

# 1. Definición del problema

Con respecto a la definición del problema, en nuestro caso no hay un problema que podamos definir, el proyecto que vamos a desarrollar es algo que a los internos de Nearsoft los ponen a realizar.

## 2. Requisitos

### 2.1. Requisitos funcionales

#### Servicio Web

- El usuario podrá crear o acceder a su cuenta, en caso de que no se haya creado o accedido a la cuenta el usuario todavía tendrá acceso a la plataforma, pero no se almacenará lo que este haga.
- EL usuario por uso de la herramienta de búsqueda de la plataforma podrá buscar la película por el nombre o género de este. El resultado de la búsqueda mostrará todas las películas con nombre similar o género similar.
- La plataforma cuenta con un sistema de calificación individual que muestra el rating asignado por el mismo usuario.
- El usuario podrá hacerle cambios a su cuenta (Información Personal, Contraseña, Nombre de Cuenta).
- La película que sea seleccionada por el usuario muestra la descripción de la película seleccionada
- Los ratings de las películas se mostrarán junto a la información de la película después de haber sido seleccionada por el usuario. Estos ratings serán mostrados de forma separada, los ratings obtenidos por el uso de un servicio tercero y los ratings asignados por el usuario a dicha película.

#### Servicio Web API

- Siendo la API del servicio web pública cualquier persona podrá consumir nuestro servicio para su uso personal.
- La API debe cumplir la obtención de cada recurso sea Película, Géneros o listado de películas de forma exitosa.
- El cliente podrá adquirir cualquier recurso de forma individual o en general.
- Los recursos obtenidos tendrán referencias que indican la dirección de la ubicación del recurso.
- El API al mandar los recursos de la petición del cliente, en recursos de tipo película se realizará previamente el consumo de servicios terceros para obtener los ratings de la película.

## 2.2. Requisitos no funcionales

### Servicio Web y API

#### 2.2.1. Eficiencia

- La Plataforma deberá ser capaz de soportar como máximo 50 usuarios conectados a la vez. (Nota: el número máximo de usuarios no se ha definido con claridad)
- El proceso de petición-respuesta hecha por el usuario para la información de película o rating no deberá tardar más de 1 segundo.
- La API del servicio web podrá soportar varias peticiones para un mismo recurso.
- La respuesta no debe tardar más de 1 segundo para que devuelva el recurso.
- Por medio de almacenamiento en caché se podrá mejorar el rendimiento y así evitar los errores a ocurrir cuando varias conexiones desean pedir el mismo recurso.

#### 2.2.2. Usabilidad

- El acceso a la plataforma no está determinado por el uso de una cuenta.
- El aspecto de la plataforma será sencillo tal que los usuarios menos experimentados no tengan problemas al hacer uso de este.
- El uso del API del servicio web es sencillo de usar y no requiere otras características necesarias para usarla, únicamente una conexión al API.

#### 2.2.3. Organización

- Cada vez que el usuario haya realizado un cambio en su cuenta, se actualizará la información de la cuenta del usuario ubicada en la base de datos relacional.

## 2.3 Resumen de los Requerimientos.

El número final de los requerimientos del proyecto terminaron siendo 20, a comparación de la primera entrega en el que hubo únicamente 11 y en relación con la segunda entrega no se agregó, modificó o se eliminó algún requerimiento. Como los requerimientos de la primera entrega eran únicamente entorno al servicio web se tuvo que agregar en la segunda entrega los requerimientos entorno al API del servicio web y la división de los requerimientos para cada uno de estos aspectos del servicio web como tal. El total de requerimientos agregados en la segunda entrega fueron de 9 y como previamente se había mencionado estos nuevos requerimientos eran entorno al API del servicio web e igualmente se realizó una modificación en uno de los requerimientos del servicio web este siendo que antes la parte entorno a los ratings se iba a consumir una servicio tercero para obtenerlos y luego almacenarlos en la base de datos pero este fue cambiado a que en el momento que se pida la información de la película se consumirá en el momento unos servicios terceros para la obtención y mostrado de los ratings este ya no los almacenará en la base de datos. Entorno a los requerimientos de tanto el servicio web y al API estos comparten unos requerimientos similares siendo por ejemplo el tiempo mínimo de la respuesta por parte del sistema para la información de la película o el recurso de la película (en el API). Como se puede ver solo hubo un cambio grande en los requerimientos entre la primera entrega y la segunda, para los requerimientos de la entrega final no se realizaron ningún cambio ni se agrego o elimino se mantuvo los requerimientos como estaban en la segunda entrega siendo la razón que nosotros veíamos que ya eran estos requerimientos completos y se apegaban a los cambios hechos en el transcurso después de la primera entrega.

