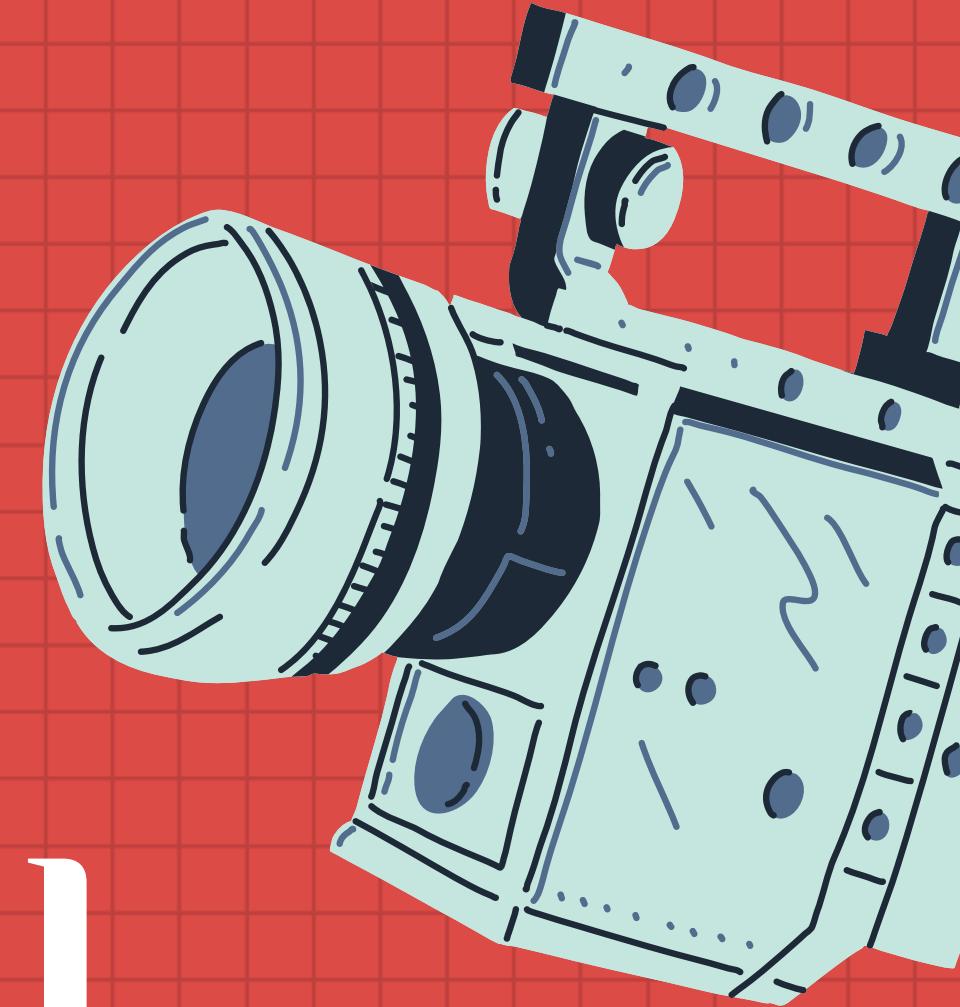
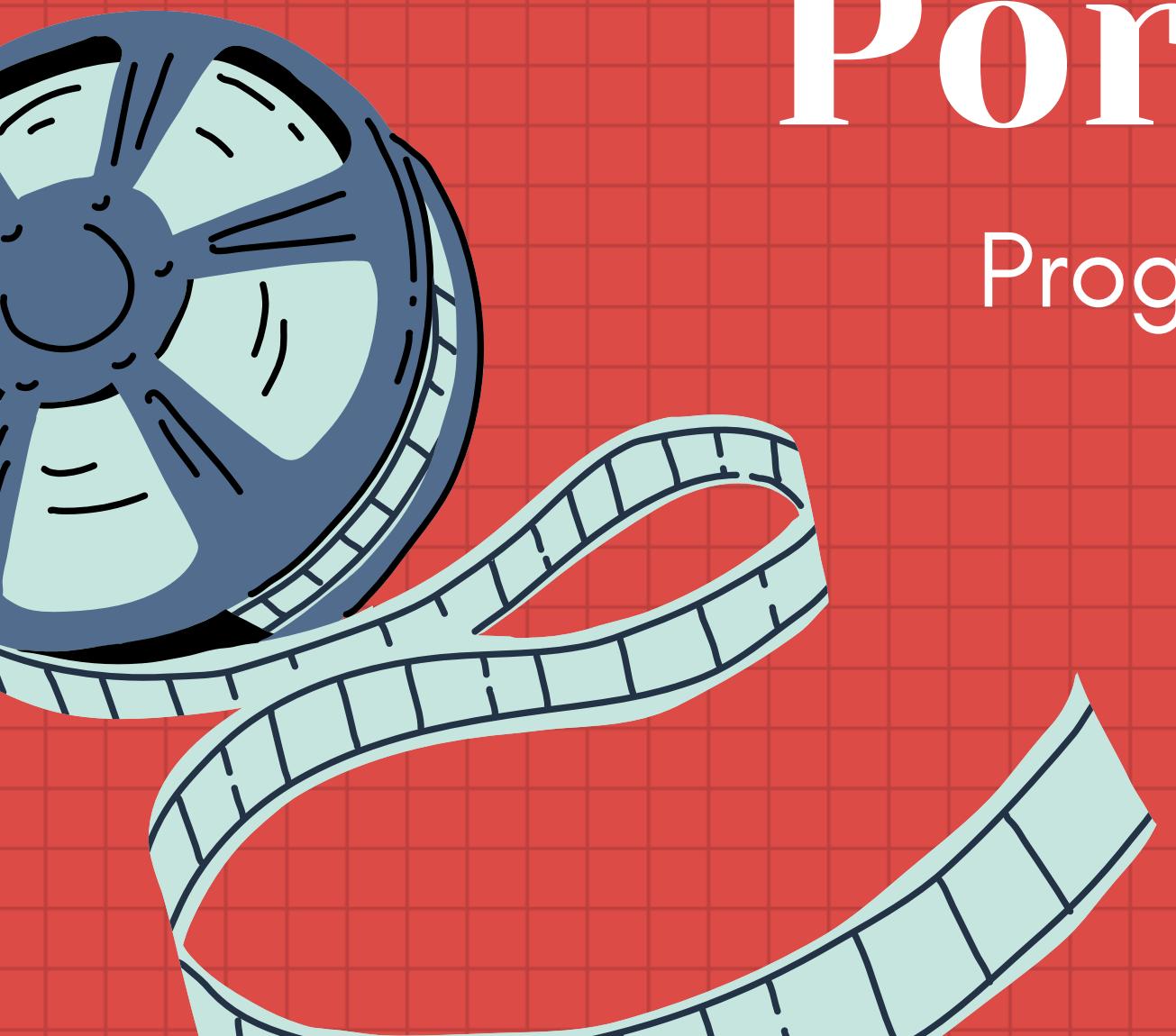


