WATCHTHIS

Memoria de Proyecto Final

Índice

Introducción

Wachthis es un servicio web especializado en series de televisión, películas y animes, a través de cualquier dispositivo conectado a internet. Contamos con un catálogo muy amplio y variado para que se pueda encontrar cualquier título que busques.

Este contenido está dividido por secciones distribuidas en tres páginas diferentes que muestran ejemplos de películas disponibles para poder visualizar. Además encontrarás una sinopsis del argumento del contenido así como de su reparto.

Este sistema dispone de un buscador para poder saber rápidamente si el contenido solicitado se encuentra disponible en la plataforma, además tenemos.

Una de las cosas más destacables de esta aplicación es que te notifica mediante correo electrónico los últimos lanzamientos según la/s categoría/s que más te interesen mandando el nombre del título lanzado y su género.

Para poder disfrutar de este servicio lo único que hay que hacer es ir a la pestaña de Sign Up e ingresando su nombre de usuario correo y contraseña podrá disfrutar completamente de todo el contenido de WachThis de forma totalmente gratuita

Cambios encontrados durante el desarrollo

Comparado con el resultado final, nuestras expectativas a principio del desarrollo fueron demasiados altas. Por ello, nos hemos centrado en conseguir las funciones que hiciesen de la aplicación una aplicación sólida, pues ofrece los contenidos que teníamos propuestos. Sin embargo, somos conscientes de que la aplicación

Aquí tenemos en rojo las funciones que no acabaron implementándose:

- Requisitos Funcionales

RF1	El sistema debe validar al usuario.
RF2	El sistema debe permitir darse de alta.
RF3	El sistema debe permitir darse de baja.
RF4	El sistema debe permitir buscar contenido.
RF5	El sistema debe permitir dirigirse a la pantalla principal.
RF6	El sistema debe permitir dirigirse al catálogo de películas.
RF7	El sistema debe permitir dirigirse al catálogo de series.

RF8	El sistema debe permitir dirigirse al catálogo de anime.
RF9	El sistema debe permitir dirigirse a las listas.
RF10	El sistema debe permitir dirigirse a un panel notificaciones.
RF11	El sistema debe permitir acceder a la notificación.
RF12	El sistema debe permitir acceder al perfil de usuario.
RF13	El sistema debe permitir acceder a los contenidos (películas, series, animes) mejor valorados en general.
RF14	El sistema debe permitir acceder a las películas mejor valoradas.
RF15	El sistema debe permitir acceder a las series mejor valoradas.
RF16	El sistema debe permitir acceder a los animes mejor valorados.
RF17	El sistema debe permitir acceder a la película seleccionada.
RF18	El sistema debe permitir acceder a la serie seleccionada.
RF19	El sistema debe permitir acceder al anime seleccionado.
RF20	El sistema debe permitir acceder a la lista seleccionada.
RF21	El sistema debe permitir elegir la temporada de la serie.
RF22	El sistema debe permitir elegir el capítulo de la serie.
RF23	El sistema debe permitir elegir la temporada del anime.
RF24	El sistema debe permitir elegir el capítulo del anime.
RF25	El sistema debe permitir acceder a la lista de enlaces.
RF26	El sistema debe permitir acceder al servicio de streaming especificado.
RF27	El sistema debe permitir acceder al enlace.
RF28	El sistema debe permitir reportar un error de enlace.
RF29	El sistema debe permitir seleccionar la causa.
RF30	El sistema debe permitir añadir un comentario al informe reportado.
RF31	El sistema debe permitir enviar el informe.
RF32	El sistema debe permitir crear listas personalizadas.
RF33	El sistema debe permitir borrar listas personalizadas.
RF34	El sistema debe permitir guardar listas personalizadas.
RF35	El sistema debe permitir añadir contenido en las listas.
RF36	El sistema debe permitir borrar contenido de las listas.
RF37	El sistema debe permitir buscar por género.
RF38	El sistema debe permitir buscar por año.
RF39	El sistema debe permitir buscar por valoración.
RF40	El sistema debe permitir acceder a la configuración del perfil.
RF41	El sistema debe permitir aplicar el tema oscuro.
RF42	El sistema debe permitir cambiar la contraseña.

