

Universidad ORT Uruguay  
Facultad de Ingeniería

## Obligatorio Diseño de aplicaciones 1

Entregado como requisito para la obtención del puntaje para la  
materia Diseño de Aplicaciones 1

Joaquín Bonifacino – 243193  
Rafael Cadenas – 269150  
Tomás Caetano– 189213

Tutor: Guillermo Germán Areosa Suárez

2022

[Enlace a GIT](#)

## 1. Declaración de Autoría:

Nosotros, Joaquín Bonifacino, Rafael Cadenas y Tomás Caetano, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras trabajamos en el proyecto.
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad.
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra.
- En la obra, hemos acusado recibo de las ayudas recibidas
- Cuando la obra se basa en trabajo realizado juntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros.
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



---

Firma

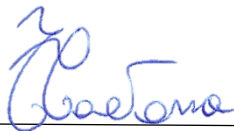
Joaquín Bonifacino  
Aclaración de firma



---

Firma

Rafael Cadenas  
Aclaración de firma



---

Firma

Tomás Caetano  
Aclaración de firma

## 2. Índice

1.	Declaración de Autoría: .....	2
2.	Índice.....	3
3.	Descripción general del Sistema .....	4
3.1.	Diagrama de paquetes .....	5
3.2.	Diagrama de clases.....	6
3.2.1.	Domain (Dominio).....	6
3.2.2.	Data (Repositorios) .....	8
3.2.3.	Logic (Lógica de negocio) .....	9
3.3.	Mecanismos generales.....	9
3.4.	Decisiones de diseño y su porque.....	10
3.4.1.	Decisiones de backend.....	10
3.5.	Criterios a la hora de asignar responsabilidades .....	11
4.	Cobertura de pruebas unitarias.....	11
5.	Informe de la funcionalidad del control de contenido PG .....	12
6.	Informe de funcionalidad de administración de movies y PG.....	12
7.	Evidencia de la ejecución de los casos de prueba .....	12
8.	Mejoras para segunda release.....	13
8.1.1.	Mejorar la robustez.....	13
8.1.2.	Mejora de diseño .....	13

### 3. Descripción general del Sistema

El sistema consiste en una plataforma de streaming de movies, la cual no tiene la funcionalidad de reproducción, dicha funcionalidad fue reemplazada con un botón para marcar como vista la movie.

El sistema permite alta de accounts y profiles de dichas accounts (máximo 4 profiles por account). El primer profile creado en una account se vuelve automáticamente el owner de la account el cual tiene el permiso de borrar y marcar como profile de niño a los demás profiles, para estas operaciones se solicita nuevamente la contraseña de la account, con el fin de proteger que los niños y no dueños de la cuenta borren o modifiquen atributos de sus profiles.

También cuenta con un profile administrador, el cual, por medio de un menú ajustes, puede agregar y borrar genre (solo se puede borrar un genre si ninguna movie lo posee como genre principal o subgenre), movies y seleccionar el ordenamiento de las películas en el catálogo, se hace una mayor descripción del flujo en el punto 6.