Amanda Padilla Leiva

Porfatolio Final

Programacion Orientada a objetos

Alvaro Cordero Peña

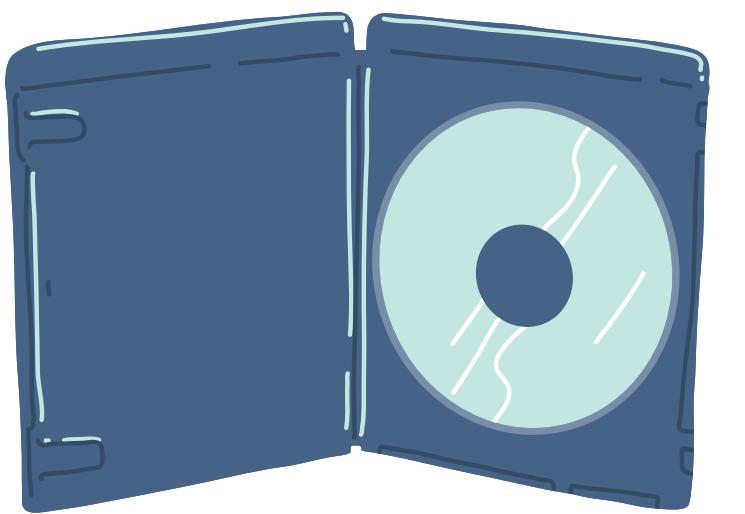




Que abarcaremos?