RF43	El sistema debe permitir cambiar el nombre de usuario.
RF44	El sistema debe permitir que las notificaciones te lleguen por correo.
RF45	El sistema debe permitir cerrar la sesión.
RF46	El sistema debe permitir puntuar películas.
RF47	El sistema debe permitir puntuar series.
RF48	El sistema debe permitir puntuar animes.
RF49	El sistema debe permitir dar like al enlace.
RF50	El sistema debe permitir dar dislike al enlace.

En resumen, no hemos incluido los informes ni el sistema de votos, ya que nuestros links redireccionan a una página con estas mismas funcionalidades. Quisimos centrarnos en poder ofrecer todo el contenido que teníamos pensado, pero de una forma más sencilla.

Planes futuros

Aunque la aplicación no cuenta con todas las funcionalidades pensadas, esto no significa que no tengamos pensado agregarlas o incluir nuevas. De hecho, estas son algunas de las ideas que podrían mejorar nuestra aplicación:

- **Nuevo Usuario: Administrador.**
  Ahora mismo la aplicación se maneja desde el código fuente. Sin embargo, nos gustaría poder validar a usuarios con rol de administrador, para que manejar el contenido sea más cómodo y rápido. Para ello, crearíamos una nueva jerarquía de usuarios en nuestra base de datos y modificaríamos nuestras operaciones de inicio de sesión.

- **Listas.**
  Nos gustaría darles a los usuarios la opción de guardarse listas personalizadas con el contenido que más gusten. Creando un nuevo elemento lista, que forme parte del usuario y arreglando la interfaz, esto es más que factible.

- **Interfaz ajustada al usuario.**
  Otro de nuestros objetivos sería conseguir datos de actividad del usuario, para poder facilitarle los contenidos que más se ajustan a su perfil, recomendándoselos en la pantalla principal y así evitar a los usuarios largos tiempos de búsqueda entre nuestras películas.

Patrones de Diseño

Para el desarrollo del proyecto se ha llevado a cabo la implementación de tres patrones de diseño, los cuales se han aplicado para la parte de Backend ya que se encargaba de la administración y creación de los modelos de datos, así como su gestión para los propios usuarios de la aplicación:

• Factory Method

Flask, la librería implementada para el desarrollo del backend en Python, cuenta con un patrón llamado "Factory" el cual puede ser útil para crear las instancias de "app" desde Flask o manejar sus "plugins" en casos necesarios (enrutamientos, decoradores…).

El patrón "Factory" o "Application Factory" nos ofrece una serie de ventajas para el propio desarrollo de la aplicación, así como una posible escalabilidad en caso de extender y aumentar la complejidad del proyecto:

- Detiene / Evita las importaciones circulares.
- Mejora y facilita la configuración de la aplicación, así como sus pruebas y los diferentes entornos.
- Permite un mejor entendimiento del proyecto, teniendo una mejor organización de clases y componentes, así como legibilidad del código.

Con este patrón hemos aplicado el término "Factory" al cual le damos una serie de datos o dependencias, obteniendo un producto totalmente configurado como salida:

