Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

Proyecto Integrador (Propuesta de solución a Situación Problema)

TC1030.311 Programación orientada a objetos

Benjamín Antonio Velasco Guzmán *A01750156*

Índice

1. Introducción

En los últimos años, han proliferado los servicios de streaming de video bajo demanda por ejemplo Netflix, Disney, DC entre otros. Algunos de ellos se especializan por el volumen de videos que proporcionan a sus usuarios mientras que otros se han puesto el reto de mostrar solamente videos de su propia marca. Una versión limitada para apoyar a un futuro proveedor de este tipo de servicios es la que se describe a continuación:

Se quiere trabajar con dos tipos de videos: películas y series. Todo video tiene un ID, un nombre, una duración y un género (drama, acción, misterio).

Las series tienen episodios y cada episodio tiene un título y temporada a la que pertenece.

Nos interesa conocer la calificación promedio que ha recibido cada uno de los videos. Esta califiación está en escala de 1 a 10 donde 10 es la mejor calificación.

El sistema debe ser capaz de:

- Mostrar los videos en general con sus calificaciones
- Mostrar los episodios de una determinada serie
- Mostrar las películas con sus calificaciones

Para dar solución a esta situación, se propone la arquitectura que se describe a continuación.

2. Arquitectura

La arquitectura que dará solución a la situación problema será una arquitectura de tipo cliente-servidor, donde a continuación se describirá a detalle únicamente la arquitectura del cliente, en este caso la aplicación escrita en Python 3.7. El servidor no se describirá a detalle porque no es el objetivo de este documento, sin embargo en los apéndices se describen los endpoints y sus parámetros.

2.1. Diagrama de clases

A continuación se presenta el diagrama de clases creado en StarUML (ver referencias). Dicho diagrama se acopla a las convenciones para nombrar variables y métodos en Python, por lo que puede discrepar un poco con lo especificado por el estándar UML.

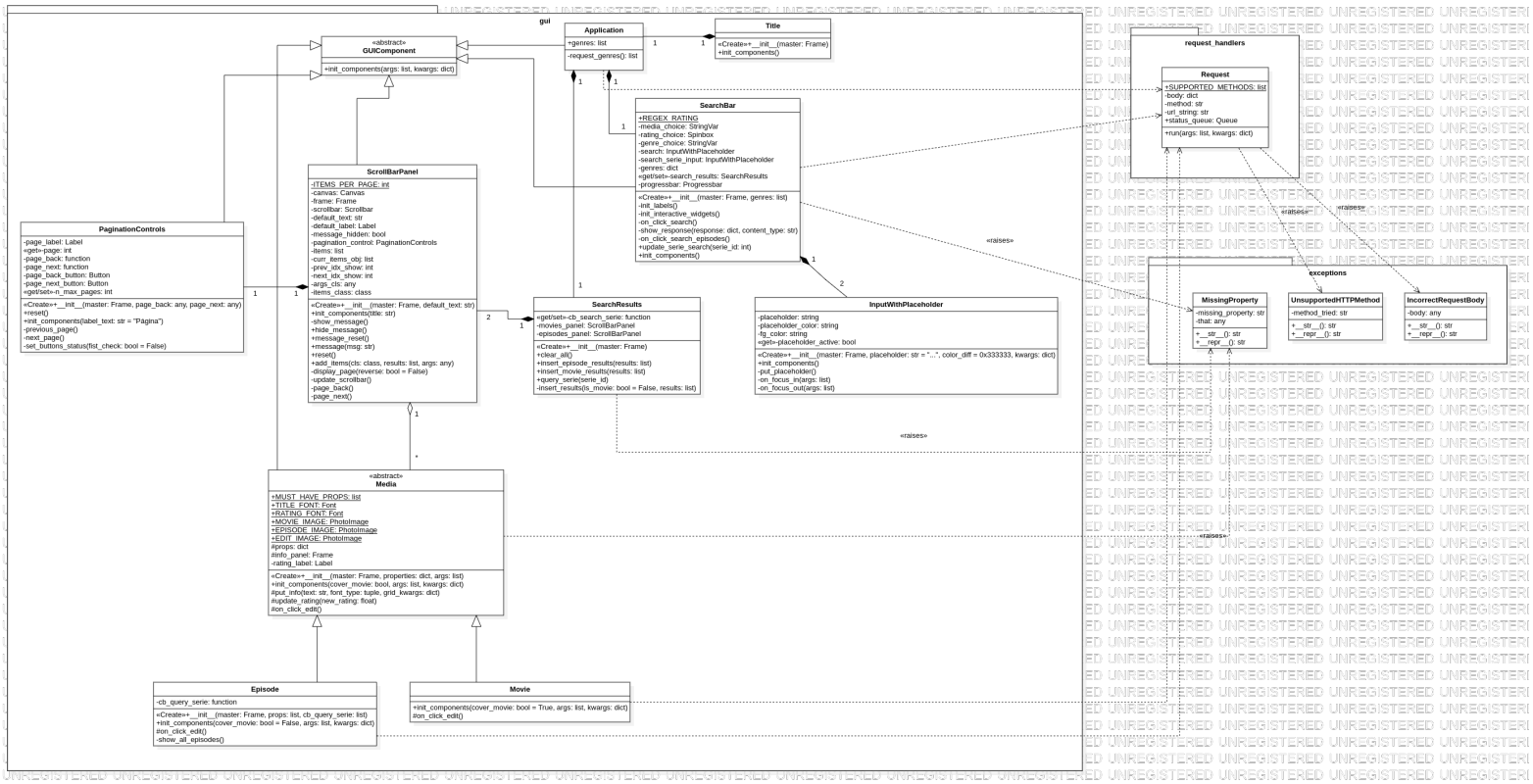

Figura 1: Diagrama de clases para el front-end

2.1.1. Paquete: gui

Este paquete contiene todas las clases que construyen a la interfaz gráfica desde la cual el usuario interactúa con el servicio. La librería utilizada para crear la interfaz gráfica es tkinter (ver referencias), se eligió dicha librería porque ya está contenida en python 3 y no es necesario instalarla manualmente, además de que ya se contaba con conocimiento previo sobre cómo usarla.