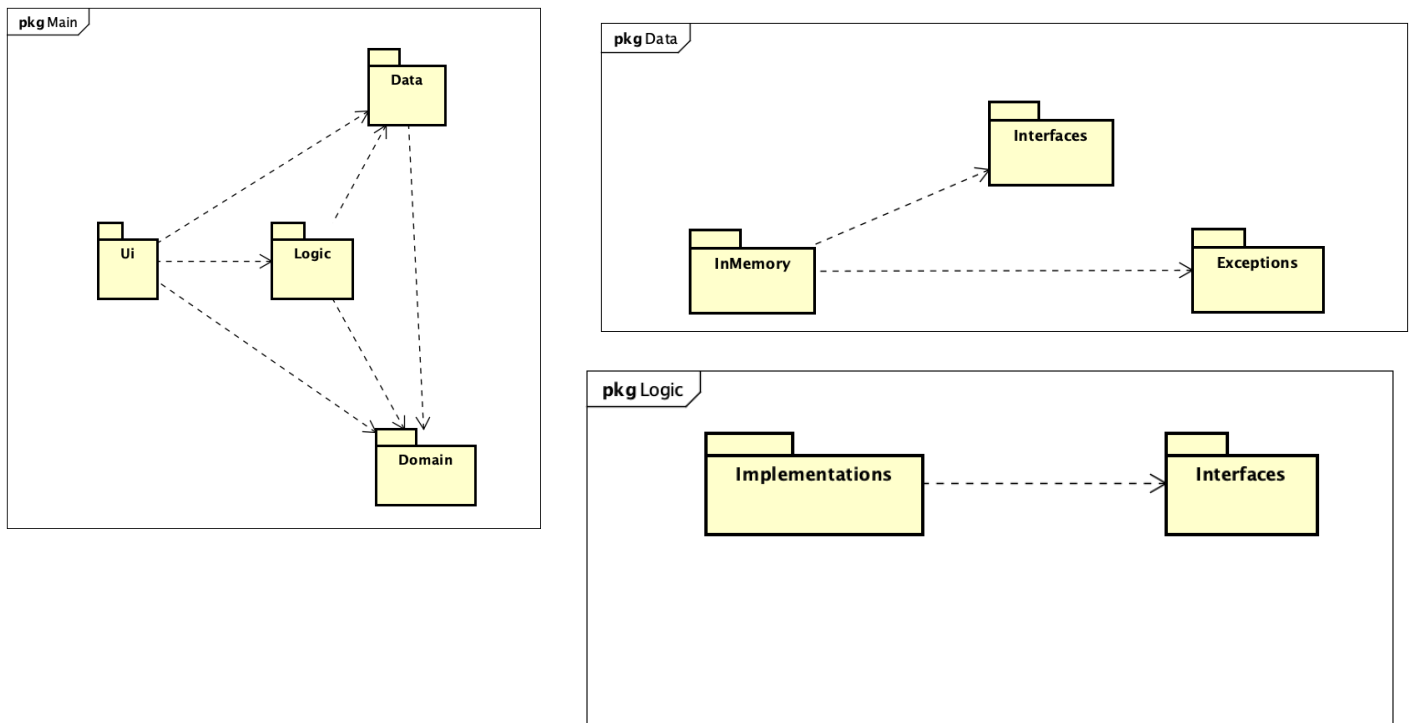
A efectos de esta entrega, la account admin tiene como nombre de usuario “adaAdminTest” y contraseña “adaAdminTest”, por último, el pin del perfil es 11111.

Hay 3 criterios disponibles de ordenación para el catálogo de movies, los cuales son:

1. **Por nombre:** Ordenan alfabéticamente las movies por nombre del genre. En caso de empate, ordena por nombre de movie (esto aplica a los demás criterios).
2. **Por score:** Ordena en base a los scores por genre del profile logeado.
3. **Por patrocinio:** Ordena las movies por si son patrocinadas o no.

Por último, las movies tienen la opción de dar dislike (-1 punto), like (+1) y superlike (++1) a través de botones, like y dislike suben y restan respectivamente 1 punto al genre principal, y superlike sube 2 puntos al genre principal y 1 punto a los subgenres.

### 3.1. Diagrama de paquetes



En total hay **3 diagramas**, el principal (main) consiste en 4 paquetes:

1. Domain: Es la que contiene las librerías de clases usadas en la lógica de negocio (Logic), los repositorios (Data) e interfaz gráfica. Todos los demás paquetes dependen de Domain.
2. Logic: Presenta archivos que manejan la lógica de negocio, en otras palabras, métodos para funcionalidades de la aplicación e interacción con repositorios. Este paquete depende de Data, ya que requiere sus repositorios para ejecutar. UI es el único paquete que depende de Logic.
3. UI: Está compuesto por la interfaz gráfica, por ende, es el paquete que tiene el archivo ejecutable. Ningún paquete depende de UI, sin embargo, UI depende de los demás paquetes, esto es esperable ya que en UI es la integración del front end con el back end. En cuanto a la dependencia de UI a Data solo se da por la creación
4. Data: Es el encargado del guardado en memoria local de todas las instancias de las clases de Domain. Por esto último, Data depende de Domain. Cabe destacar que UI no depende de este paquete ya que no es correcto ni necesario que UI acceda a los repositorios directamente, si requiere datos, los obtiene a través de los métodos implementados en el paquete Logic.