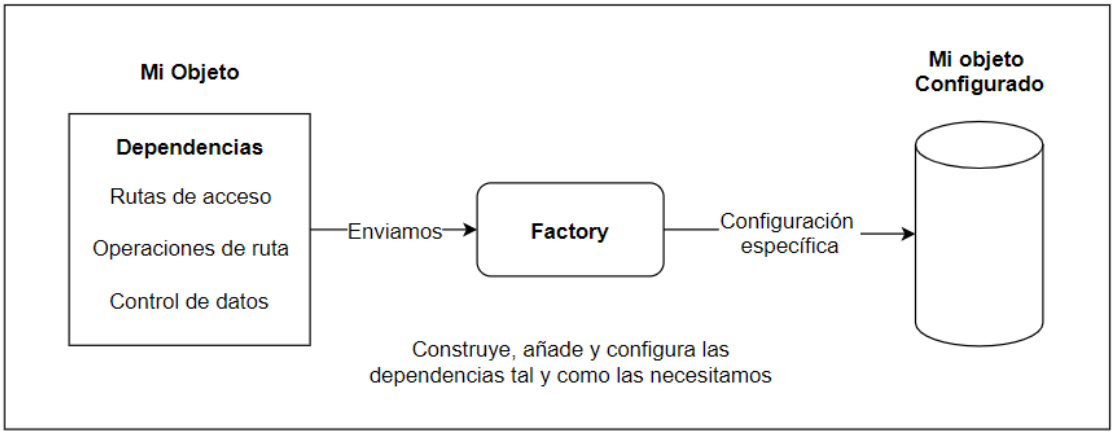


Diagrama de Factoría

En este caso nuestro objeto va a ser "app", la aplicación de Flask la cual vamos a necesitar configurar para adaptar las dependencias acordes a los modelos de datos creados, así como las rutas de la página, contando con las posibles operaciones CRUD.

Para ello, con el fin de poder configurar el objeto será necesario importarlo en cada una de las clases que quieran modificar y tratar con él, en este caso, con el fin de evitar la importación cíclica o circular vamos a crear unas factorías en las cuales vamos a pasar nuestro objeto como parámetro:

- Clase app.py: Clase principal de la aplicación, en la cual se instancia el objeto y se importan las clases que lo configuran.

- Database Models: Contiene todos los modelos de datos.

- Database Operations: Contiene todas las factorías, aplicando una configuración específica para cada una de las dependencias que se contemplan.

Para la configuración de nuestro objeto se sigue el siguiente flujo de ejecución:

1. Instanciación del objeto “app” en “app.py”

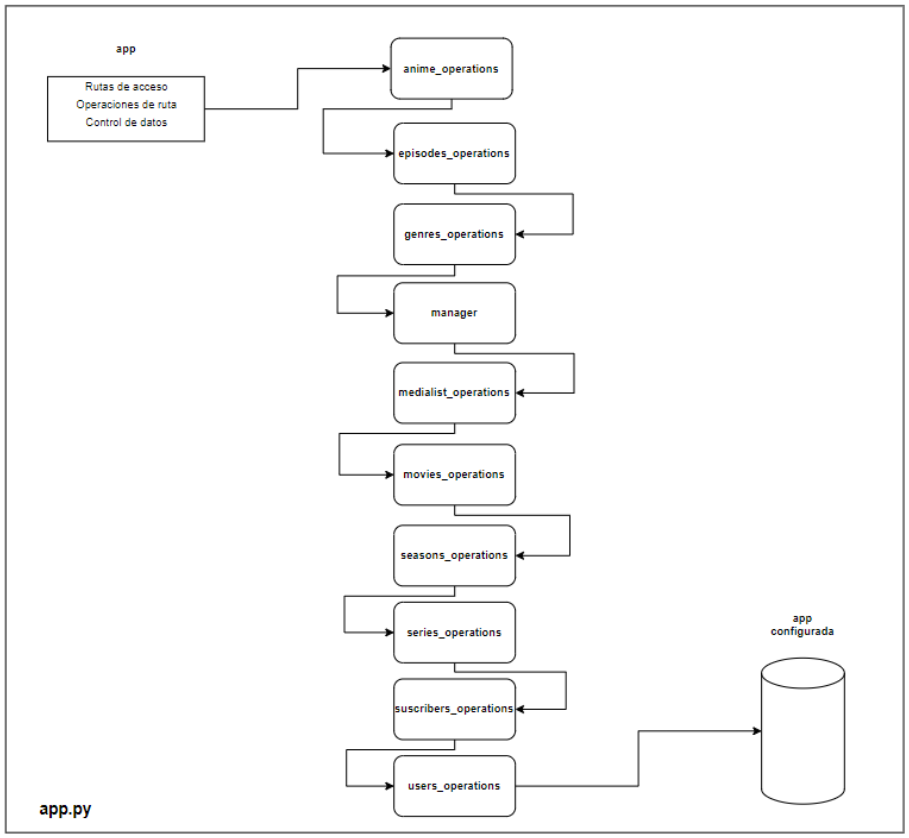
2. Inicialización de los modelos de base de datos para nuestro objeto “app”