Todas las clases de este paquete, excepto Episode, Movie, GUIComponent, InputWithPlaceholder y Application heredan directamente de la clase Frame proveída por tkinter, en el último caso, la clase Application hereda de la clase Tk también proveída por la misma librería. Todos aquellas clases utilizadas pero no programadas, es decir que las provee directamente Python 3 o una librería externa no se colocaron evidentemente en el diagrama.

Clase GUIComponent

Esta realmente es una interfaz puesto que solo declara más no define el comportamiento de su método `init_components`, por lo que cada clase que implemente (herede) esta interfaz deberá implementar dicho método, además dichas clases pueden además sobreescribir o sobrecargar dicho método. Este método difiere del constructor de una clase en que éste se encargará de instanciar los atributos de la clase que no sean componentes gráficos o que sean objetos parcialmente sencillos (como una lista vacía o valores por defecto), mientras que el método `init_components` se encargará de instanciar los componentes de la interfaz gráfica, así como de computar los valores de los atributos de la clase que sean más complejos (como llenar la lista vacía).

Clase Application

Esta clase es la ventana principal y hereda de tkinter.Tk, sobreescribe el método `init_components` que se encargará de instanciar un objeto Request que hará una petición al back-end para obtener todos los géneros disponibles. Esto se realiza porque los géneros son un parámetro de búsqueda para el usuario, por lo que la instancia de SearchBar de este objeto deberá tener conocimiento de los géneros para desplegarlos en un dropdown. Además la cantidad de géneros es mucho menor a la cantidad de películas o episodios, por lo que cargar los géneros ocupa menos recursos que las otras entidades. Aunado a esto, la lista de géneros es indispensable para hacer la búsqueda (entre otros campos, que se pueden construir on-the-fly para ahorrar recursos, y se platicará sobre ello en la clase SearchBar).

Clase Title

Si bien, la creación de esta clase no era muy necesaria, se decidió crear por simplicidad y legibilidad del código y por tener un diseño modular más sencillo. Dicha clase se encarga de crear los componentes que generan el título junto con el nombre del autor (Benjamín Antonio Velasco Guzmán A01750156).

Clase SearchBar

Clase encargada de contener y desplegar los cuatro filtros para realizar una búsqueda:

- Tipo de contenido: video, episodio o ambos
- Nombre del contenido: Caracteres con los que inicie el contenido que se desea buscar (e. g. "The D", si se desea buscar "The Dark Knight")
- Rating mínimo: El rating mínimo (números del 0 al 10 con saltos de 0.5) que debe tener el contenido que se desea buscar.
- Género: El género del contenido que se desea buscar.

Esta clase también es la encargada de manejar todos los eventos relacionados a la búsqueda, es decir se encarga de instanciar un objeto Request para realizar la petición de búsqueda en el back-end en cuanto se da click al botón de "Buscar", y con los resultados recibidos notifica al objeto SearchResults instanciado por Application para finalmente mostrar los resultados.

En esta clase también se lleva a cabo la validación de la entrada del usuario, específicamente del Rating mínimo, que es un campo de texto que potencialmente pueden recibir cualquier texto aunque sólo los números enteros o decimales con sólo un decimal son recibidos.

No se lleva a cabo la validación de otros campos porque no son campos que puedan recibir cualquier entrada, son dropdowns con valores válidos predefinidos.

Clase InputWithPlaceholder

Extiende de tkinter.Entry, y se encarga de colocar un placeholder en el input simplemente por estética, esta clase no es lógicamente indispensable. Naturalmente esta clase maneja todos los eventos de focus in/out sobre el input para remover o colocar el placeholder, junto con el manejo de colores para diferenciar el placeholder del texto insertado.

Clase SearchResults

Clase encargada de contener a dos paneles que contendrán a su vez los resultados de las películas y episodios resultantes (de ahí que se requieran dos paneles). Esta clase al igual que Title no tiene mucho funcionamiento pero es útil para separar lógicamente el código y servir de contenedor.

SearchResults

Esta clase se encargará de instanciar a los objetos correspondientes de tipo Media (Episode o Movie), así como de destruir a los objetos instanciados cuando se cambie de página e instanciar los correspondientes de la nueva página. De la misma forma esta clase con ayuda de la clase PaginationControls se encargará de manejar los eventos de cambio de página.

Clase PaginationControls

Clase encargada de contener un panel con los controles para la paginación y navegación sobre los resultados mostrados en un ScrollbarPanel. También se encarga de recibir los eventos de cambio de página, y ejecuta un callback previamente definido en la instancia del ScrollbarPanel que contiene a la instancia de esta clase para que se puedan mostrar resultados diferentes para cada página.

Clase Media

Clase abstracta que contiene muchos métodos útiles para desplegar un resultado de una búsqueda independientemente de si es episodio o película, por ello contiene varios atributos con modificador de acceso tipo protected, así como otros métodos que pueden ser sobreescritos por sus clases hijas. Dado que cualquier contenido, sea película o episodo contiene las siguientes propiedades:

- rating
- duration
- name

Que son las que esta clase toma en cuenta para desplegar la información de cualquier tipo de contenido, si se desea agregar más contenido, éste deberá contenerse en la clase hija y para desplegarlo en la interfaz gráfica se proporciona el método `put_info`.

Naturalmente este diseño permite bastante la reutilización de código, así como la generalización de tipos.

Episode

Clase que hereda de Media, se encarga de mostrar y contener todos los atributos exclusivos de un episodio:

- ID de la serie a la cual pertenece el episodio
- Nombre de la serie a la cual pertenece el episodio
- Número de episodio
- Número de temporada a la cual pertenece el episodio

Sobreescribe el método `on_click_edit` para editar el rating del episodio y enviar la petición al backend. Para ello crea una instancia de Request. Esto sucede claramente después de pedirle al usuario el nuevo rating para el episodio mediante un cuadro de diálogo que también tiene validación para sólo ingresar números flotantes entre 0 y 10.

De igual forma tiene su método propio `show_all_episodes` para que al darle click al botón "Ver serie"se muestren todos los episodios de la serie. Evidentemente esta clase también crea y maneja el botón previamente mencionado

Movie

Clase que hereda de Media, una película al tener los mismos atributos que Media, no existen muchas implementaciones o sobreescrituras de métodos, salvo el método `on_click_edit` que crea una instancia de Request y envía una petición al back-end para actualizar el rating de la película. Esto sucede claramente después de pedirle al usuario el nuevo rating para el episodio mediante un cuadro de diálogo que también tiene validación para sólo ingresar números flotantes entre 0 y 10.