### 3. Escenarios de uso

#### *Escenarios de Uso entorno al Servicio Web*

##### 3.1. Información de la película

###### 3.1.1. Información general

Datos del escenario	
Nombre	Información de película
Actores	Usuario
Descripción	El usuario busca la película que desee checar y al seleccionarla podrá ver la información de la película junto al rating que tiene este.
Precondición	El usuario tiene acceso a la plataforma.
Postcondición	Ninguna

###### 3.1.2. Flujo principal

1. *Usuario*: El usuario hace la búsqueda de la película mediante la herramienta de búsqueda.
2. *Sistema*: El sistema devuelve en pantalla todos los títulos con el nombre o género que usó el usuario para buscar la película
3. *Usuario*: El usuario selecciona el título que desea checar.
4. *Sistema*: El sistema carga la información y el rating del título seleccionado por el usuario.

###### 3.1.3. Flujo alternativo

1. *Sistema*: El sistema no encuentra el título que el usuario desea buscar.
2. *Sistema*: El sistema no logra mandarle al usuario la información y rating de la película, puede ser por desconexión del servidor con el usuario o con el sistema.

##### 3.2. Recibir ratings

###### 3.2.1. Información general

Datos del escenario	
Nombre	Recibir ratings
Actores	Sistema
Descripción	El sistema hace una petición a los servicios terceros para poder recibir los ratings de cierta película. De igual forma se conecta con la base de datos para obtener los ratings del usuario.
Precondición	El sistema ya recibió una petición hecha por el usuario.
Postcondición	Ninguna

###### 3.2.2. Flujo principal

1. *Sistema*: El sistema realiza las acciones de comunicación con los servicios terceros, y consume el recurso correspondiente para obtener los ratings de una película en especial

2. *Sistema*: Muestra en pantalla el recurso consumido para que el usuario lo vea.
3. *Sistema*: El sistema realiza la acción de conectarse a la base de datos y le manda la petición a la base de datos para obtener los ratings asignados por el usuario
4. *Base de Datos*: La Base de Datos no relacional procesa la petición y devuelve la información que la petición pedía.
5. *Sistema*: Recibe la información de la base de datos no relacional y lo muestra en pantalla para el usuario.

### 3.2.3. Flujo alternativo

1. *Base de datos*: No pudo procesar la petición, se rechazó la petición o se perdió la comunicación con el sistema.
2. *Sistema*: No pudo consumir bien el recurso del servicio tercero, no encontró el recurso, no pudo conectarse al servicio tercero.

## Escenarios de Uso entorno al API

### 3.3. Recurso Película

#### 3.3.1. Información general

Datos del escenario	
Nombre	Recurso película
Actores	Cliente/Usuario
Descripción	El cliente/usuario hace una petición del método HTTP GET para obtener el recurso de una película en especial.
Precondición	El cliente/usuario tiene conexión al API
Postcondición	Ninguna

#### 3.3.2. Flujo principal

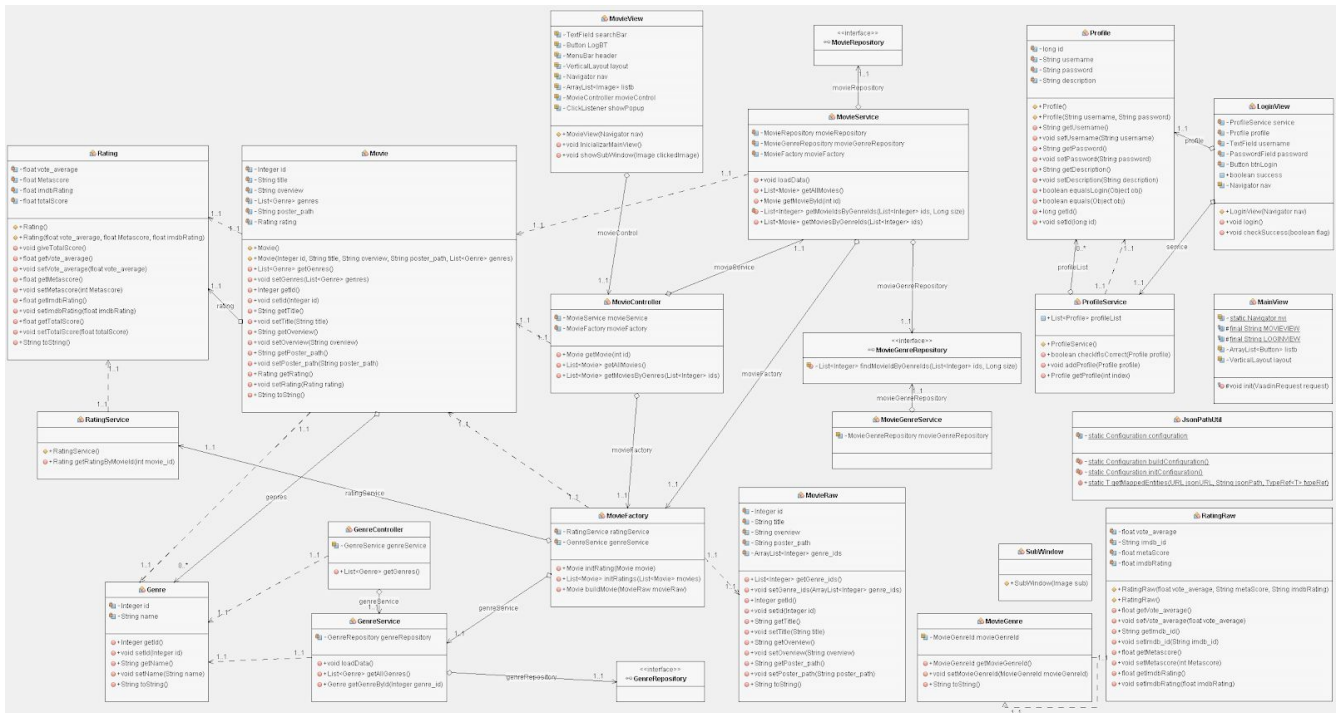
- 1.*Cliente/Usuario*: Accede a la URL base de nuestra API.
- 2.*Cliente/Usuario*: Extiende la URL de forma que apunte al recurso que quiere.
- 3.*API*: Checa que se realizó un método HTTP GET en la dirección URL, se realiza el método correspondiente a dicha URL y se devuelve el recurso correspondiente.
- 4.*Cliente/Usuario*: El cliente/usuario recibe en formato JSON el recurso correspondiente a su petición.