1

Estructura



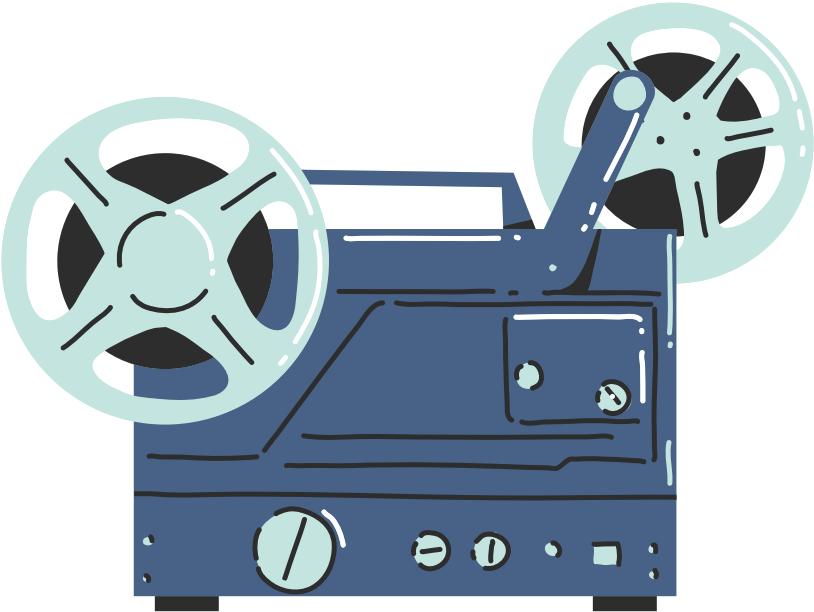
2

Conceptos



3

Función del proyecto

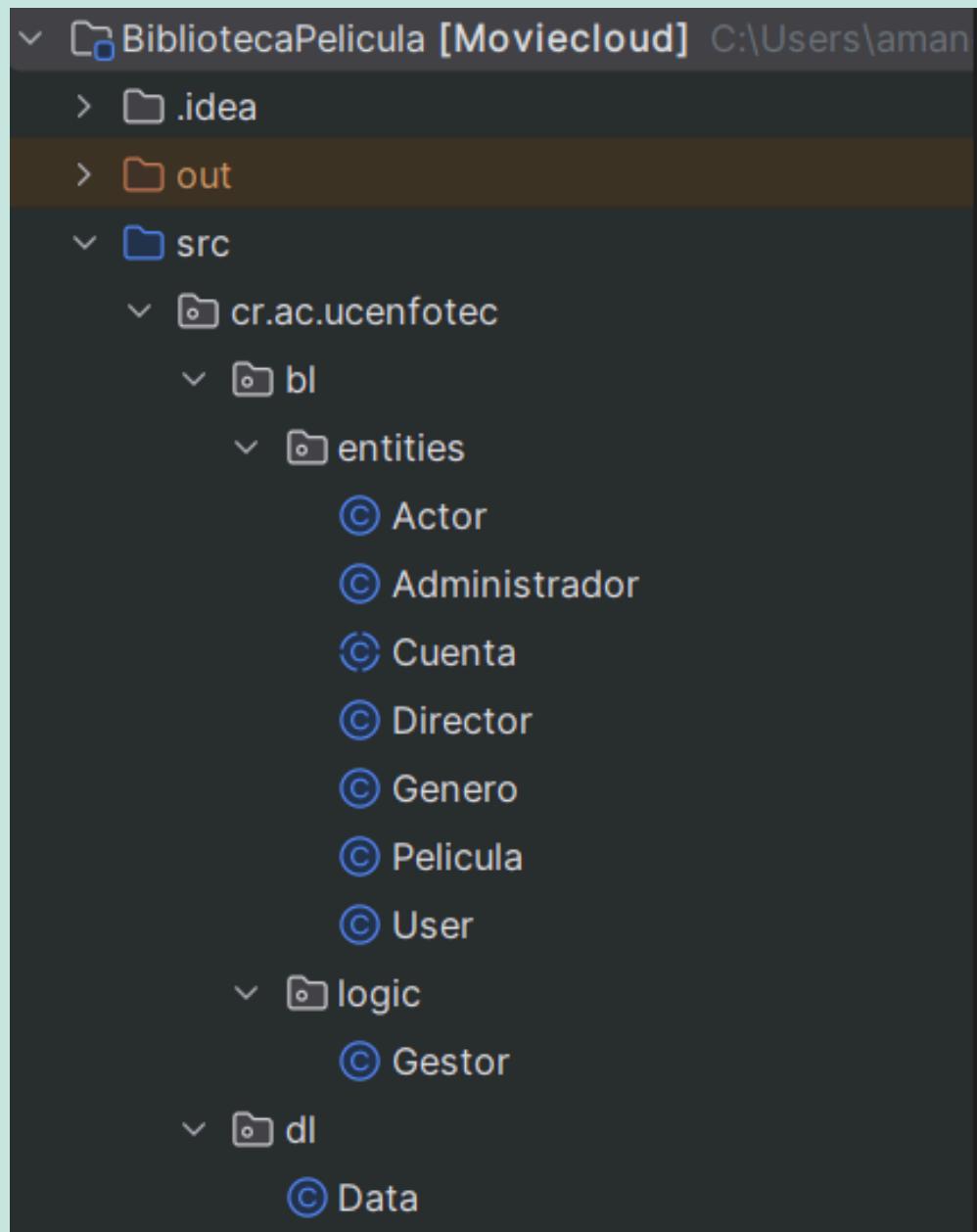


Objetivo del Portafolio

Este portafolio aborda una problemática actual inspirada en las plataformas de streaming: la gestión de una biblioteca de películas. El sistema permite que un administrador registre películas con datos como nombre, director, sinopsis, duración, género, año de estreno y elenco, además de registrar actores y directores y asociarlos a sus obras. Por otro lado, un usuario normal puede explorar el catálogo y agregar películas a su lista de favoritas, similar a como funciona Letterboxd.

Estructura

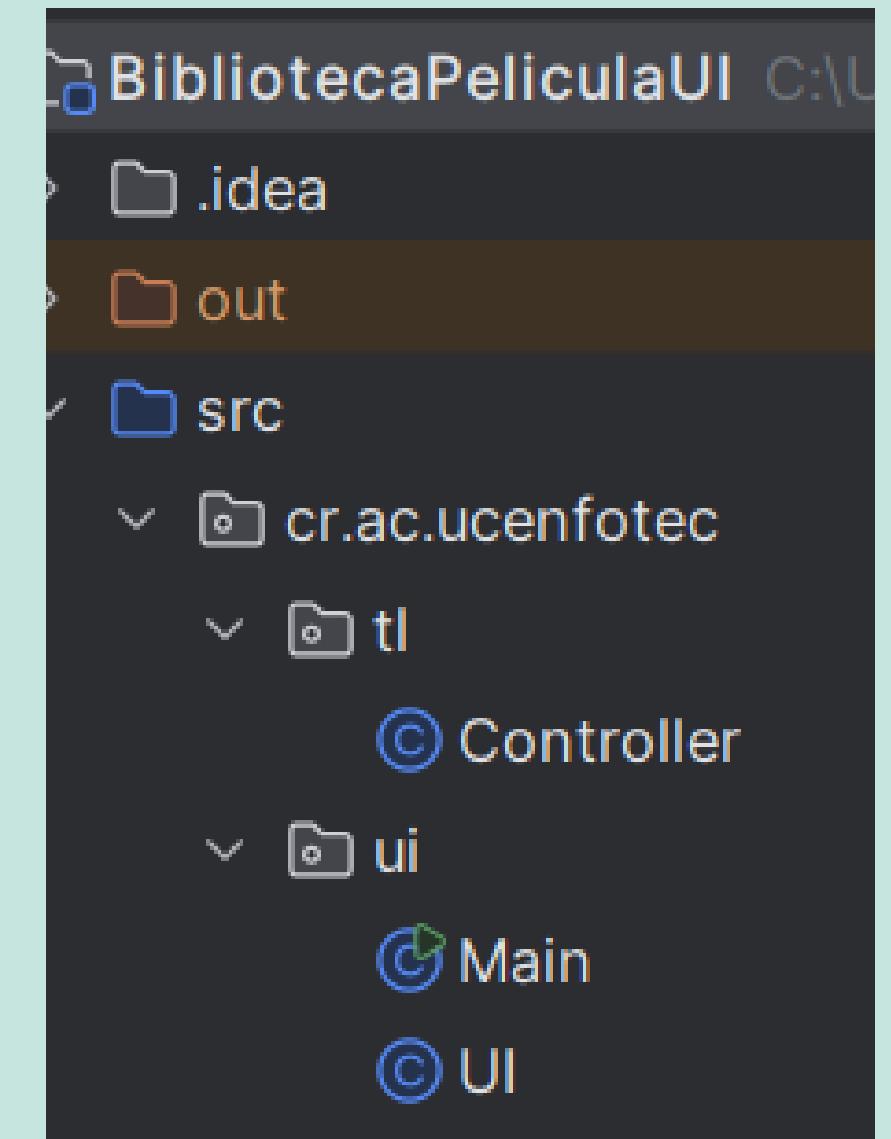
Biblioteca



El proyecto se dividió en dos partes para lograr una arquitectura más ordenada y modular. El primer proyecto contiene toda la lógica del sistema: entidades, reglas de negocio y la capa de datos. Este proyecto se compiló como un JAR, convirtiéndose en una librería reutilizable.

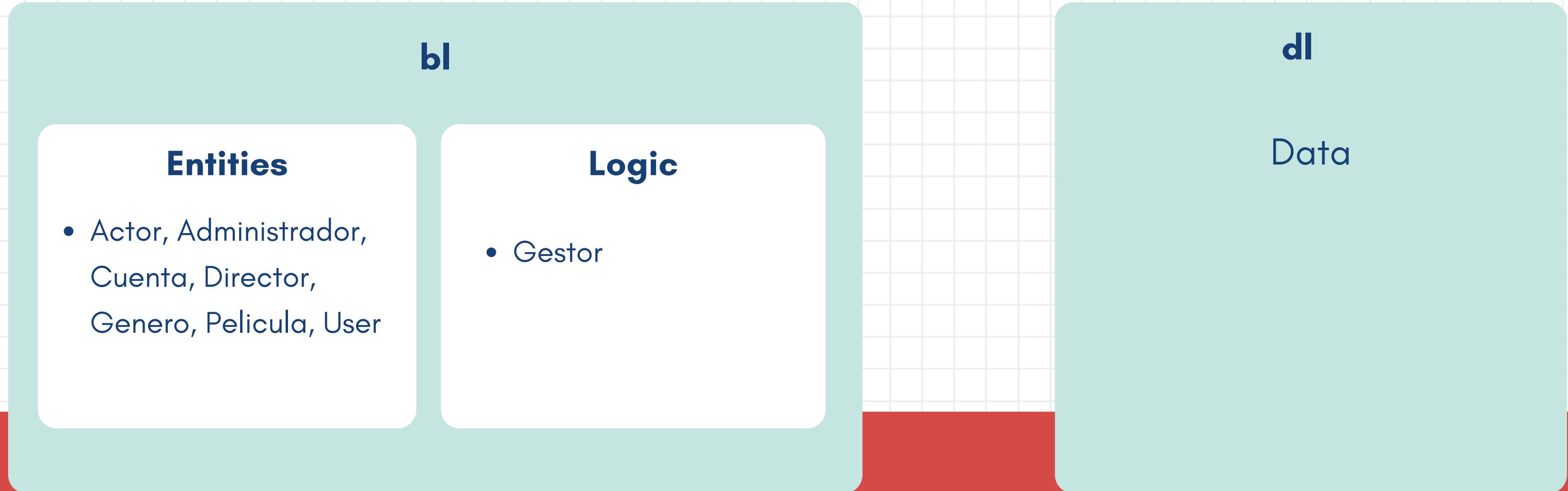
El segundo proyecto corresponde a la interfaz de usuario, que utiliza ese JAR para ejecutar las funciones del sistema sin manejar la lógica interna. Esta separación permite un código más limpio, fácil de mantener y con capas bien definidas.