2.1.2. Paquete: request_handlers

Este paquete, a pesar de solamente contener una clase se decidió separar por simplicidad y un mejor control de los archivos, además de que el uso de la única clase Request que contiene es de propósito general, se puede usar en alguna clase dentro del paquete gui o fuera.

Clase Request

Clase que hereda de Thread, y se encarga de enviar peticiones HTTP al backend. Como se mencionó, esta clase intenta ser de propósito general por lo que el método de la petición, payload o body, ruta e incluso host y puerto son definidos por quien instancie un objeto de esta clase.

Al tener en cuenta que esta clase será usada dentro del paquete gui y mientras el hilo de ejecución principal esté principalmente ocupado por la interfaz gráfica, se decidió que esta clase heredara de Thread para evitar interferir y bloquear el hilo principal, por esta naturaleza de ejecución paralela también se utilizó mutex (mutual exclusion) para modificar el atributo status_queue que contiene el último status reportado, de forma tal que quien use esta clase pueda saber el estatus de su petición mediante la obtención del primer valor en la queue (recordar que la cola o queue funciona con el principio FIFO (First In First Out), además de que python guardará la queue en heap, lo que hace posible que tanto Request como la clase que usa Request puedan acceder al mismo objeto)

Las anteriores consideraciones pudieron no haber sido tomadas en cuenta dados los bajos niveles de concurrencia y trabajo que puede llegar a tener la aplicación (de hecho, de todas formas se bloquea el hilo principal de la aplicación mientras se recibe la respuesta del back-end pues no es necesario por el momento que se ejecute estrictamente en otro hilo la petición), pero se decidió implementar por escalabilidad y buena práctica de ingeniería de software.

Por último, esta clase puede lanzar excepciones definidas en el paquete exceptions si el backend no soporta algún método (HTTP-MethodUnsupported) o si el payload o body de la petición no es correcto (IncorrectBodyRequest)

2.1.3. Paquete: exceptions

Contiene las excepciones no estándar definidas por mí. Cabe recalcar que no son las únicas excepciones que puede lanzar algún método o función, de hecho en el paquete gui se lanzan excepciones predefinidas también.

MissingProperty

Excepción útil cuando hace falta o tiene valor None algún atributo de la clase, usualmente esto se debe a una mala inicialización del objeto o debido a que no se invocó algún método que coloque valores correctos a los atributos de una clase (e. g. `init_components`)

UnsupportedHTTPMethod

Excepción estrictamente relacionada a la clase Request, pues este error es lanzado cuando se intenta realizar una petición al back-end con un método no soportado.

IncorrectBodyRequest

Nuevamente, es una excepción estrictamente relacionada a la clase Request, pues este error es lanzado cuando se intenta enviar un payload o body que no sea `dict` o herede de esta clase (la conversión y correcta codificación del objeto es manejada por el paquete requests)

3. Ejemplos de ejecución

Para esta sección se mostrarán imágenes de la aplicación en ejecución en un sistema operativo con el tema para las interfaces gráficas Tk modificado, por ello se observa en tonalidades oscuras, porque tkinter adpotará los colores nativos del sistema operativo, si se visualizara en Windows, probablemente la interfaz gráfica tendría un color blanco.

A continuación se muestra la presentación de la aplicación

Figura 2: Presentación de la aplicación

Es importante aclarar que las imágenes a continuación muestran la información del episodio o película al lado de otra imagen, la cual fue colocada simplemente por fines estéticos y no representa nada, también fue colocada por términos de escalabilidad por si en determinado momento se llegara a requerir la imagen real del episodio, serie o película.

3.1. Flujo básico

A continuación se muestran imágenes de la ejecución de la aplicación y funcionamiento del flujo básico (cómo debería comportarse la aplicación si todos los datos ingresados son correctos)