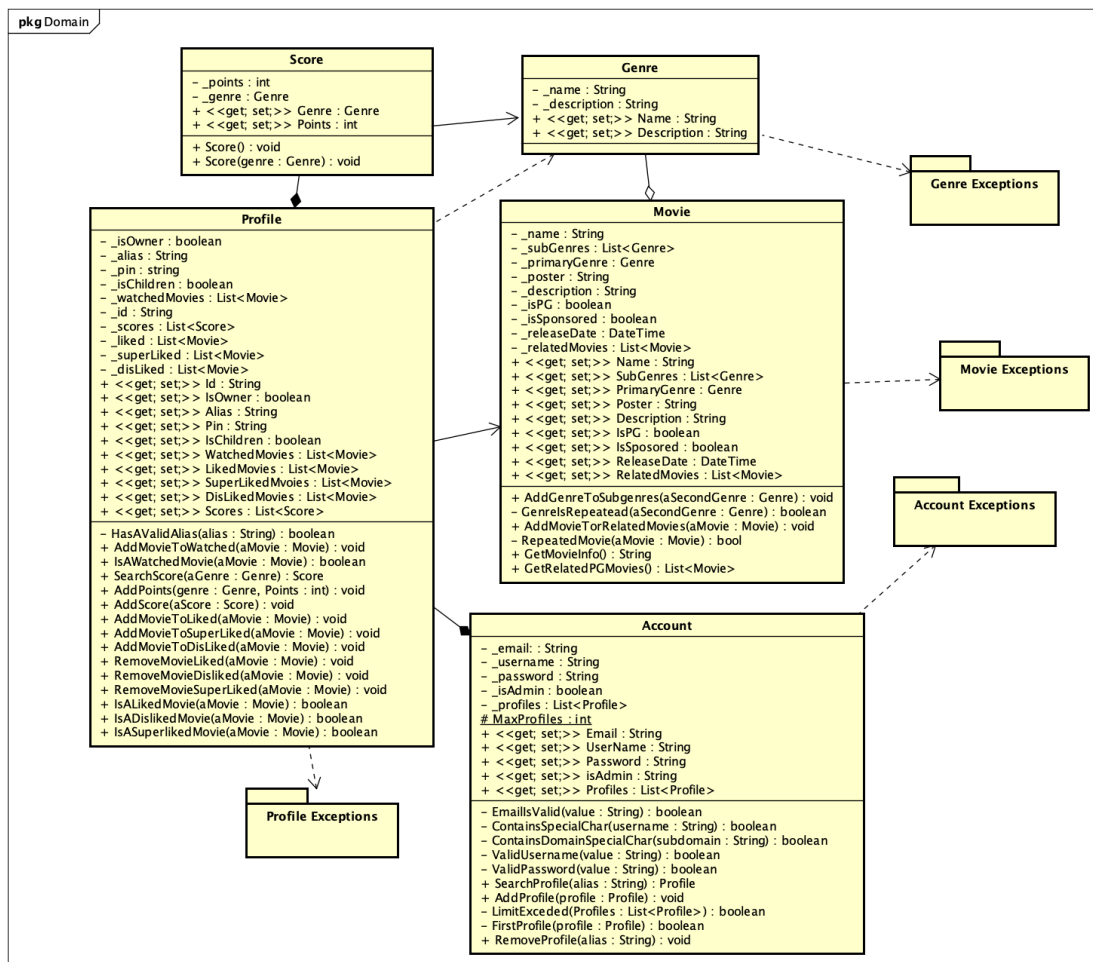
## 3.2. Diagrama de clases

### 3.2.1. Domain (Dominio)

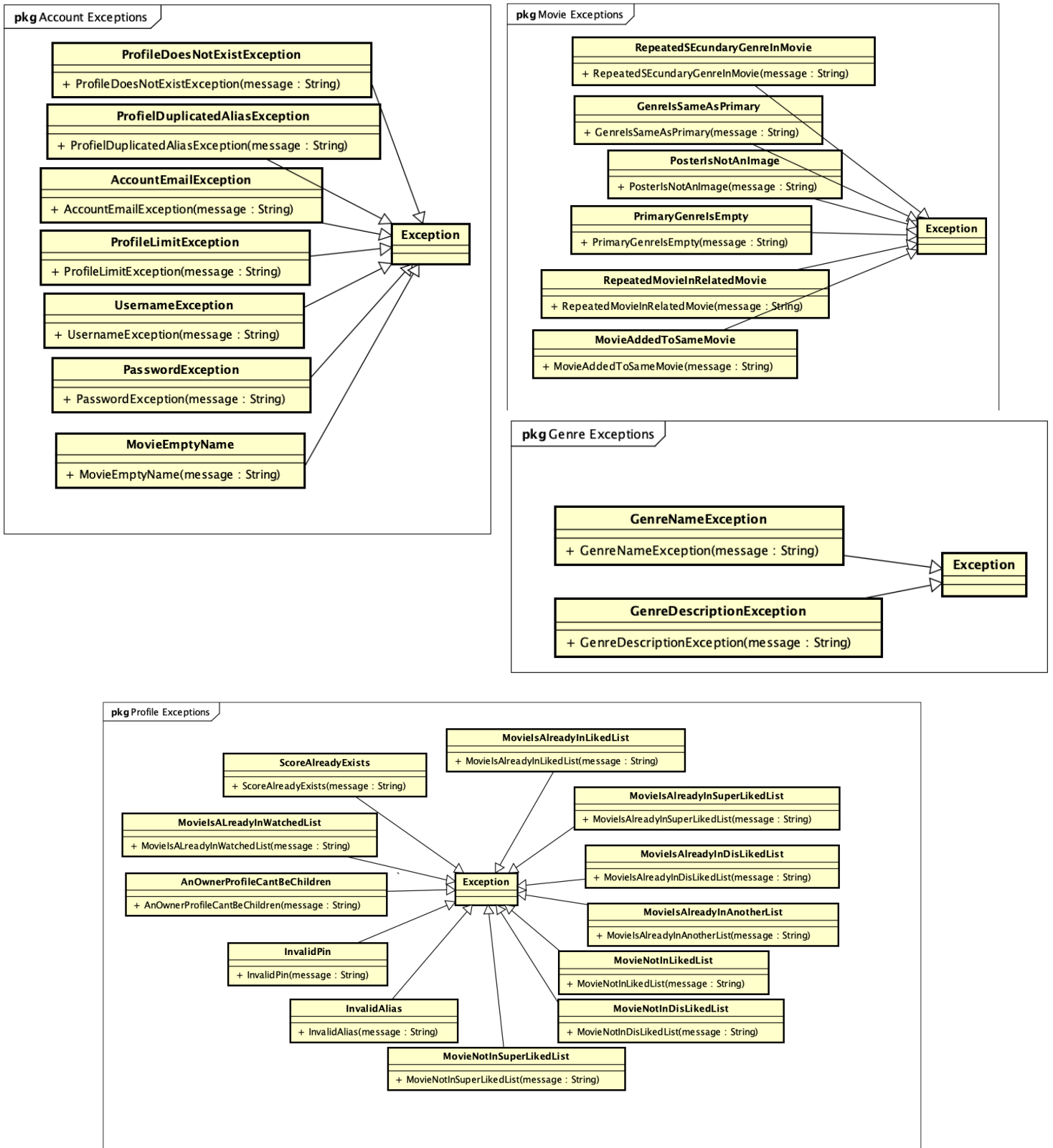
Como se puede apreciar en el diagrama, el dominio comprende 5 clases con 4 carpetas de exceptions para Profile, Genre, Account y Movie. Podemos ver una composición dentro de otra en las clases Account, Profile y Score, donde Score es componente de Profile y este último es componente de Account, es decir, un Score no existe si no hay un Profile que lo contenga, y a su vez no existe un Profile si no hay un Account que lo contenga, además de que un Account no puede tener más de 4 Profiles.

Luego el diagrama muestra una agregación entre Movie y Genre, donde hay una dependencia que va de 1 a n genre en los atributos PrimaryGenre(NotNull) y SubGenres(Puede ser vacía).