Biblioteca UI



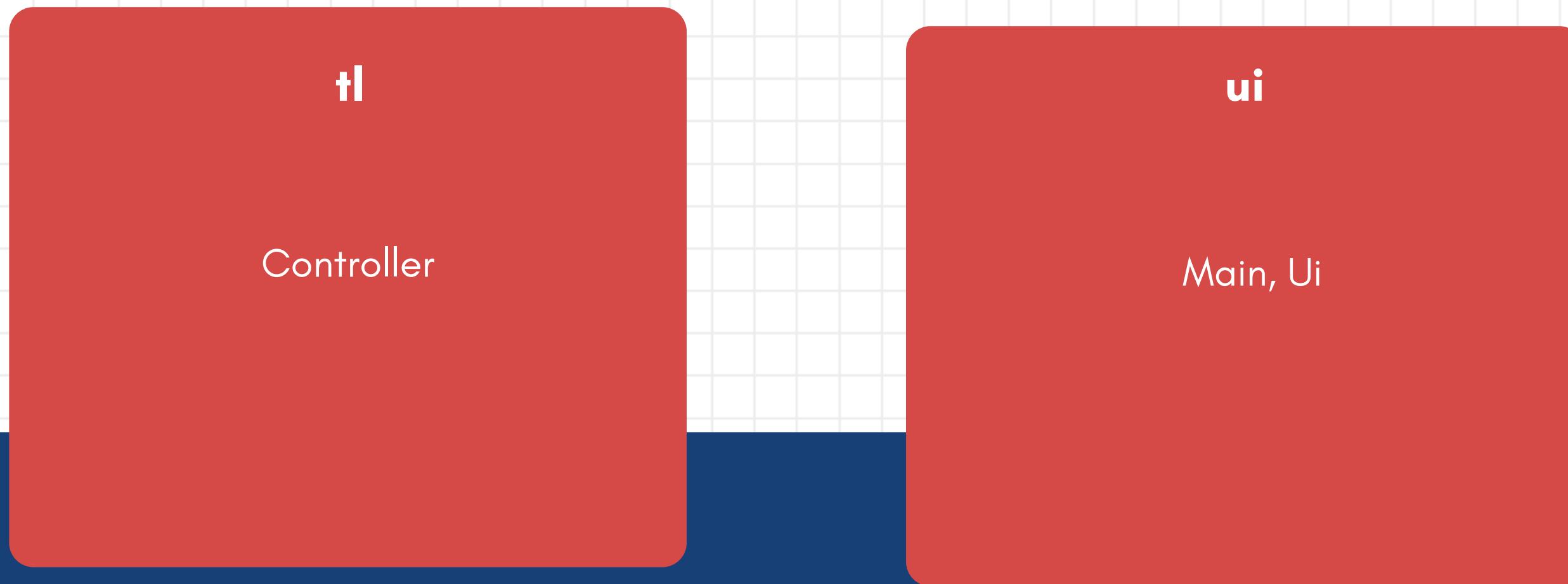
Clases del proyecto Biblioteca

El sistema se organiza mediante una arquitectura por capas que separa responsabilidades para facilitar el mantenimiento y la claridad del código. La lógica principal del programa se ubica en un proyecto independiente, que contiene las entidades, la capa de negocio (BL) y la capa de datos (DL). Este proyecto se compila como un archivo JAR, lo que permite reutilizarlo como una librería.



Clases de proyecto Biblioteca UI

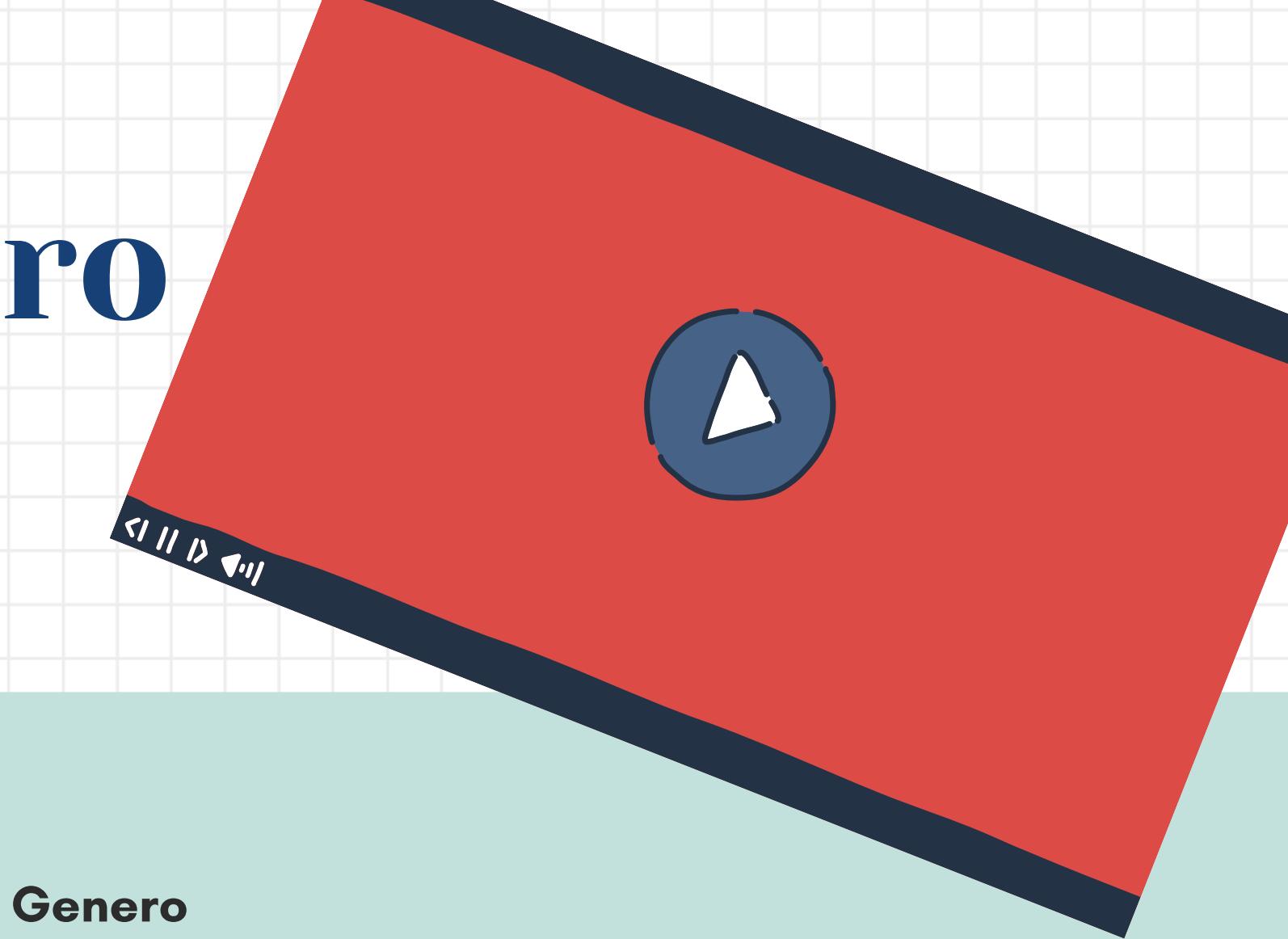
Por otro lado, el segundo proyecto contiene únicamente la capa de interfaz de usuario (UI), encargada de la interacción con el usuario. Esta interfaz utiliza el JAR del primer proyecto para acceder a todas las funcionalidades sin conocer su implementación interna. Gracias a esta división, cada capa cumple un rol específico: la UI muestra y recibe información, la capa lógica procesa reglas del sistema, y la capa de datos administra la información almacenada en memoria. Esta estructura asegura un programa ordenado, escalable y fácilmente ampliable.