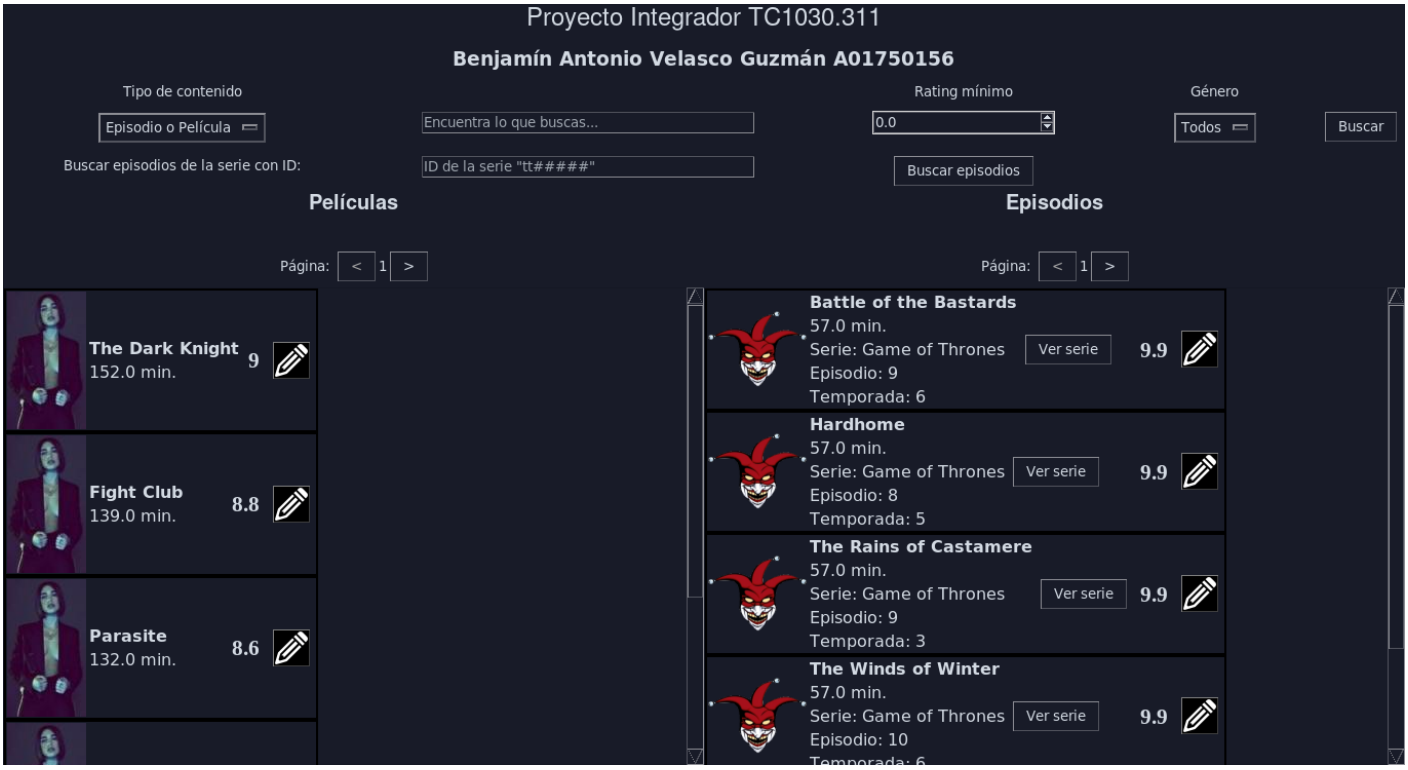

Figura 3: Búsqueda de cualquier película o episodio, con cualquier nombre, cualquier género y rating mínimo de 0