3. Configuración del objeto para todos y cada uno de los modelos de datos, inicializando el método “init” de cada uno de ellos agregando cada una de las operaciones a nuestra aplicación.

4. Ejecución de la aplicación

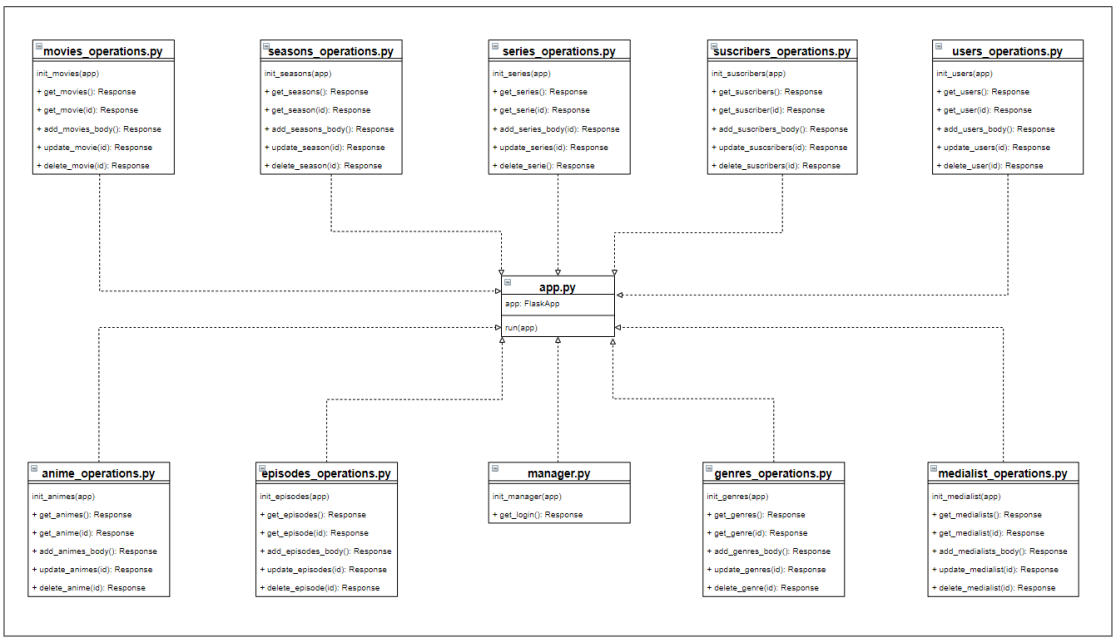
```
33      # Initialize db and factory routing
34      initialize_db(app)
35      episodes_operations.init_episodes(app)
36      movies_operations.init_movies(app)
37      series_operations.init_series(app)
38      seasons_operations.init_seasons(app)
39      users_operations.init_users(app)
40      medialist_operations.init_medialists(app)
41      anime_operations.init_animes(app)
42      genres_operations.init_genres(app)
43      suscribers_operations.init_suscribers(app)
44      manager.init_manager(app)
45
46      app.run()
47
```

Código de ejecución “app.py”



Ciclo de ejecución "app.py"

Con esta implementación conseguimos desacoplar cada una de las operaciones para los modelos de datos implementados, así como los posibles nuevos modelos a incorporar.