Clases Película, Género

Película

- Atributos:
- Contiene información esencial de una película como id, título, año, duración, clasificación, además de listas de géneros y actores, un director asociado y una ficha con la sinopsis.
- Funciones:
- Permite acceder y modificar sus datos mediante getters y setters, gestionar géneros y elenco, y definir su identidad única mediante equals() y hashCode() basados en el id.

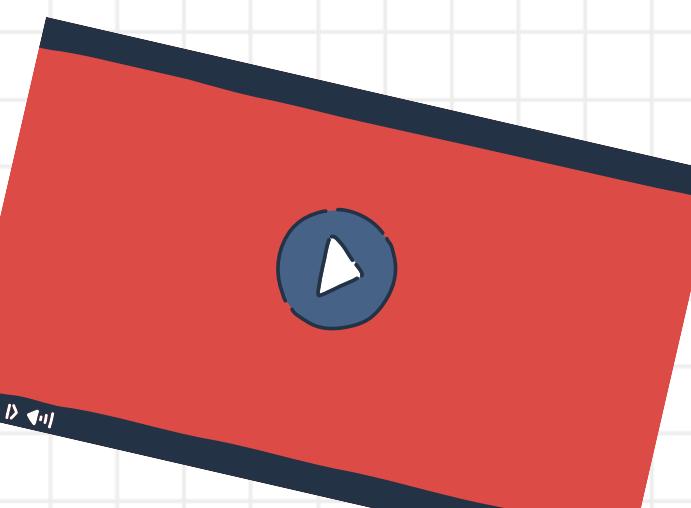


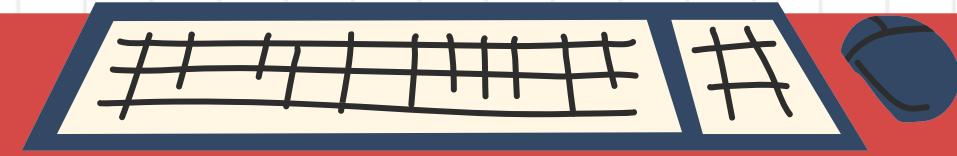
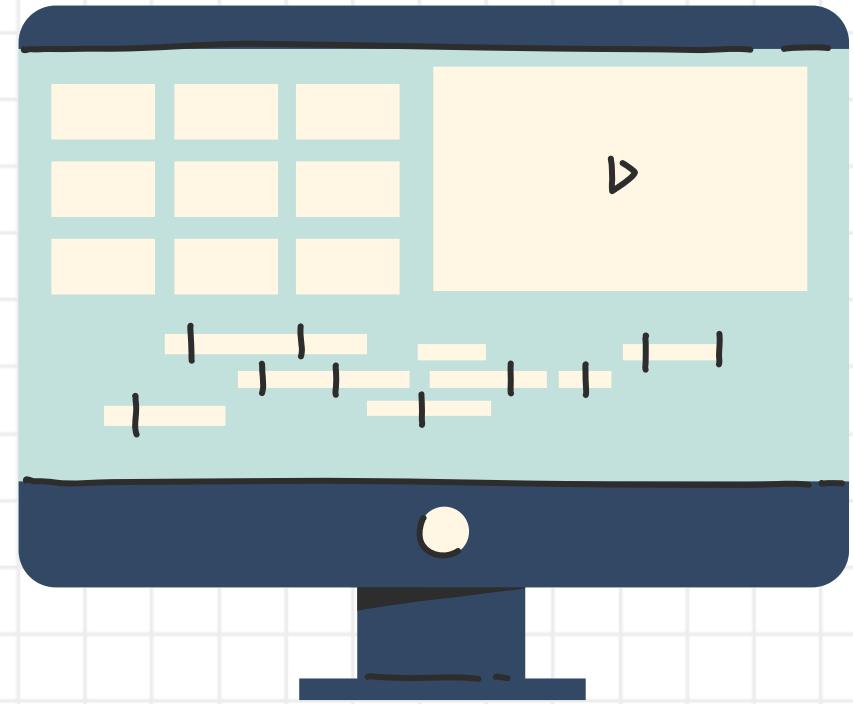
Género

Mantiene únicamente el nombre del género.

Funciones:

- Define su identidad por el nombre y facilita su representación en listas o catálogos mediante toString().





Cuenta (Abstracta)

Incluye los datos comunes de cualquier cuenta: id, username y email.

- Funciones:
- Obliga mediante un método abstracto a que las subclases definan el tipo de cuenta, y provee la estructura base de identidad y representación textual.