#### 3.3.3. Flujo alternativo

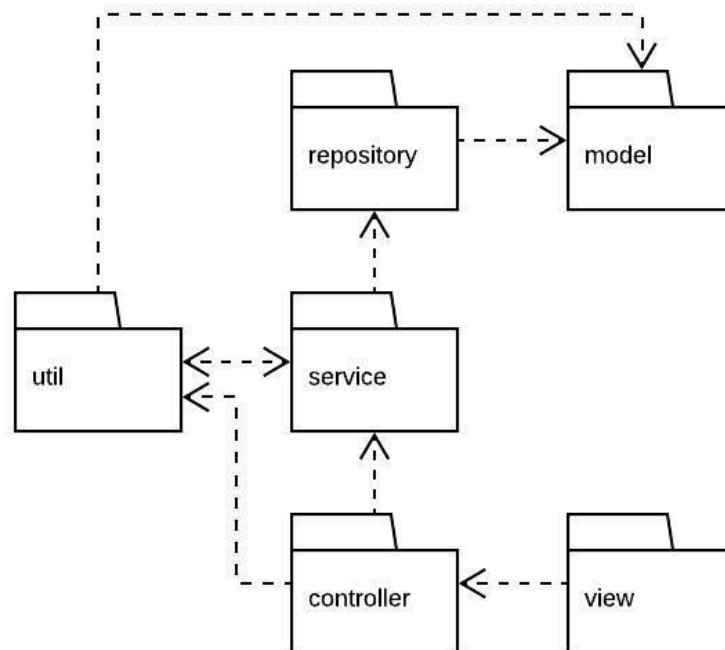
- 1.*Cliente/Usuario*: No tiene acceso o conexión al API.
- 2.*API*: No existe la ruta al recurso que el cliente/usuario ingreso.
- 3.*API*: No existe el recurso. El recurso que se devuelve no está completo.

## 4. Diagramas

### 4.1 Diagrama de clases



### 4.2. Diagrama de paquetes MVC



## 5. Casos de Uso

### 5.1.1 Casos de Uso del Servicio Web

Case Name	Principal Actor	Description	Priority (5 Highest 1 Lowest)
Log-in	User	"User" Puede iniciar sesión si cuenta con una cuenta o crear una si así lo desea.	5
Account Information	User	"User" Puede modificar la información de su cuenta	5
Movie Searcher	User	"User" Puede buscar películas desde un buscador por Título ó género.	3
Ratings	User/ServicioWeb	"User" puede asignar un rating a una película. El servicio consumirá un tercer servicio para la obtención de los ratings.	5
Profiles	User/ServicioWeb	"User" Puede crear ó remover perfiles en una misma cuenta. El servicio web permite al usuario realizar dichas acciones.	3
Add/Remove movies	Admin	"Admin" Puede agregar o remover películas de la plataforma.	5

## 5.1.2 Flujo alternativo de los Casos de Uso.

Case: Log-in

Actors: User.

Flujo Alternativo:

1. Hubo un error de conexión que por consecuencia negó el acceso del usuario a su cuenta.
2. La contraseña o nombre de usuario fueron incorrectos, se le pide al usuario ingresar los datos de forma correcta.
3. El usuario perdió la conexión con el servicio.

Case: Account-Information.

Actors: User.

Flujo Alternativo:

1. No se refleja los cambios establecidos en el perfil.
2. No se tiene acceso a la edición de la información del perfil.
3. El usuario perdió la conexión con el servicio.

Case: Movie-Searcher

Actors: User.

Flujo Alternativo:

1. No existe el recurso de la película introducida para buscarla.
2. Errores del cliente o servidor.
3. No se carga el recurso después de haberlo encontrado con éxito.

Case: Rating

Actors: User and Webservice.

Flujo Alternativo:

1. No se pudo realizar la asignación del rating por el usuario.
2. El ServicioWeb no pudo obtener los ratings de la película presente y por ende su calificación queda nula.
3. La conexión a los servicios terceros para los ratings fue interrumpida.
4. No se refleja la asignación de rating hecho por el usuario.