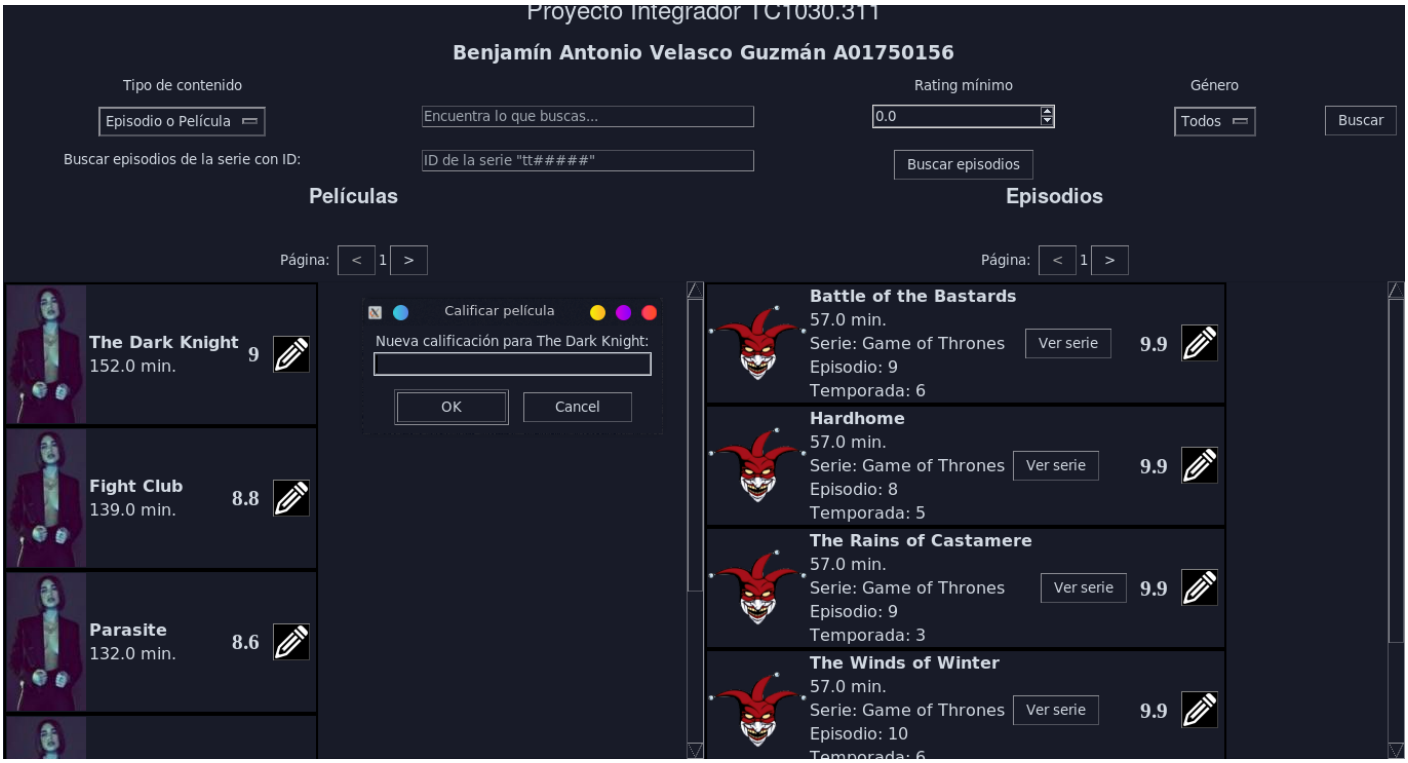

Figura 4: Cuadro de diálogo para modificar el rating de una película

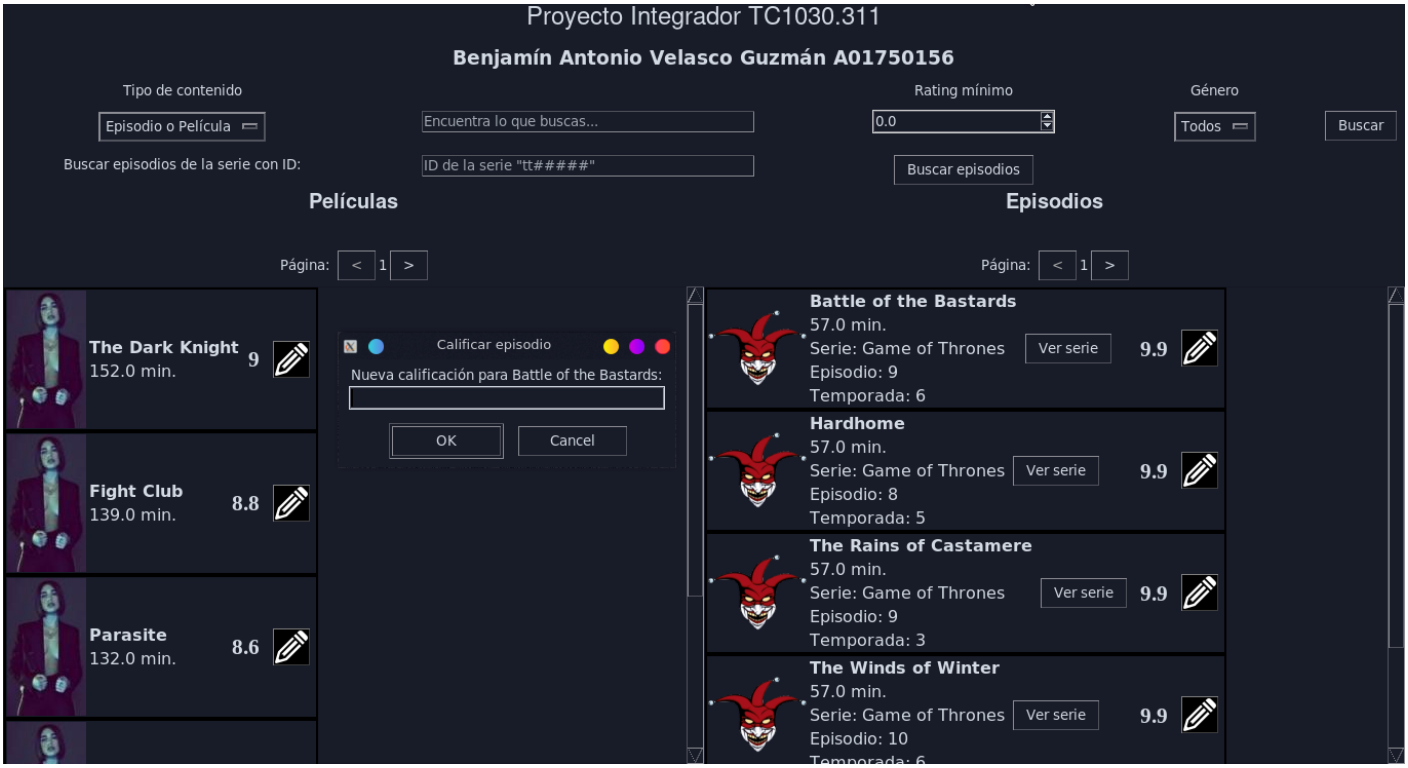

Figura 5: Cuadro de diálogo para modificar el rating de un episodio

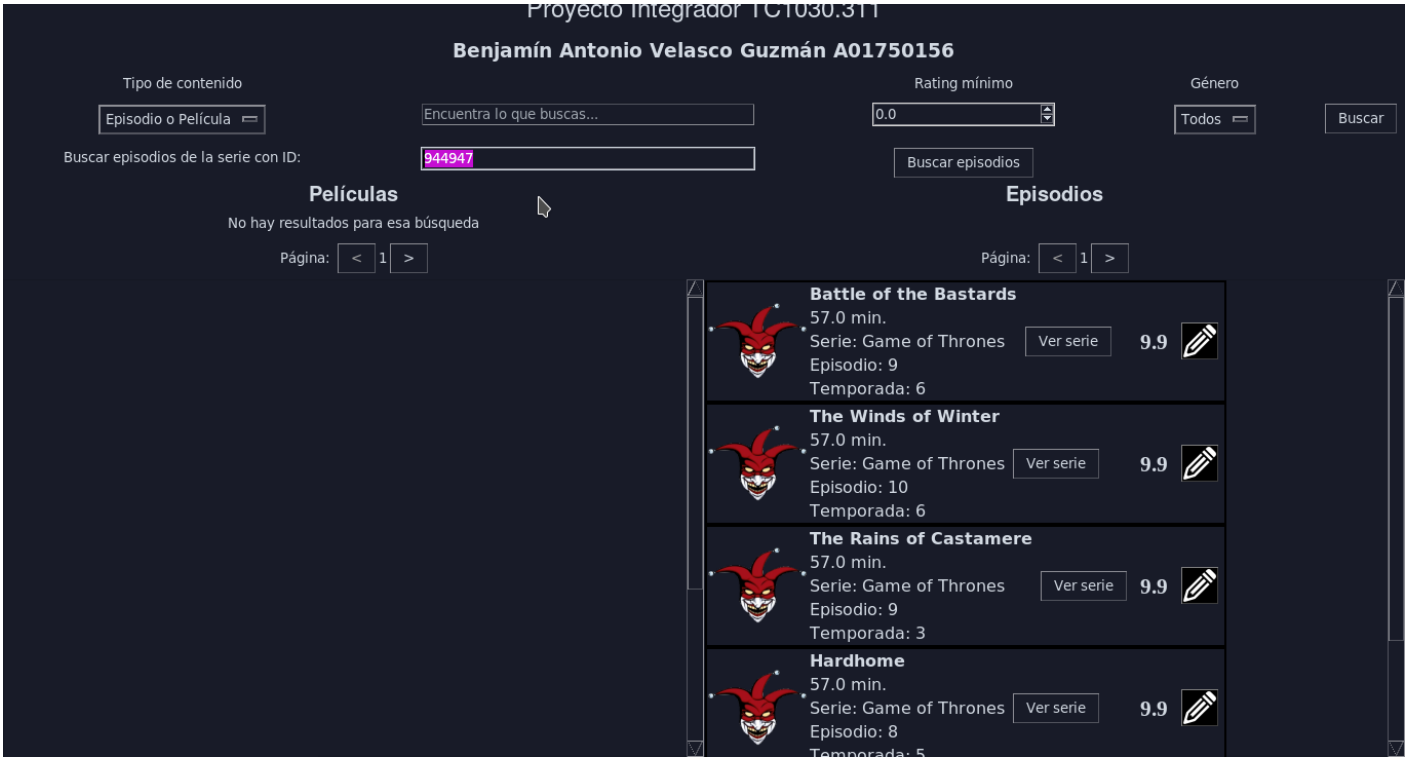