Clase Cuenta (Abstracta), User y Administrador

User

Heredados de Cuenta más una lista de películas favoritas.

Funciones:

- Permite agregar o eliminar favoritas, consultar la lista y determina su tipo de cuenta devolviendo "Usuario".

Administrador

Heredados de Cuenta más una lista de privilegios.

Funciones:

- Permite agregar o quitar privilegios, verificar permisos específicos y definir su tipo como "Administrador".

Clase Gestor y Data

Clase Gestor (Lógica de Negocio)

Administra una instancia de Data para gestionar toda la información del sistema.

Funciones:

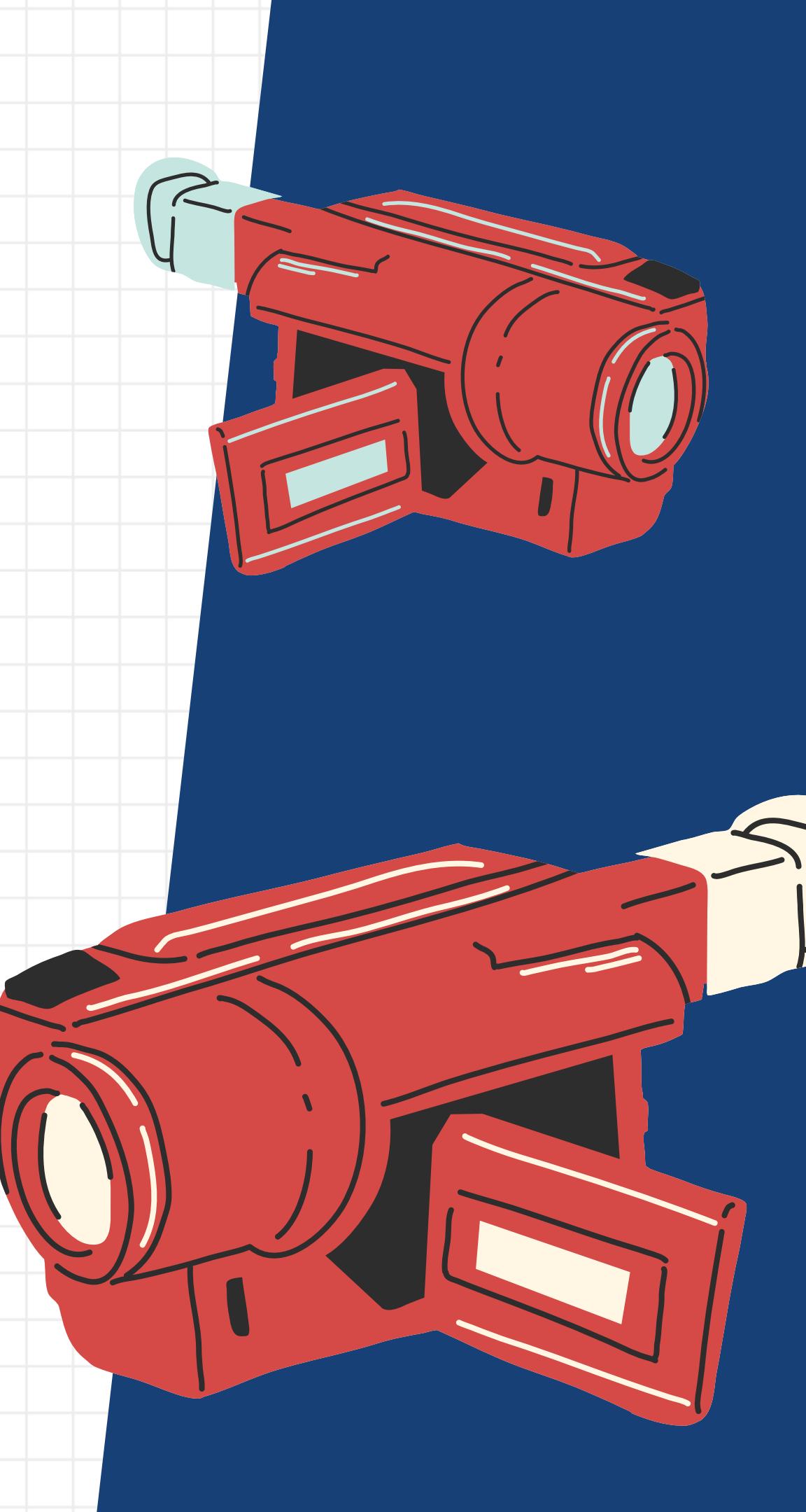
- Implementa el CRUD completo para películas, registra actores y directores, maneja asociaciones, favoritos y la lista polimórfica de cuentas.

Clase Data (Capa de Datos)

Almacena listas en memoria: películas, usuarios, administradores, actores, directores y cuentas.

Funciones:

- Permite agregar, buscar y eliminar entidades; actúa como un repositorio simple que simula persistencia en memoria.



Clase Controller y UI

Clase Controller (Capa de Control)

Mantiene un gestor, una UI y el estado de sesión.

Funciones:

- Coordina el flujo del programa, administra menús, gestiona sesiones y conecta la UI con la lógica del sistema.

Clase UI (Interfaz de Usuario)

Maneja un lector de entrada y no almacena datos persistentes.

Funciones:

- Muestra menús, captura información del usuario y presenta los resultados de las operaciones del sistema.



Conceptos - Asociación



```
public boolean asociarPeliculaConDirector(Pelicula peli, Director dir)
    if (peli == null || dir == null) return false;
    peli.setDirector(dir);
    if (dir.getPeliculasDirigidas() == null) {
        dir.setPeliculasDirigidas(new ArrayList<>());
    }
    if (!dir.getPeliculasDirigidas().contains(peli)) {
        dir.getPeliculasDirigidas().add(peli);
    }
    return true;
```

En el sistema, la asociación se da cuando dos clases se relacionan sin depender una de la otra.

Un ejemplo es Pelicula → Director o Pelicula → Actor:

la película puede existir sin tener director asignado, y los directores y actores también existen independientemente.

La relación expresa colaboración, no dependencia.



Conceptos – Agregación



```
private ArrayList<Genero> generos;
```

```
public void setGeneros(ArrayList<Genero> generos) { 1 usage & Amane  
this.generos = (generos != null) ? generos : new ArrayList<>()
```

La agregación representa una relación todo-parte donde las partes mantienen su autonomía.

Esto ocurre con Pelicula → Género: una película contiene una lista de géneros, pero los géneros no dependen de la película para existir.

Es una relación flexible y no exclusiva.