Case: Profiles

Actors: User and Webservice.

Flujo Alternativo:

1. Se produjo un error en la parte de la creación o eliminación del perfil del usuario.
2. El usuario no puede crear un perfil o eliminarlo.
3. Se perdió conexión con la base de datos o no se tiene acceso a la base de datos.
4. Errores generales de cliente o servidor.

Case: Add/Remove Movies.

Actors: Admin.

Flujo Alternativo:

1. No se pudo eliminar o agregar una película.
2. El Admin no cuenta con un nivel aceptable para realizar acciones de eliminar o agregar, solo puede gestionar y administrar.
3. No se refleja los cambios en la base de datos relacional.

## 5.2.1 Casos de Uso del API.

Case Name	Principal Actor	Description	Priority (5 Highest 1 Lowest)
MovieResource	Client/User	El cliente/usuario puede mandar una petición GET para obtener el recurso de una película en especial.	4
GenreResource	Client/User	El cliente/usuario puede pedir el recurso de los géneros de películas.	4
GeneralMovieResource	Client/User	El cliente/usuario puede pedir la colección completa de las películas o película con el mismo género.	4

## 5.1.2 Flujo alternativo de los Casos de Uso.

Case: MovieResource.

Actors: Client/User.

Flujo Alternativo:

1. La ruta del recurso introducido no es existente.
2. El recurso que se desea obtener no existe o no se ha encontrado.
3. Error de servidor, se pierde la conexión con el servidor por lo que no se puede acceder a la base de datos para obtener el recurso.
4. Se obtuvo el recurso, pero no está completo.



Case: GenreResource.

Actors: Client/User.

Flujo Alternativo:

1. No se encuentra ninguna lista de géneros por lo tanto no se puede enviar el recurso.
2. Conexión perdida por parte del cliente o del servidor.
3. No existe el recurso género con el id introducido.

Case: GeneralMovieResource.

Actors: Client/User.

Flujo Alternativo:

1. No fue posible enviar el recurso porque no existe películas con el id del género introducido en común.
2. Se obtuvo el recurso, pero se incluye películas con géneros diferentes.
3. No se encontró ningún recurso, o la base de datos con la información de películas esta vacía.

## 6. Mapeo de requerimientos.

Requerimientos agregados.

### Ω Funcionales

- ω Cualquier persona podrá consumir la API nuestro servicio para su uso personal.
  - Proceso: Completado
- ω La API debe cumplir la obtención de cada recurso de forma exitosa.
  - Proceso: Completado
- ω El cliente puede adquirir muchos recursos a la vez.
  - Proceso: Completado
- ω Los recursos obtenidos tendrán referencias que indiquen su ubicación.
  - Proceso: Completado
- ω Al solicitar los ratings de una película se va a realizar el consumo de un servicio de terceros para su obtención.
  - Proceso: Completado

### Ω No funcionales (eficiencia)

- ω La API del servicio web podrá soportar varias peticiones para un mismo recurso.
  - Proceso: Completado
- ω Por medio de almacenamiento en caché se va a mejorar el rendimiento, evitando errores que puedan surgir cuando varias conexiones piden un mismo recurso.
  - Proceso: Completado

### Ω No funcionales (usabilidad)

- ω El API del servicio web es sencillo de usar y no requiere características adicionales para su uso.
  - Proceso: Completado

Requerimientos suprimidos:

### Ω Funcionales

- ω El usuario podrá crear o acceder a su cuenta. En caso de que no se haya creado o accedido a la cuenta el usuario todavía tendrá acceso a la plataforma, pero no se va a almacenar lo que éste haga.

- ω La plataforma cuenta con un sistema de calificación individual que muestra el rating asignado por el mismo usuario.
- ω El usuario podrá hacerle cambios a su cuenta (información personal, contraseña, nombre de cuenta).

#### Ω No funcionales (organización)

- ω Cada vez que el usuario haya realizado un cambio en su cuenta, se actualizará la información de la cuenta del usuario ubicada en la base de datos relacional.

Requerimientos existentes.

#### Ω Funcionales

- ω EL usuario, mediante la herramienta “búsqueda”, podrá buscar una película por su nombre o género. El resultado de la búsqueda mostrará todas las películas con nombre o género similar.
  - Proceso: Completado.
- ω La película que sea seleccionada por el usuario también va a mostrar su descripción.
  - Proceso: Completado.
- ω Los ratings de las películas se mostrarán junto a la información de la película después de haber sido seleccionada por el usuario.
  - Proceso: Completado.