Modelo de Clases Factory

La clase “app.py” proporciona la instancia de “app” la cual va a ser implementada por cada una de las clases concretas las cuales van a agregar e implementar sus propias operaciones.

Dado a que estamos implementando la generación de un objeto a partir de una librería externa, se necesitan inyectar una serie de decoradores dependientes los cuales impiden la creación de una interfaz intermedia entre nuestras clases concretas y “app.py”, siendo “app.py” la propia interfaz la cual está siendo implementada por cada una de las clases concretas.

- ## Decorator

Los decoradores de Python son funciones que se utilizan para transformar otras funciones, con Flask el decorador “@app.route” se utiliza para hacer coincidir las URL con las funciones de las vistas en la aplicación instanciada.

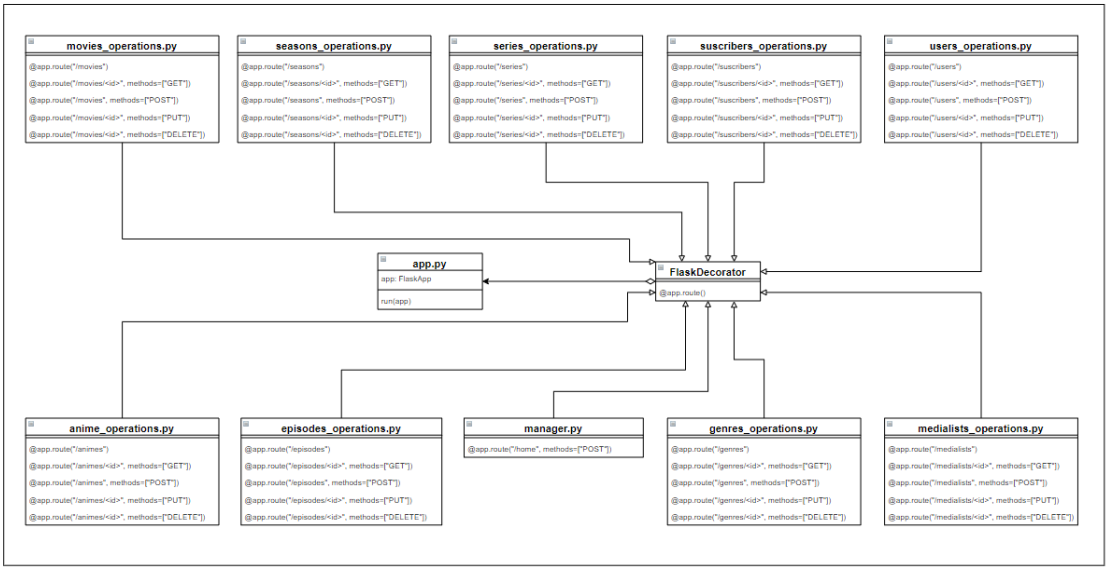
Las rutas definidas en la aplicación se han especificado utilizando variables de Python y pasadas a la función vista:

```
@app.route('/users/<id>', methods=['DELETE'])
@cross_origin(origin='localhost', headers=['Content- Type', 'Authorization'])
def delete_user(id):
```

Método “get_users” de “users_operations”

En “@app.route” se está pasando el parámetro ‘/users/<id>, methods=[‘DELETE’]’, donde se especifica una ruta estática (‘users’) y un método (‘DELETE’), pero el campo <id> es una variable la cual va a depender del valor que le pasemos, en este caso el id del usuario que queramos eliminar.

Por otra parte, se ha incluido un segundo decorador también proporcionado por la librería de Flask, “@cross_origin”, el cual nos permite añadir cabeceras en las respuestas entre las diferentes aplicaciones, verificando el intercambio de datos dado a los posibles problemas de seguridad de autenticación y verificación de contenido.