Conceptos – Composición

```
public record Ficha(String sinopsis) {}  
  
private Ficha ficha; 2 usages
```

La composición describe una relación en la que la vida de una parte depende totalmente del objeto principal.

En el sistema, esto ocurre con Pelicula → Ficha (la sinopsis). La ficha está contenida dentro de la película, no existe por sí sola y se destruye si la película desaparece.

Por eso se implementó como una clase interna.



Conceptos – Dependencia



```
d actualizarPelicula() throws IOException { 1usage  
    parMensaje( msg: "\n==== Actualizar pelicula ===");  
    id = ui.leerId( label: "ID de la pelicula a actualiza  
    a p = gestor.buscarPeliculaPorId(id);  
    if (p == null) {  
        mostrarMensaje( msg: "Pelicula no encontrada.");  
        return;
```

La dependencia aparece cuando una clase utiliza temporalmente a otra para realizar una tarea.

En este sistema, Controller depende de Gestor, y Gestor depende de Data para recuperar o modificar información.

No es una relación permanente: simplemente una clase necesita a otra en ciertos momentos para cumplir una función.

Conceptos – Herencia

```
public abstract class Cuenta { 18 usages
```

```
protected String id; 12 usages
```

```
protected String username; 6 usages
```

```
protected String email; 6 usages
```

```
public class User extends Cuenta { 22 usages & Amanda Padilla +1
```

```
private ArrayList<Pelicula> favoritos; 8 usages
```

```
/**
```

```
* Constructor por defecto.
```

```
* Inicializa la lista de favoritos vacía.
```

```
*/
```

```
public User() { } 1 usages & Amanda Padilla
```

La herencia se usa para modelar comportamientos comunes y reutilizar código.

La clase Cuenta es abstracta y define atributos compartidos por todos los tipos de usuarios.

De ella derivan User y Administrador, que extienden y especializan este comportamiento. Esto permite manejar ambas clases bajo una misma referencia.

Conceptos – Polimorfismo

```
public boolean agregarUsuario(User u) { 1 usage & Amanda Padilla
    if (u == null || u.getId() == null) return false;
    if (buscarUsuarioPorId(u.getId()) != null) return false;

    boolean agregado = usuarios.add(u);
    if (agregado) {
        cuentas.add(u); // User es una Cuenta (polimorfismo)
    }
    return agregado;
}

/*
 * Busca un usuario por su id.
 *
 * @param id identificador del usuario
 * @return el usuario encontrado o null si no existe
 */

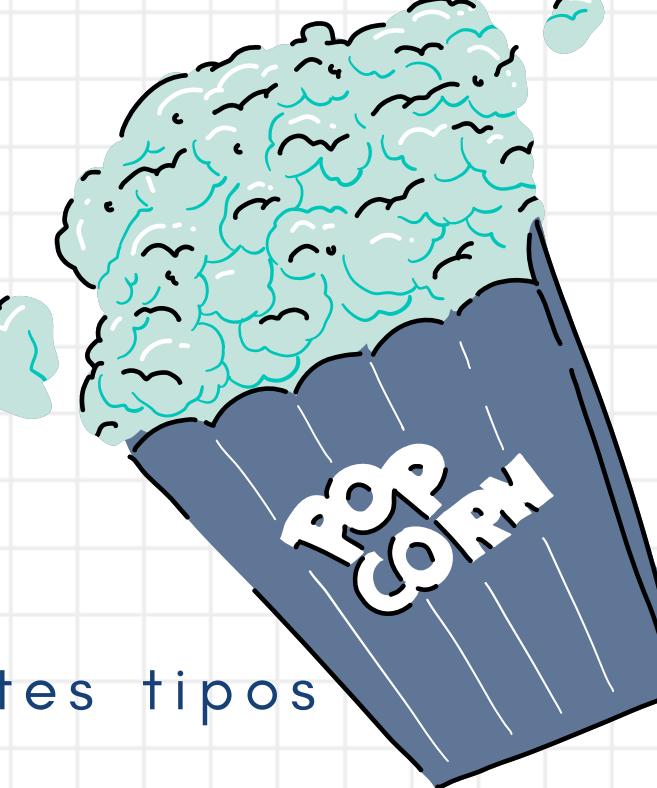
public User buscarUsuarioPorId(String id) { 2 usages & Amanda Padilla
    if (id == null) return null;
    for (User u : usuarios) {
```

Gracias a la herencia, el sistema aplica polimorfismo al manejar diferentes tipos de cuentas dentro de una misma lista.

En la clase Data, la colección List<Cuenta> puede almacenar tanto User como Administrador, y ambos se comportan según su tipo real en tiempo de ejecución.

Esto simplifica la gestión y permite que la UI trate todas las cuentas de manera uniforme.

Como funciona el proyecto?



Gracias a la herencia, el sistema aplica polimorfismo al manejar diferentes tipos de cuentas dentro de una misma lista.

En la clase Data, se puede almacenar tanto User como Administrador, y ambos se comportan según su tipo real en tiempo de ejecución.

Esto simplifica la gestión y permite que la UI trate todas las cuentas de manera uniforme.

En todo momento, la interacción fluye desde la UI, pasa por el Controller, que coordina la navegación y el estado de sesión, y llega al Gestor, que ejecuta la lógica del negocio utilizando la capa de datos. Gracias a esta estructura, el programa mantiene un orden claro entre la presentación, las reglas del sistema y el almacenamiento en memoria.

El resultado es un proyecto funcional que demuestra en la práctica todos los conceptos fundamentales de la programación orientada a objetos: herencia, polimorfismo, encapsulamiento, relaciones entre clases y arquitectura por capas.