#### Ω No funcionales (eficiencia)

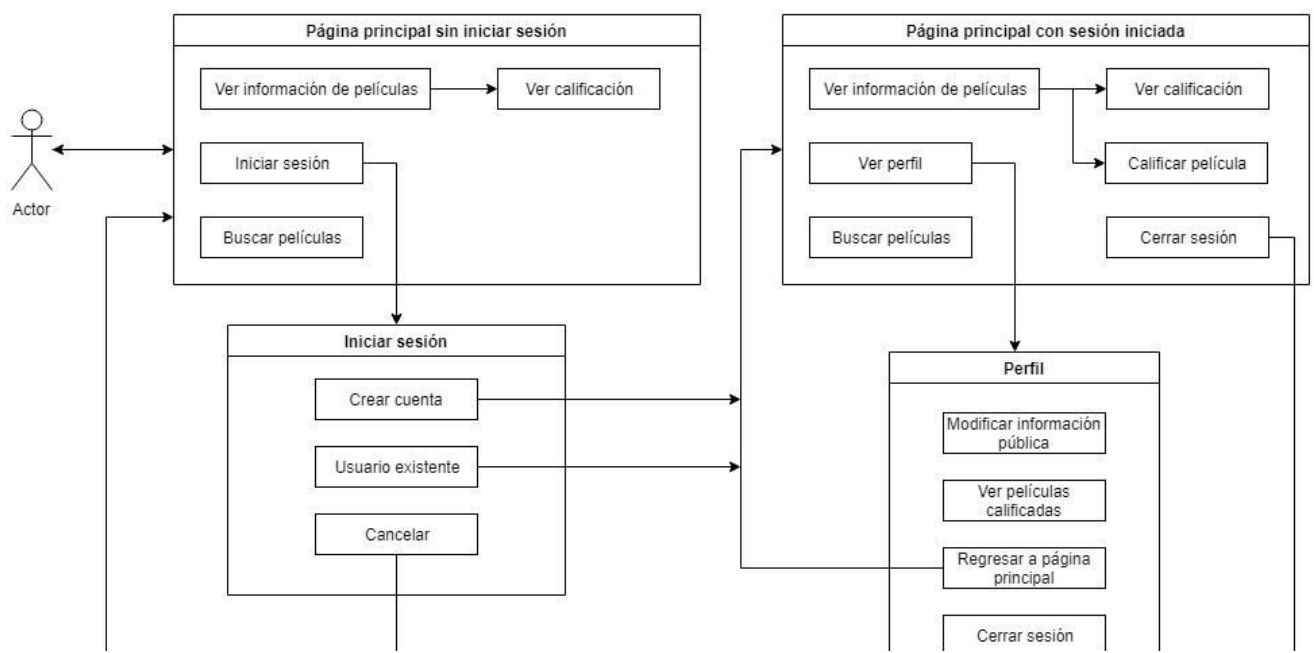
- ω La Plataforma deberá ser capaz de soportar como máximo 50 usuarios conectados a la vez. (Nota: el número máximo de usuarios no se ha definido con claridad)
  - Proceso: Sin medios para probar.
- ω El proceso de petición-respuesta hecha por el usuario para la información de película o rating no deberá tardar más de 1 segundo.
  - Proceso: Sin medios para probar.

#### Ω No funcionales (usabilidad)

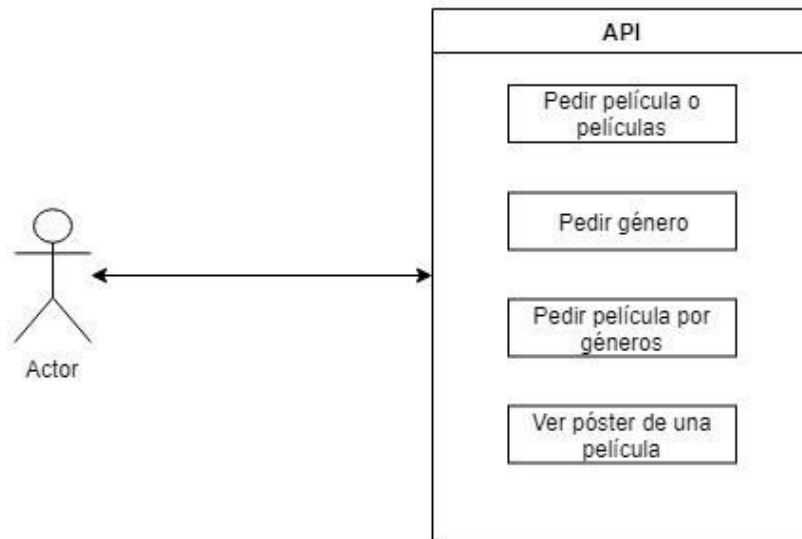
- ω El acceso a la plataforma no está determinado por el uso de una cuenta.
  - Proceso: Completado.
- ω El aspecto de la plataforma será sencillo, de tal manera que los usuarios menos experimentados no tengan problemas al hacer uso de esta.
  - Proceso: Completado.

## 7. Diagramas de Casos de Uso

### Servicio web



## API



## Bitácora de actividades

### Investigaciones *(Octubre)*

Solid: Javier.

Vaadin: Javier.

Spring boot: Hernán.

SQL y NoSQL: César.

Diego: REST API.

Maven: Hernán.

Hibernate: Hernán.

### Diseños *(Octubre)*

Diseño de bases de datos: César y Hernán.

Casos de uso: Javier y César.

Endpoints: Diego.

Diseño del servicio web: Diego

Diagrama de paquetes: Hernán.

### Codificación *(Octubre-Diciembre)*

Conexión de una base de datos mediante Hibernate: Hernán.