Figura 6: Búsqueda de episodios de una serie

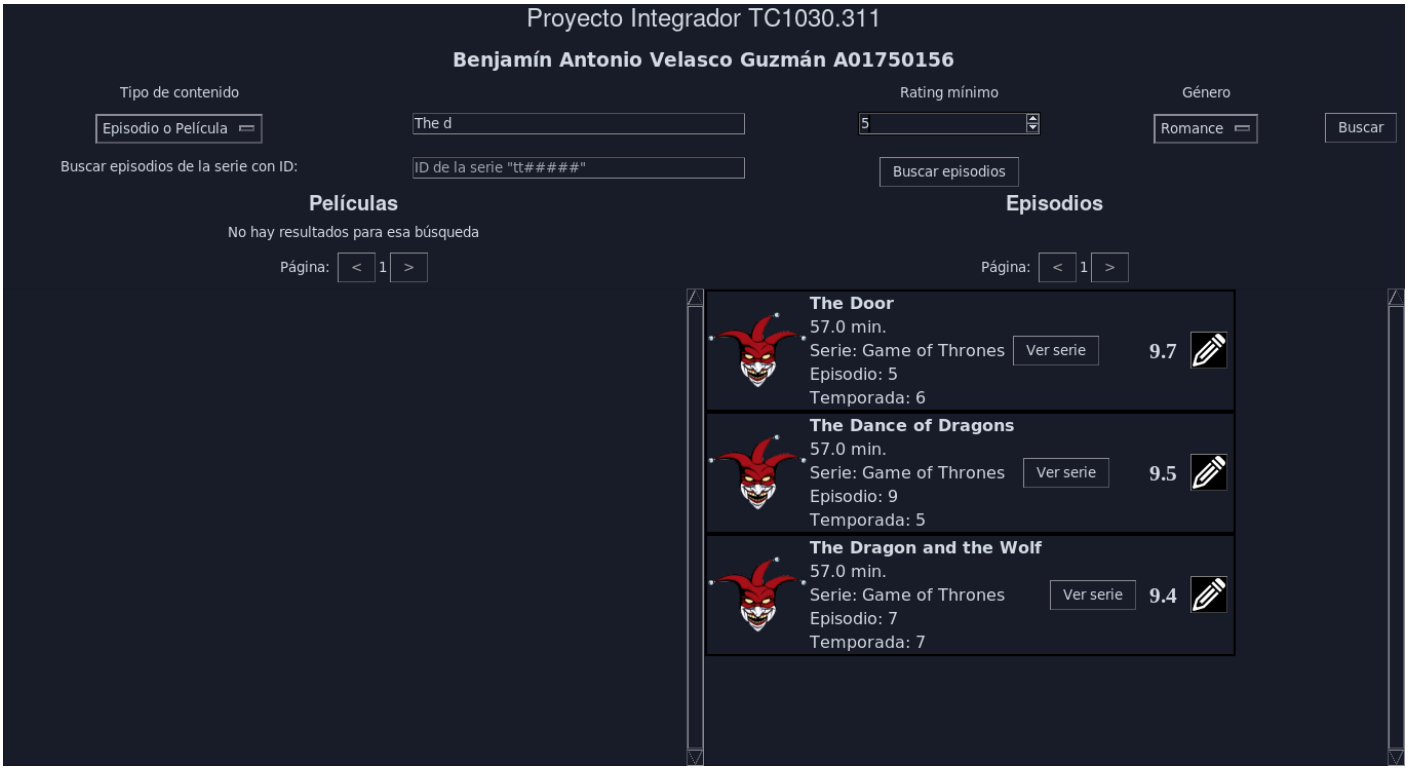

Figura 7: Búsqueda de cualquier película o episodio, que empiece con los caracteres "The d", con rating mínimo de 5 y de género Romance

