div></div>
UsuarioTest	170	10	180	331	94.4%	<div><div></div></div>

Cobertura Clases

— ObligatorioDisenoDeAplicaciones	570	0	570	994	100%	<div><div></div></div>
Album	29	0	29	61	100%	<div><div></div></div>
Datos	7	0	7	23	100%	<div><div></div></div>
Direccion	24	0	24	59	100%	<div><div></div></div>
DireccionException	3	0	3	12	100%	<div><div></div></div>
DominioException	3	0	3	14	100%	<div><div></div></div>
Escuchando	19	0	19	47	100%	<div><div></div></div>
Estado	18	0	18	44	100%	<div><div></div></div>
FuncionHash	7	0	7	26	100%	<div><div></div></div>
FuncionLista	218	0	218	264	100%	<div><div></div></div>
Juego	26	0	26	52	100%	<div><div></div></div>
JuegoException	3	0	3	13	100%	<div><div></div></div>
Jugada	12	0	12	22	100%	<div><div></div></div>
Libro	17	0	17	38	100%	<div><div></div></div>
Logica	96	0	96	123	100%	<div><div></div></div>
Publicacion	5	0	5	18	100%	<div><div></div></div>
PublicacionException	3	0	3	12	100%	<div><div></div></div>
Usuario	80	0	80	166	100%	<div><div></div></div>

Contenidos

```
public abstract class Publicacion
{
    public Usuario creadoPor;
    public DateTime fechaCreacion;

    3 references
    public Publicacion(Usuario usuarioCreador) {...}
    18 references | 10/10 passing
    public abstract Usuario CreadoPor { get; set; }
    31 references | 7/7 passing
    public abstract DateTime FechaCreacion { get; set; }
}
```

```
public class Estado : Publicacion
{
    private const int EstadoLargoMinimo = 10;
    private const int EstadoLargoMaximo = 260;
    private string textoDeEstado;

    7 references | 1/1 passing
    public Estado(Usuario creadaPor, string texto) {...}
    8 references | 7/7 passing
    public string TextoDeEstado {...}
    5 references | 3/3 passing
    public override Usuario CreadoPor {...}
    12 references | 4/4 passing
    public override DateTime FechaCreacion {...}
    3 references | 1/1 passing
    public override string ToString() {...}
}
```

```
public class Jugada : IComparable<Jugada>
{
    private DateTime fechaCreacion = new DateTime();
    5 references | 1/1 passing
    public Jugada()
    {
        fechaCreacion = DateTime.Now;
    }
    6 references
    public string Juego { get; set; }
    12 references | 3/3 passing
    public int Puntaje { get; set; }
    7 references | 1/1 passing
    public Usuario Jugador { get; set; }
    1 reference | 1/1 passing
    public int CompareTo(Jugada other) {...}
    7 references
    public DateTime FechaCreacion { get; set; }
}
```

```
public class Album : Publicacion
{
    private readonly int maximoImagenes = 10;
    private IList<string> album;
    private String pathImagen;
    private string nombreAlbum;

    8 references | 1/1 passing
    public Album(Usuario creadaPor, string nombre) {...}
    4 references | 2/2 passing
    public override Usuario CreadoPor {...}
    11 references | 2/2 passing
    public override DateTime FechaCreacion {...}
    4 references | 2/2 passing
    public void Agregar(string unaFoto) {...}
    1 reference | 1/1 passing
    public void Borrar(string unaFoto) {...}
    6 references | 3/3 passing
    public IList<string> TodasLasFotos() {...}
    9 references | 3/3 passing
    public string NombreAlbum {...}
    3 references | 1/1 passing
    public override string ToString() {...}
}
```

```
public class Escuchando : Publicacion
{
    private string artista;
    private string cancion;
    private string album;

    8 references | 1/1 passing
    public Escuchando(Usuario creadaPor,
        string cancionEscuchando,
        string artistaEscuchando,
        string albumEscuchando) {...}
    9 references | 5/5 passing
    public override Usuario CreadoPor {...}
    8 references | 1/1 passing
    public override DateTime FechaCreacion {...}
    6 references | 5/5 passing
    public string Artista {...}
    4 references | 3/3 passing
    public string Cancion {...}
    6 references | 4/4 passing
    public string Album {...}
    1 reference | 1/1 passing
    public override string ToString() {...}
}
```

Test de Contenidos

EscuchandoTest (17)	1 ms
ComparaarCreadorDeEscuchando	1 ms
ComparaarDistintoCreadorDeEscuchando	< 1 ms
CreadorDeEscuchando	< 1 ms
CreadorIncorrectoDeEscuchando	< 1 ms
GetAlbumDeEscuchando	< 1 ms
GetAlbumIncorrectoEscuchando	< 1 ms
GetArtistaEscuchando	< 1 ms
GetArtistaIncorrectoEscuchando	< 1 ms
GetCancionEscuchando	< 1 ms
GetCancionIncorrectoEscuchando	< 1 ms
GetHoraEscuchando	< 1 ms
GetIncorrectoEscuchando	< 1 ms
GetToString	< 1 ms
SetAlbumDeEscuchando	< 1 ms
SetArtistaEscuchando	< 1 ms
SetCancionEscuchando	< 1 ms
SetCreadorDeEscuchando	< 1 ms

AlbumTest (10)	36 ms
Agregar11Imagenes	31 ms
AgregarUnalmagen	1 ms
BorrarUnalmagen	< 1 ms
ConstructorDeAlbum	< 1 ms
GetCantidadElementosInicial	< 1 ms
GetNombreDelAlbum	< 1 ms
GetToString	4 ms
SetCreadoPor	< 1 ms
SetFechaDelAlbum	< 1 ms
SetNombreDelAlbum	< 1 ms