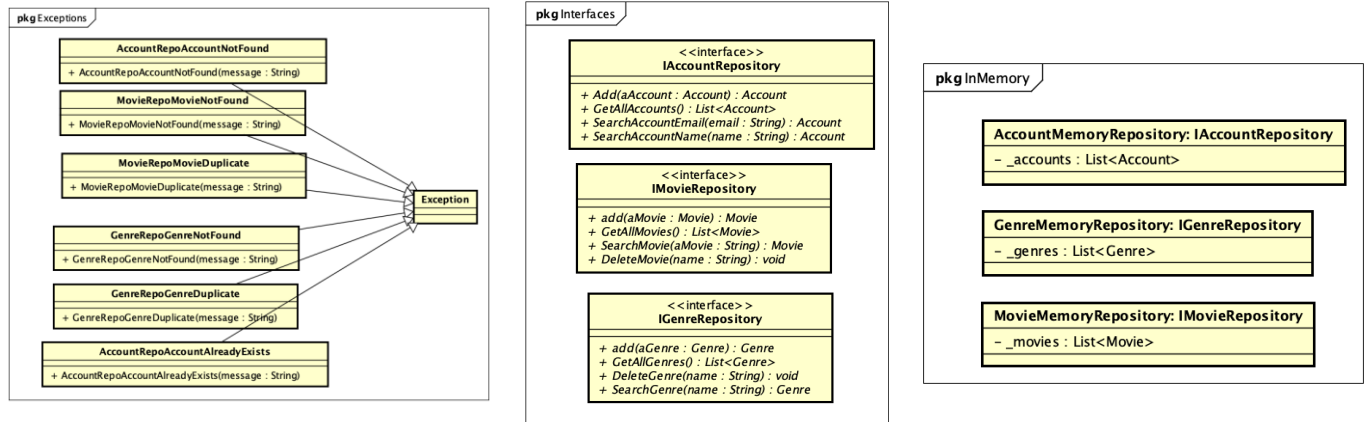
Justificamos que la propiedad "isOwner" de Profile y "isAdmin" de Account sean valores booleanos ya que no presentan un comportamiento distinto desde la logica, su única diferencia es para mostrar o ocultar ítems en la UI



A continuación se muestran las excepciones personalizadas para cada paquete:



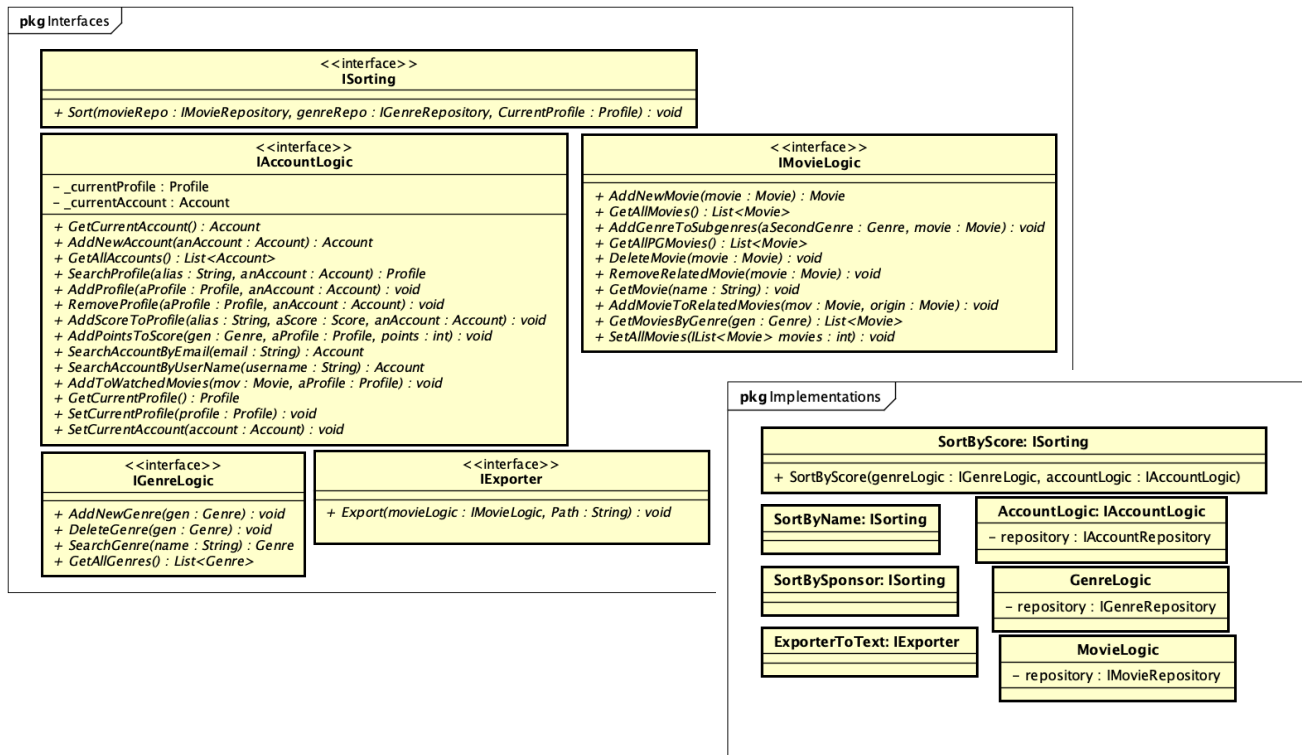
### 3.2.2. Data (Repositorios)



El paquete de data posee las interfaces, implementaciones y excepciones de los tres almacenamientos de datos de la aplicación. Ya que las implementaciones solo tienen los métodos requeridos por su respectiva interfaz, son implícitos los métodos en su respectiva implementación.