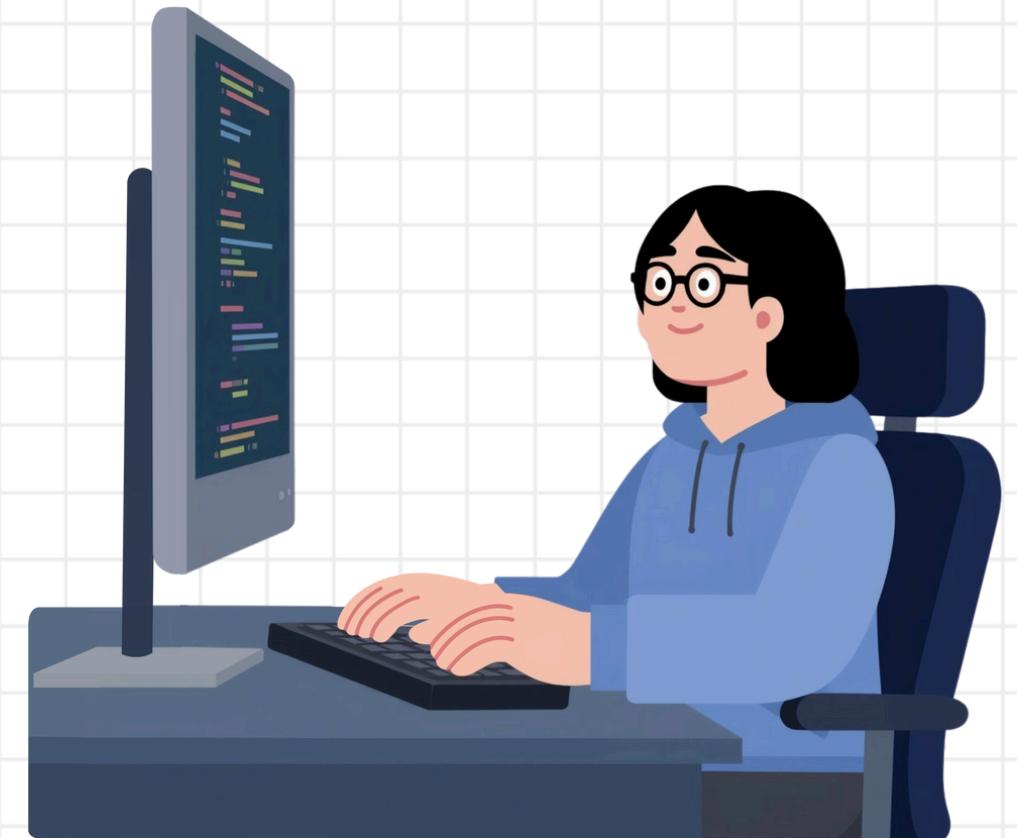
Administrador

Si se trata de un administrador, se habilita un menú completo desde el cual puede registrar, listar, actualizar o eliminar películas, además de gestionar directores y actores. El administrador puede asociar actores a películas, crear directores directamente durante el registro de una película y mantener la coherencia del catálogo. Este menú concentra todas las operaciones CRUD del sistema, permitiendo un control total sobre las entidades del dominio.

Usuarios

Usuario normal

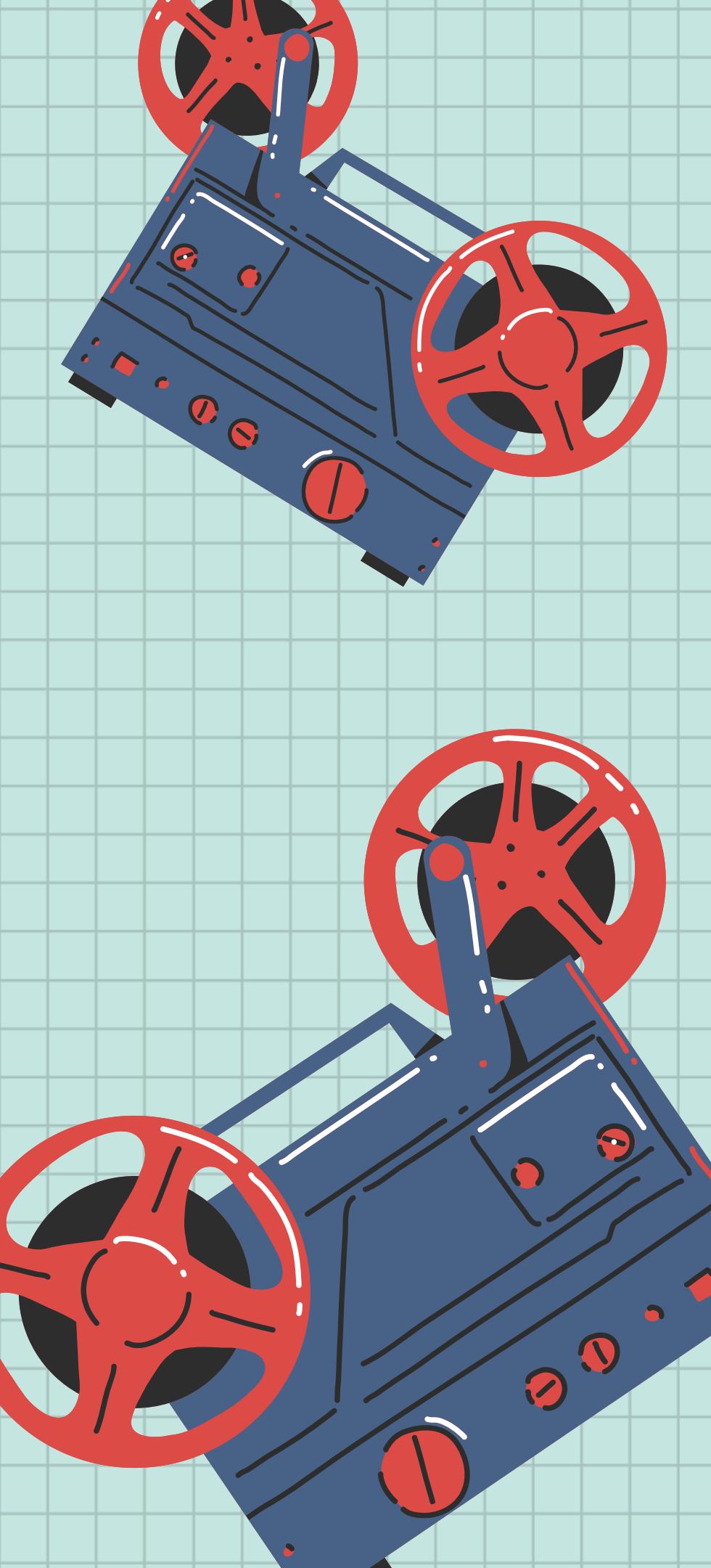
Si quien inicia sesión es un usuario normal, el sistema muestra un menú más sencillo, enfocado en la experiencia de consulta. Puede ver el catálogo de películas, revisar directores y actores, y agregar películas a su propia lista de favoritos, sin modificar la información global del sistema.



Reflexión Personal

Este proyecto me permitió aplicar de forma real los principios de POO y comprender cómo se integran dentro del diseño de un sistema. La arquitectura por capas clarificó las responsabilidades y mejoró la organización del código. Las correcciones fortalecieron el uso de herencia, polimorfismo y la identidad de objetos, permitiéndome escribir un sistema más coherente y robusto.

En general, el portafolio consolidó mi forma de pensar y estructurar software, dejando una base sólida para futuras mejoras como persistencia y una interfaz gráfica.



Muchas Gracias !