Front end que se conecte a la base de datos: Diego.

Servicio de la aplicación: Hernán y Diego.

Creación de las clases: Hernán y Diego.

Endpoints: Diego.

Compatibilidad Spring-Vaadin: Diego.

Carga visual de los pósters: Diego.  
Sub-ventanas de Log-in y Películas: Diego.  
Módulo de películas: Hernán.  
Barra de búsqueda: Hernán.  
Servicio (películas y perfiles): Hernán.

## **Documentación** *(Noviembre-Diciembre)*

### **Primera entrega:**

Diego Escenarios de uso y Requerimientos.  
Javier Casos de uso y Diagrama de casos de usos.  
César : Diseño preliminar de DB  
Hernán Diagrama de clases.

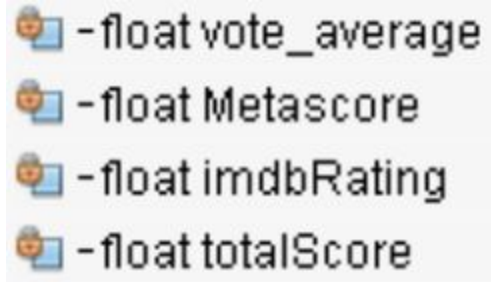
### **Segunda entrega:**




Diego Editado de requerimientos.  
Javier Documento incluido en el repositorio.  
César Mapeo de requerimientos y Diagramas de casos de usos.  
Hernán Diagrama paquetes.














### **Entrega final:**

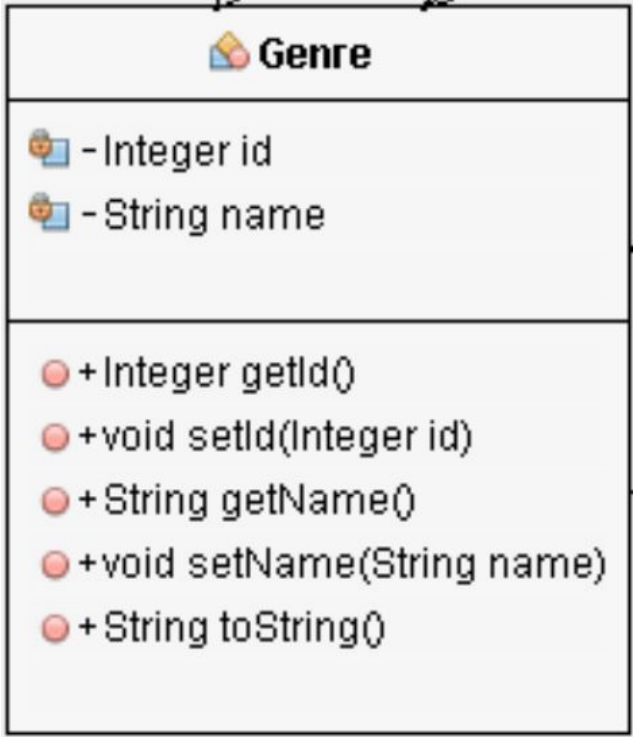
Diego Resumen de requerimientos  
Javier Proceso de Desarrollo.  
César Mapeo de requerimientos  
Hernán

## **Mapeo de diseño**

<b>Antes</b>	<b>Ahora</b>
Ratings: No estaban en una clase	<p>Ratings es una clase que tiene como datos:</p>  <p>Con métodos importados desde JDBC y también tiene implementado un servicio llamado RatingService que permite obtener el rating con el ID de una película.</p>
Movie era una clase que tenía como datos:	Movie es una clase que tiene como datos:

<ul style="list-style-type: none"> <li>- title: String</li> <li>- description: String</li> <li>- rating: String</li> </ul>	<div data-bbox="824 153 1360 613">  <ul style="list-style-type: none"> <li>- Integer id</li> <li>- String title</li> <li>- String overview</li> <li>- List&lt;Genre&gt; genres</li> <li>- String poster_path</li> <li>- Rating rating</li> </ul> </div> <p>Y también tiene implementado un servicio llamado MovieService con los siguientes métodos:</p> <div data-bbox="824 697 1461 861">  <ul style="list-style-type: none"> <li>+void loadData()</li> <li>+List&lt;Movie&gt; getAllMovies()</li> <li>+Movie getMovieById(int id)</li> <li>-List&lt;Integer&gt; getMovieIdsByGenreIds(List&lt;Integer&gt; ids, Long size)</li> <li>+List&lt;Movie&gt; getMoviesByGenreIds(List&lt;Integer&gt; ids)</li> </ul> </div>
<p>Movieview: no estaba en una clase.</p>	<p>MovieView es una clase que tiene como componentes:</p> <div data-bbox="824 976 1461 1495">  <ul style="list-style-type: none"> <li>~TextField searchBar</li> <li>~Button LogBT</li> <li>~MenuBar header</li> <li>~VerticalLayout layout</li> <li>~Navigator nav</li> <li>~ArrayList&lt;Image&gt; listb</li> <li>~MovieController movieControl</li> <li>~ClickListener showPopup</li> </ul> </div>
<p>Controladores: Antes no teníamos implementado ningún controlador.</p>	<p>Controladores: Implementamos los siguientes controladores</p>