### 3.2.3. Logic (Lógica de negocio)



Dentro del paquete de Logic se toman en cuenta 5 interfaces, donde solo 1 (ISorting) posee más de 1 implementación, las demás poseen una implementación donde únicamente están presentes los métodos requeridos. Además, las implementaciones de AccountLogic, MovieLogic y GenreLogic, tienen como atributo su respectivo repositorio en el cual consulta información y realiza operaciones con la misma.

### 3.3. Mecanismos generales

La interfaz gráfica posee 4 atributos, 3 son interfaces de lógica de negocio de accounts, genre y movie, y la restante es el sorter. La interfaz interactúa con el almacenamiento a través de las interfaces de lógica correspondientes, ya que son las encargadas de validar la información y modificar el almacenamiento interno.

Dentro de las funcionalidades de alta de account, logeo, creación de perfiles, seteo de si un profile es niño, eliminación de un profile de una account, dar like, dislike y superlike, y marcar como vista una movie se realizan a través de la invocación de los métodos de la interfaz de la lógica de account (IAccountLogic).

Por otro lado, la creación y borrado de las movies se manejan por medio de la lógica de movie (IMovieLogic). Por último, la creación y borrado (con su respectiva validación) es posible gracias a la lógica de genre (IGenreLogic).

### 3.4. Decisiones de diseño y su porque

#### 3.4.1. Decisiones de backend

##### 3.4.1.1. Paquete Domain

La decisión más importante fue la creación de la clase Score, de esta manera el manejo de los puntajes por profile se resume en una lista de instancias de Score, que únicamente tienen los puntos y el genre asociado. Previamente la idea original era un array doble dentro de la clase Profile, esto fue desechado por su complejidad.

Sucedió algo similar con el manejo de likes, dislikes y superlikes, previamente no se vio directamente la necesidad de listas de movies a las cuales la instancia les haya dado dislike, like o superlike.

Al momento de la integración con interfaz gráfica de la funcionalidad de likes, se intuyó que un profile no puede dar like a la misma movie más de una vez, por ende, fue necesaria la creación de las listas dentro de la clase Profile. De esta forma se resolvió de manera simple el manejo de likes a través de la interfaz gráfica.

##### 3.4.1.2. Paquete Logic

Se determinó que la interfaz encargada de la mayoría de las funcionalidades sería IAccountLogic con su respectiva implementación, ya que, al ser una composición de una composición, el compuesto mayor sería el principal en su lógica, es decir, ya que la única forma de que se instancie Score es que este dentro de una instancia de Profile, y para que un Profile exista tiene que estar dentro de una instancia de Account.

Luego el manejo y validación de movies seria delegada a otra interfaz la cual es IMovieLogic con su respectiva implementación. Previamente ésta se encargaba también del manejo de Genre, pero eso generaría acoplamiento innecesario, por esta razón, se decidió modularizar IMovieLogic, resultando en la creación e implementación de IGenreLogic, que solo se encarga de la creación, consultas y borrado de los datos del repositorio de genre.

La decisión de manejar el sorting a través de una interfaz facilitó el poder cambiar el tipo de sort. Gracias al polimorfismo, con solo cambiar el tipo de implementación en una instancia de la interfaz, cambiaba totalmente el cómo el método sort funcionaba, permitiendo llamar siempre a el método "Sort" en todos los casos evitando código extra en el frontend.

Lo mismo sucede con la interfaz IExporter, la idea de su diseño es el poder exportar en diferentes formatos a futuro, con tan solo cambiar el tipo de implementación y llamar al mismo metodo "Export".

Cabe destacar que el uso de interfaces para la lógica de negocio va a permitir el poder modificar el cómo funcionan las implementaciones sin necesidad de modificar los archivos en los cuales estas implementaciones son utilizadas.

#### 3.4.1.3. Paquete Data

Los datos de la aplicación actualmente se manejan en la memoria interna, pero eso no implica que existan diversas maneras de guardar datos. Con esta idea en mente, se usó del polimorfismo de los repositorios de Movie, Genre y Account, es clave para poder implementar las demás formas de guardado, si es solicitado.