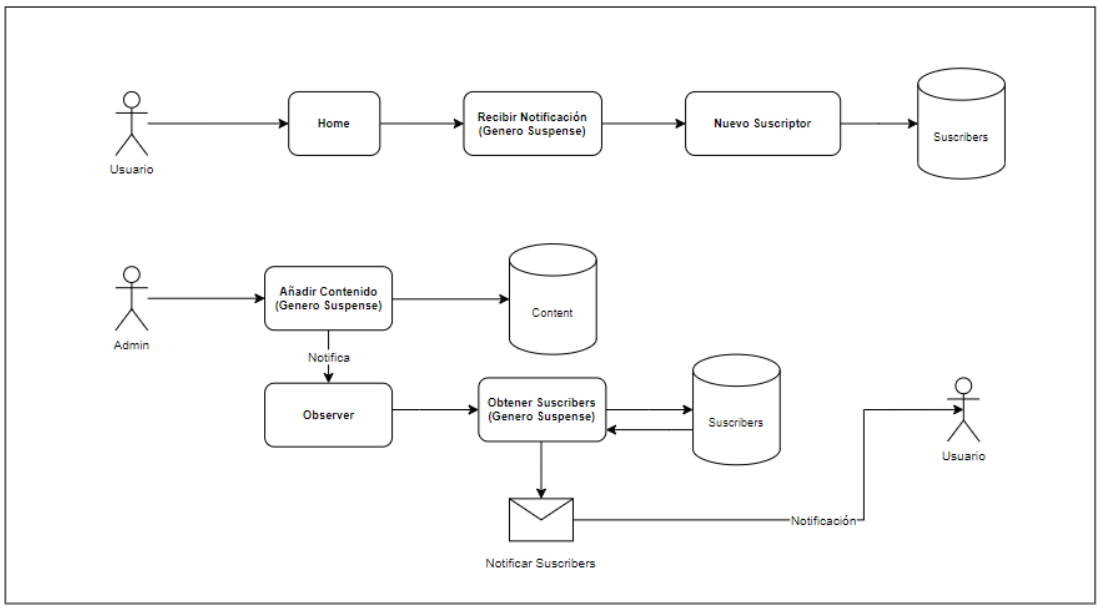
Modelo de Clases Decorator

- **Observer**

Para poder notificar a los usuarios que estén interesados en un tipo de genero independientemente del contenido, hemos implementado el patrón de diseño “Observer”, donde se van a crear suscriptores para cada uno de los géneros disponibles, el observador estará disponible para “Movies”, “Series” y “Animes”, en caso de que se incorpore un nuevo contenido y haya algún suscriptor con el mismo género que el contenido, este será notificado vía email.

En esta implementación se ha necesitado crear dos nuevos modelos de datos, “Genres” y “Suscribers”:

- “Genres”: Contiene todos los géneros de los contenidos de la aplicación.
- “Suscribers”: Usuarios que se han registrado para ser notificados cuando se incluya un nuevo contenido con un tipo de género concreto.