	<div><div><div> <b>MovieController</b></div><div><div> - MovieService movieService</div><div> - MovieFactory movieFactory</div></div><div><div> + Movie getMovie(int id)</div><div> + List&lt;Movie&gt; getAllMovies()</div><div> + List&lt;Movie&gt; getMoviesByGenres(List&lt;Integer&gt; ids)</div></div></div></div> <div><div><div> <b>GenreController</b></div><div><div> ~ GenreService genreService</div></div><div><div> + List&lt;Genre&gt; getGenres()</div></div></div></div>
User: Clase que guardaba la contraseña, el nombre de usuario y su historia.	<div><div>Se cambió el User por Profile, con los datos:</div><div><div> - long id</div><div> - String username</div><div> - String password</div><div> - String description</div></div><div>y los métodos</div></div>

	<ul style="list-style-type: none"> <li>◆ +Profile()</li> <li>◆ +Profile(String username, String password)</li> <li>● +String getUsername()</li> <li>● +void setUsername(String username)</li> <li>● +String getPassword()</li> <li>● +void setPassword(String password)</li> <li>● +String getDescription()</li> <li>● +void setDescription(String description)</li> <li>● +boolean equalsLogin(Object obj)</li> <li>● +boolean equals(Object obj)</li> <li>● +long getId()</li> <li>● +void setId(long id)</li> </ul>
Los géneros no estaban contemplados	<p>Hay una clase de géneros:</p>  <pre> classDiagram     class Genre {         -Integer id         -String name         +Integer getId()         +void setId(Integer id)         +String getName()         +void setName(String name)         +String toString()     } </pre> <p>The diagram shows a class named <b>Genre</b> with two attributes: <code>- Integer id</code> and <code>- String name</code>. It has six methods: <code>+ Integer getId()</code>, <code>+ void setId(Integer id)</code>, <code>+ String getName()</code>, <code>+ void setName(String name)</code>, and <code>+ String toString()</code>.</p>
No se tomaban en cuenta los servicios.	Hay varias clases dedicadas a los servicios:

### GenreService




 - GenreRepository genreRepository


- +void loadData()
- +List<Genre> getAllGenres()
- +Genre getGenreById(Integer genre\_id)

### RatingService

- +RatingService()
- +Rating getRatingByMovieId(int movie\_id)

### MovieService

 - MovieRepository movieRepository  
 - MovieGenreRepository movieGenreRepository  
 - MovieFactory movieFactory

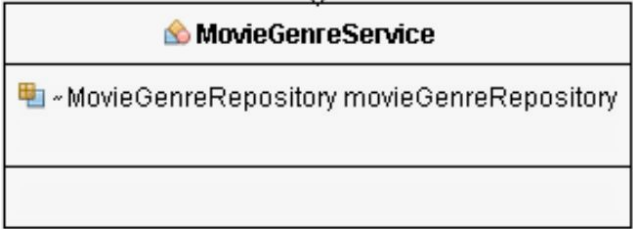
- +void loadData()
- +List<Movie> getAllMovies()
- +Movie getMovieById(int id)
-  -List<Integer> getMovieIdsByGenreIds(List<Integer> ids, Long size)
- +List<Movie> getMoviesByGenreIds(List<Integer> ids)

### ProfileService

 +List<Profile> profileList

- +ProfileService()
- +boolean checkIfIsCorrect(Profile profile)
- +void addProfile(Profile profile)
- +Profile getProfile(int index)



	
No se contemplaba la interfaz en el diseño	Hay interfaces para las películas y para la lista de películas-géneros.

**La documentación del código está en el código fuente**

## Descripción de Roles, Tareas y Actividades

Integrante	Roles	Tareas	Actividades
Diego	Backend (Rest API, Spring Boot)	Agendar reuniones y juntas del equipo. Monitoreo del progreso del proyecto. Desarrollo del módulo de Ratings del API. Desarrollo de la funcionalidad y diseño base de la parte visual del servicio web.	Estudio de la estructuración y creación de un REST API,
Hernán	Backend (Spring, Spring Boot ,Hibernate, MySQL)	Creación de la base de datos del servicio web(Servicio web y API). Desarrollo del módulo de Películas del API. Desarrollo de la comunicación y almacenamiento en la base de datos.	Estudio de los conceptos necesarios para hacer uso de los framework como Spring y Hibernate.
Cesar	Backend (Bases de datos MYSQL)		Estudio de bases de datos MYSQL NOSQL
Javier	Frontend (Vaadin)	Diseño de UI en vaadin para el FrontEnd -Login e implementación -Register -MainPage -Profile	Estudio de SOLID, Estudio de Vaadin