JugadaTest (6)	2 ms
AsignarJugador	< 1 ms
CompararJugada	< 1 ms
JugarTest	< 1 ms
NuevaJugada	< 1 ms
PuntajeDistinto	1 ms
SetPuntaje	1 ms

EstadoTest (14)	22 ms
CompararEstados	1 ms
CreadorDeEstado	< 1 ms
CreadorIncorrectoDeEstado	< 1 ms
EstadoCorto	6 ms
EstadoLargo	4 ms
GetCreadorEstado	< 1 ms
GetFechaDeCreacion	3 ms
GetHoraDeCreacion	< 1 ms
GetTextoEstado	< 1 ms
GetToString	< 1 ms
SetHoraDeCreacion	< 1 ms
SetTextoEstado	< 1 ms
SetTextoNull	7 ms
TextoincorrectoDeEstado	1 ms

Excepciones en Test

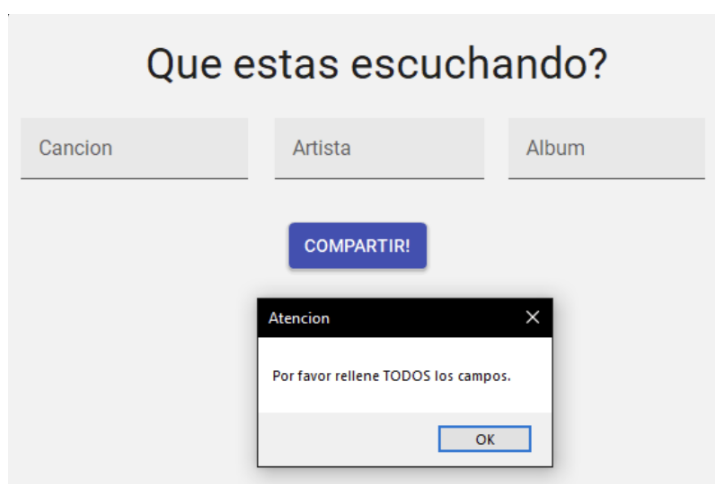
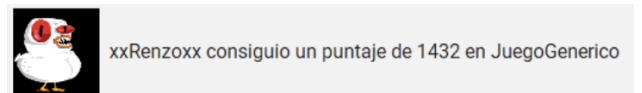
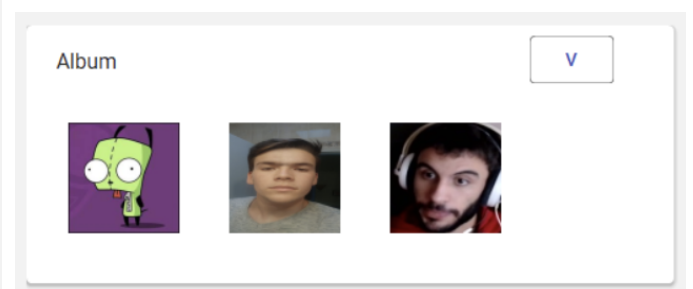
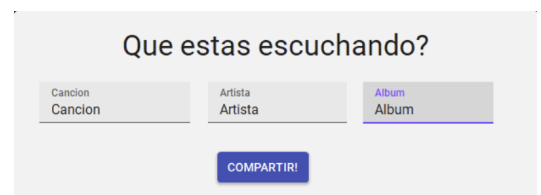
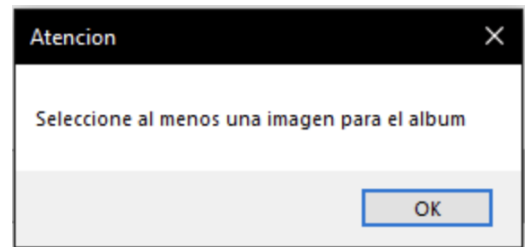
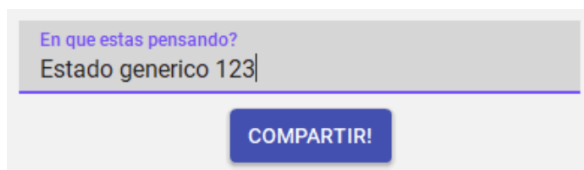
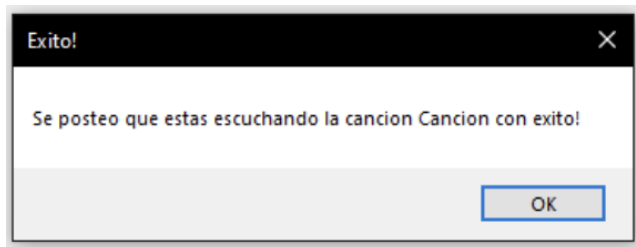
```
[TestMethod]
[ExpectedException(typeof(ArgumentNullException))]
✓ | 0 references
public void SetTextoNull()
{
    EstadoFulano.TextoDeEstado = null;
}
```

```
[TestMethod]
[ExpectedException(typeof(PublicacionException))]
✓ | 0 references
public void Agregar11Imagenes()
{
    for (int i = 0; i < 11; i++)
    {
        albumDeFulano.Agregar(@"Resources\test-image"+i+".jpg");
    }
}
```

```
[TestMethod]
[ExpectedException(typeof(PublicacionException))]
✓ | 0 references
public void EstadoLargo()
{
    EstadoFulano.TextoDeEstado = GenerarStringLargo(textoFulano, EstadoLargoMaximo);
}
```

```
[TestMethod]
[ExpectedException(typeof(PublicacionException))]
✓ | 0 references
public void EstadoCorto()
{
    EstadoFulano.TextoDeEstado = "M";
}
```


Como se ven en la aplicación



Clase abstracta