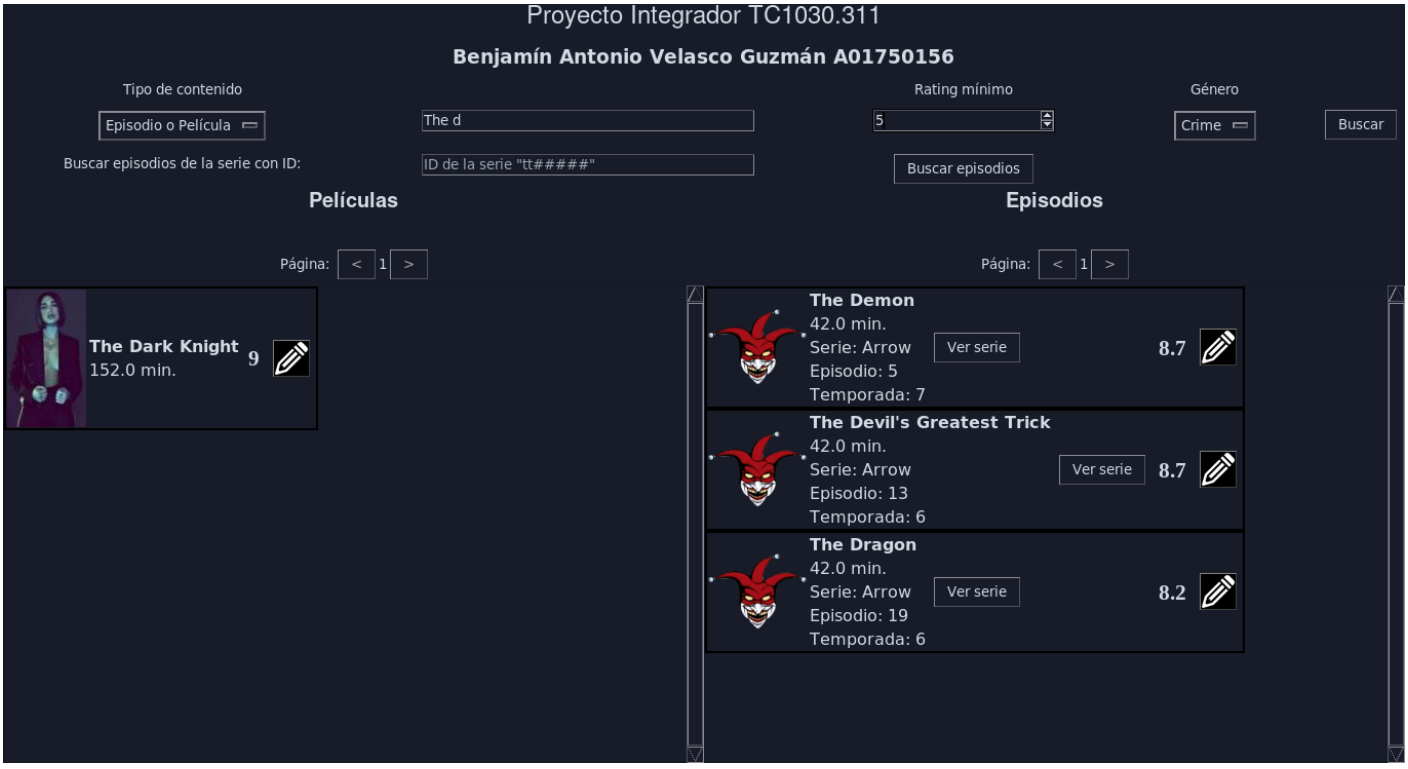

Figura 8: Búsqueda de cualquier película o episodio, que empiece con los caracteres "The d", con rating mínimo de 5 y de género Crime

Nótese la diferencia entre los valores de los parámetros de búsqueda y los resultados entre las dos últimas imágenes.

3.2. Flujos de excepción

A continuación se muestran imágenes de la ejecución de la aplicación y funcionamiento cuando existe algún flujo de excepción (cómo debería comportarse la aplicación si algún valor no es válido)

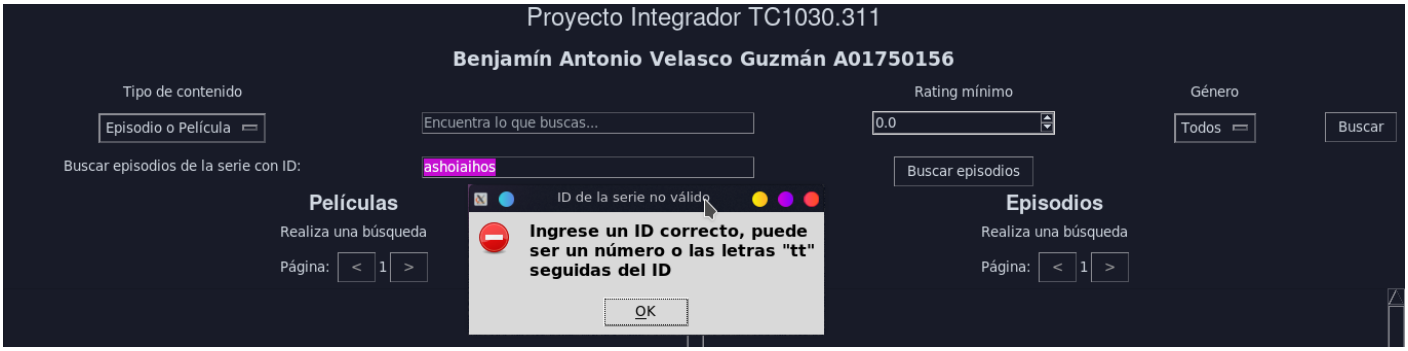

Figura 9: Mensaje de error si al buscar una serie se ingresa un ID no válido

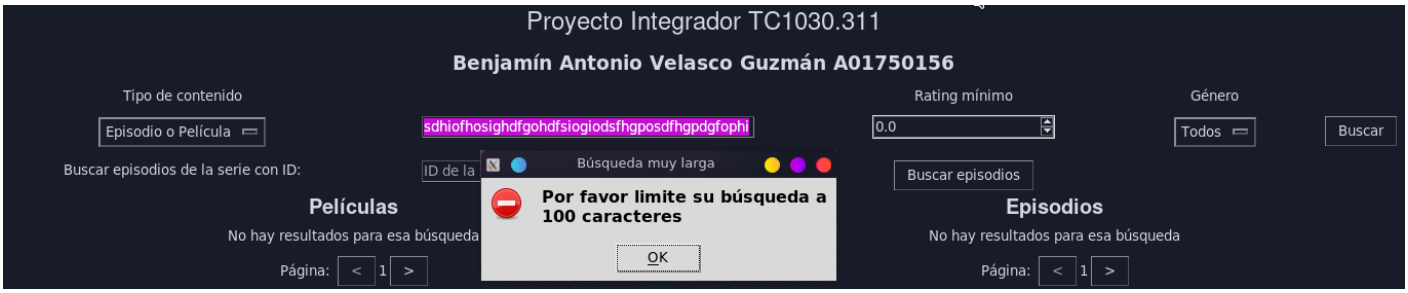

Figura 10: Mensaje de error si al buscar contenido por nombre se ingresan más de 100 caracteres

Figura 11: Mensaje de error si al buscar contenido por rating mínimo se ingresan valores que no sean números enteros o flotantes

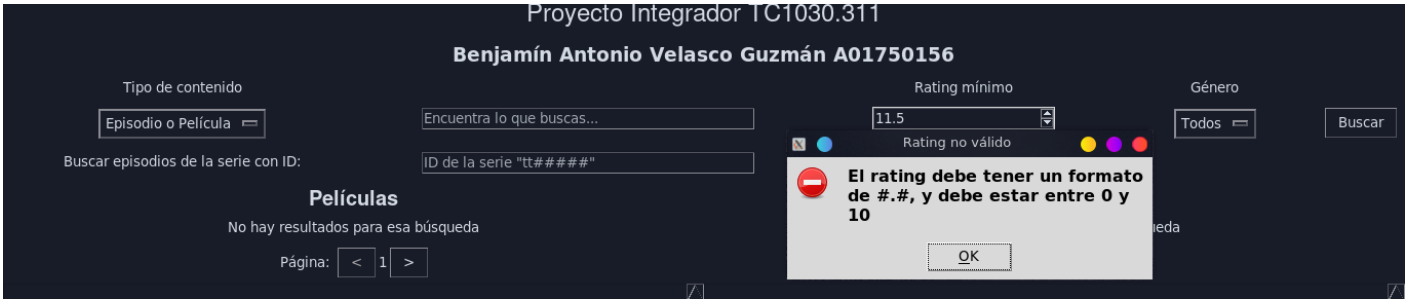

Figura 12: Mensaje de error si al buscar contenido por rating mínimo se ingresan valores fuera de rango