## Artefactos Producidos

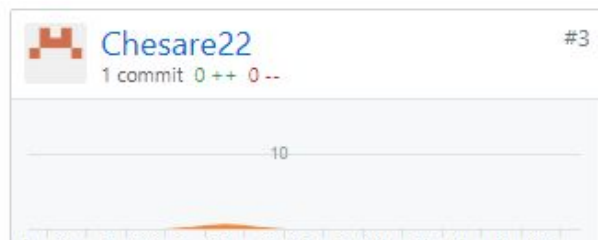
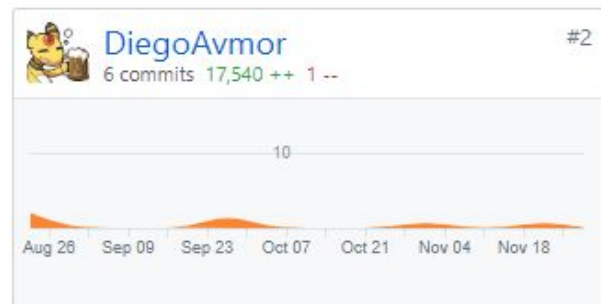
Integrante	Artefactos
Diego	Endpoints de API, RatingService, Userservice, Escenarios de uso
Hernán	Interfaz visual, Modulo de peliculas,Profile system,Diagrama de clases
Cesar	Diagrama de clases, Casos de uso
Javier	Interfaz visual,LoginService , Casos de uso

## Reporte de contribuciones del repositorio

Aug 26, 2018 – Dec 2, 2018

Contributions: Commits ▼

Contributions to master, excluding merge commits



Aug 26, 2018 – Dec 2, 2018

Contributions: Additions ▾

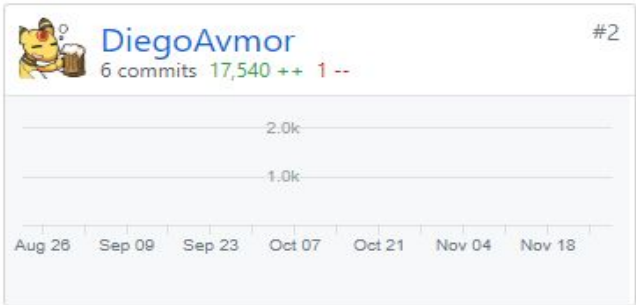
Contributions to master, excluding merge commits



Aug 26, 2018 – Dec 2, 2018

Contributions: Deletions ▾

Contributions to master, excluding merge commits



## Reporte de contribuciones individuales

Integrante	Contribuciones
Diego	<ul style="list-style-type: none"><li>● Diseño e implementación de endpoints del API</li><li>● Diseño e implementación del módulo que recupera ratings.</li><li>● Diseño de visual del servicio web, Resolución de incompatibilidad con Spring-Vaadin, implementación de la carga visual de los posters de películas y la implementación de las sub-ventanas para el login y películas.</li></ul>
Hernán	<ul style="list-style-type: none"><li>● Colaboración en el diseño de la base de datos.</li><li>● Diseño e implementación del módulo de películas.</li><li>● Diseño visual del servicio web, implementación de la barra de búsqueda e mayor velocidad de respuesta, implementación de la funcionalidad de Perfiles en el servicio web, Mejora visual del diseño de las ventanas y la muestra de información.</li></ul>
Javier	<ul style="list-style-type: none"><li>● Implementación del sistema de login.</li><li>● Cambios visuales al servicio web.</li><li>● Diseño de UI para FrontEnd</li></ul>
César	<ul style="list-style-type: none"><li>● Colaboración en el diseño de la base de datos.</li></ul>

## Reporte de avance individual por entrega

Integrante	Primera entrega	Segunda entrega	Tercera entrega
Javier	Casos de uso y su diagrama	Documento anexo en el repositorio que incluía la bitácora	Proceso de desarrollo correspondiente a la documentación
Diego	Escenarios de uso y requerimientos.	Modificación de requerimientos y escenarios de uso.	Edición del video, Resumen de requerimientos y modificación de escenarios de uso
Hernán	Diagrama de clases.	Diagrama de paquetes.	
César		Diagrama de casos de uso	Diagrama de clases

## Reporte de contribución general basada en elementos

Reportes de investigación: RestAPI, SOLID, Vaadin, Spring, SpringBoot  
DataBase, MYSQL y NOSQL.

Diseños: Endpoints, paquetes, base de datos, casos de uso.

Modelos: Movie, genre, Account, PersonalRating, Rating, MovieGenre

Repositorios: AccountRepository, GenreRepository, MovieGenreRepository,  
MovieRepository, PersonalRatingRepository

Servicios: AccountService, GenreService, MovieGenreService, MovieService,  
PersonalRatingService, RatingService

Controladores: Account, Genre, Movie, PersonalRating

Interfaz de usuario: AccountCreation, EditPassWindow, Login, MainView,  
MoviePoster, MovieWindow, ProfileWindow.

Clases útiles: JsonPathUtil, MovieFactory, MovieRaw, PersonalRatingId,  
RatingRaw.

Documentación:

**Objetivos (medibles) de la participación individual en todo el proyecto**

Creación de diseños, modelos, repositorios, servicios, controladores, interfaz de usuario, demás clases y documentación.

El proyecto se encuentra almacenado en: <https://github.com/HerCerM/OOP-movies>