Los repositorios de Score y Profile no existen ya que estos solo son clases componentes.

El polimorfismo de Data permite el poder mejorar la aplicación al conectarlo, por ejemplo, a una base de datos o a una API que maneje los datos.

### 3.5. Criterios a la hora de asignar responsabilidades

Definimos clases con una buena cohesión, donde cada una se encarga de su objeto en sí, a lo sumo almacenan otros objetos, pero no se encargan de modificarlos, sino que cada objeto solo modifica sus propios atributos y retorna valores que le competen a su clase.

Un ejemplo es el de Account con Profile. Account solo se encarga de almacenar el Profile en una lista, pero es la clase Profile la que se encarga de manejar las instancias de Score, sus atributos y operaciones sobre ellos mismos.

El paquete de UI solo se encarga de las ventanas de interfaz gráfica y sus comportamientos, utilizando las diferentes lógicas para las operaciones. Un ejemplo es el de AddPointsToScore, este método recibe un género y un puntaje, dejando la responsabilidad a otros módulos de crear o no el objeto score (dependiendo de si ya existe en ese Profile o no). Esto permite trabajar de manera asíncrona con los integrantes del equipo, donde quien desarrollaba la UI solo necesitaba saber que método ejecutar, confiando en que el backend realizaba su tarea correspondiente (sin tener que saber cómo).

## 4. Cobertura de pruebas unitarias

Se realizaron 174 pruebas unitarias para todas las clases del dominio, lógica y data. Alcanzando el 100% de cobertura ya que se testearon todas las líneas del código de dichas

clases. Esto no significa que existan casos particulares donde haya errores, pero creemos haber cubierto los casos más generales para las clases.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
└─ Tomas_PC-CASA 2022-10-07 18_...	64	2.39%	2618	97.61%
└─ data.dll	0	0.00%	99	100.00%
└─ domain.dll	0	0.00%	637	100.00%
└─ logic.dll	0	0.00%	246	100.00%
└─ tests.dll	64	3.76%	1636	96.24%

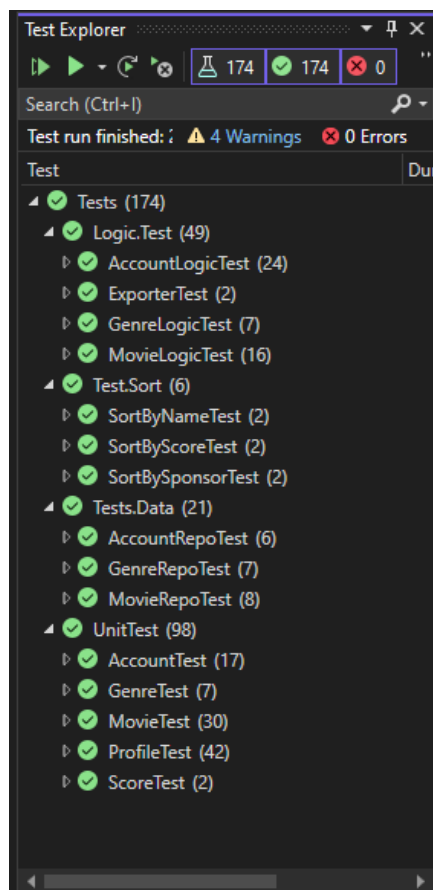
## 5. Informe de la funcionalidad del control de contenido PG

Se optó por verificar si el perfil está marcado como perfil de niño, la devolución de la lista de películas se filtre por si es PG o no, esto también se aplica a las películas relacionadas, devolviendo una vista de las películas relacionadas que sean PG. Todo es posible a través de queries de linq aplicadas a los repositorios implicados.

## 6. Informe de funcionalidad de administración de movies y PG

[Link to youtube video](#)

## 7. Evidencia de la ejecución de los casos de prueba



## 8. Mejoras para segunda release

### 8.1.1. Mejorar la robustez

1. En las funciones que son utilizadas por el administrador (como addMovie o addGenre) seria buena practica verificar que solo sean utilizadas por una account del tipo admin, mas alla de que la UI directamente no le de la posibilidad.
2. Refactoring de algunas de las funciones de la UI

### 8.1.2. Mejora de diseño

1. Se podria utilizar un Id unico en las peliculas para que las mismas puedan repetir su nombre como ocurre con los remakes, extended editions, reboot, etc.