4. Casos que harían que el proyecto dejará de funcionar

4.1. Servicio del back-end no activo

Evidentemente si el servidor no está activo y con la capacidad para recibir peticiones el programa no podrá funcionar y tendrá una salida parecida a la siguiente.

```
Obteniendo los géneros disponibles de http://127.0.0.1:8080/genres
Cargando...
Algo salió bastante BASTANTE mal
Verifica tu conexión a internet
O, que el servidor esté prendido y a la escucha
```

Figura 13: Salida de la aplicación cuando el backend no está activo

4.2. Otros casos menores

Fuera de los casos donde deja de funcionar la aplicación, existen casos en los que la aplicación no funciona correctamente pero sigue funcionando, tales son los casos en los que la visualización por página se desacomoda momentaneamente al hacer scroll, o que no se muestren los detalles completos de una película o episodio si su nombre es bastante largo. De igual forma existe un pequeño bug en la visualización de los resultados por paginación si el número de resultados no es múltiplo de 5, al colocarse en la última página, supongamos que por no ser múltiplo de 5 la cantidad de resultados que se muestran son simplemente 3 y después se regresa a la primera página en la cual aparecerán en vez de 5 resultados, 3 resultados. Este es un error de visualización, sin embargo no representa ninguna pérdida de datos y todos los resultados se pueden seguir viendo, para solucionar dicho problema se tendría que modificar el incremento o decremento de una variable que controla la paginación en la clase PaginationControls.

Por otro lado, normalmente no se podrá ejecutar la aplicación si tanto para el backend como para el frontend (la aplicación de python) no se tienen instaladas las dependencias requeridas, por lo que será necesario instalarlas simplemente con `pip install -r <ruta hacia el archivo requirements.txt>`, dicho archivo `requirements.txt` contiene las dependencias y versiones requeridas para ejecutar el frontend.

Asimismo, naturalmente pueden existir fallas en la aplicación debidas no al código de la aplicación directamente, sino al código externo utilizado adentro de este mismo proyecto, por ejemplo, en la clase Request donde se utiliza un queue, es posible que existan fallos a pesar de que se programó de forma que se eviten tales inconvenientes, pero, de la misma documentación de Python se conoce que los métodos no pueden funcionar como uno espera, e. g. " If empty() returns True it doesn't guarantee that a subsequent call to put() will not block. Similarly, if empty() returns False it doesn't guarantee that a subsequent call to get() will not block.", obtenido de la documentación de python (ver referencias), y lo mismo aplica para el backend.

5. Conclusión personal

Concluyo que para el desarrollo de aplicaciones gráficas Python (y tkinter) no es buena idea, por lo menos para aplicaciones que requieran una interfaz bien diseñada y con elementos robustos, pues hay otras alternativas mucho más robustas que ofrecen prácticamente lo mismo, sin embargo, creo que para la parte lógica del programa es bastante bueno pues la escritura de su código es similar a lo que sería el pseudo código (en inglés, claro), además de que tiene muchas funcionalidades ya incluidas en el propio lenguaje, pero esto también conlleva mucho más trabajo por parte del desarrollador si se quiere hacer una aplicación rápida y robusta pues por un lado Python es interpretado y a diferencia de otros lenguajes que también lo son (e. g. JavaScript) Python es mucho más lento y además se debe tener mucho cuidado con los tipos lo cual puede llegar a dificultar un poco más la escritura tanto del código como la de su documentación.

Todos los endpoints que se tratarán a continuación pueden retornar status codes status codes `4XX` o `5XX`, los que sean `20X` status codes no tendrán información extra, sólo el status code. La siguiente documentación para los endpoints aplica para el flujo básico.

6. Apéndice: GET endpoints

6.1. /genres

Retorna todos los géneros registrados en la base de datos

6.1.1. Parameters

NONE

6.1.2. Response

Un `JSON` con la siguiente estructura:

```
{
        "genres": Array<{
                "id": number,
                "genre": string
        }>
}
```

6.2. /media

Retorna información sobre un tipo de contenido, ya sea películas, episodios o ambos.

6.2.1. Parameters

Un `JSON` con la siguiente estructura:

```
{
        type_of_content: string, //required, enum: ["movie", "episode", "video"]
        search: string,
        min_rating: number,
        genre: number
}
```

Si `min_rating` o `genre` son `undefined`, los valores default son 0. Si `search` es `undefined` o simplemente ” o cualquier otro valor que en Javascript tenga un valor booleano de `false`, entonces, este campo es ignorado.

6.2.2. Response

Un `JSON` con la siguiente estructura:

```
{
        "movies": Array<{
                "id": number,
                "rating": number,
                "duration": number,
                "year": number,
                "name": string,
                "cover": string | null
        }>,
        "episodes": Array<{
                "id": number,
                "serie_id": number,
                "serie_name": string,
                "n_episode": number,
                "n_season": number,
                "rating": number,
                "duration": number,
                "name": string,
                "cover": string | null
```

```
        }>
}
```

Si solamente `movies` es pedido, entonces la propiedad .ºpisodes"no aparecerá y viceversa, si los dos son pedidos, entonces los dos aparecerán.

6.3. /episodes

Retorna todos los episodios de una determinada serie

6.3.1. Parameters

Un `JSON` con la siguiente estructura:

```
{
        "serie_id": number
}
```

6.3.2. Response

```
{
        "episodes": Array<{
                "id": number,
                "serie_id": number,
                "n_episode": number,
                "n_season": number,
                "rating": number,
                "duration": number,
                "name": string,
                "cover": string | null
        }>
}
```

7. Apéndice: PUT endpoints

7.1. /media-rating

Cambia el rating de algún episodio o película según sea el caso

7.1.1. Parameters

Un `JSON` con la siguiente estructura:

```
{
        "type_of_content": string, //required, enum: ["movie", "episode"]
        "id": number, // required
        "new_rating": number // required
}
```

7.1.2. Response

NONE

8. Referencias

- http://staruml.io/
- https://docs.python.org/3.9/library/tk.html
- http://effbot.org/tkinterbook/
- https://docs.python.org/3/library/threading.html
- https://github.com/ajv-validator/ajv