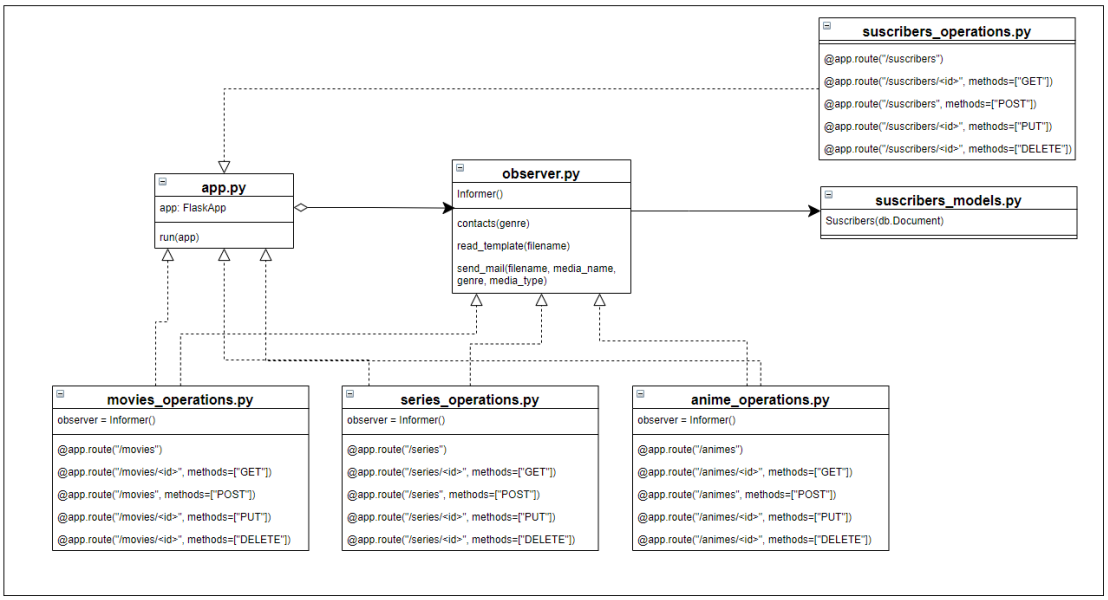


Ciclo de ejecución "Observer"

En el ciclo de ejecución del "Observer" podemos ver que sigue dos fases:

1- Un usuario accede al "Home" y registra su dirección de correo junto con un tipo de género determinado, este será suscrito y registrado dentro de la base de datos.

2- Un administrador añade un nuevo contenido a la aplicación con el mismo género que el usuario que se había registrado, una vez añadido el "Observer" coge todos los "Suscribers" que se han dado de alta y los filtra por el nuevo género del contenido incorporado, en caso de que haya alguno este será notificado vía email.



Modelo de Clases "Observer"

Las clases "movies_operations.py", "series_operations.py" y "anime_operations" instancian un objeto "Informer" de la clase "observer.py", de la cual van a implementar los métodos de la clase acorde al contenido y género insertado.

"Observer.py" por otra parte, va a utilizar la clase "suscribers_models.py" para obtener los modelos de base de datos de "Suscribers" con el fin de poder gestionar todos aquellos usuarios registrados.

```python
def add_movies_body():
    try:
        body = request.get_data()
        body = body.decode('utf8').replace("'", '"')
        body = json.loads(body)
        movie = Movies(**body).save().to_json()
        message = {"Success": 'Movie created successfully!'}
        movie = json.loads(movie)
        genres = movie['genres']
        observer.send_mail('templates/media_templates.txt', movie['name'], genres[0], 'movie')
```

Método "add_movies_body" de "movies_operations.py"

```python
def add_series_body():
    try:
        body = request.get_data()
        body = body.decode('utf8').replace("'", '"')
        body = json.loads(body)
        serie = Series(**body).save().to_json()
        message = {"Success": 'Serie created successfully!'}
        serie = json.loads(serie)
        genres = serie['genres']
        observer.send_mail('templates/media_templates.txt', serie['name'], genres[0], 'serie')
```

Método "add_series_body" de "series_operations.py"

```python
def add_animes_body():
    try:
        body = request.get_data()
        body = body.decode('utf8').replace("'", '"')
        body = json.loads(body)
        anime = Animes(**body).save().to_json()
        message = {"Success": 'Anime created successfully!'}
        anime = json.loads(anime)
        genres = anime['genres']
        observer.send_mail('templates/media_templates.txt', anime['name'], genres[0], 'anime')
```

Método "add_animes_body" de "animes_operations.py"

Como se puede observar en las tres capturas, cuando se ha conseguido realizar la inserción del nuevo contenido y se actualiza esto en base de datos, se implementa al "Observer" con el fin de notificar a los suscriptores correspondientes.