****

**Escuela de Ingenierías**

**Industrial, Informática y Aeroespacial**

Sistema híbrido de recomendación de películas empleando grafos de conocimiento

Sistemas de Información de Gestión y Business Intelligence

**GRADO EN INGENIERÍA INFORMÁTICA**

(Diciembre, 2020)

Autor: D. Álvaro González Jiménez

Tutor: D. Enrique López González

**ÍNDICE**

[1. Descripción del problema 2](#_Toc57559781)

[2. Herramientas utilizadas 5](#_Toc57559782)

[1. Base de datos: 5](#_Toc57559783)

[2. Interfaz de usuario (Frontend): 6](#_Toc57559784)

[3. Servidor (Backend): 6](#_Toc57559785)

[3. Base de datos utilizada 9](#_Toc57559786)

[1. Selección de los datos 9](#_Toc57559787)

[2. Conversión de los datos 10](#_Toc57559788)

[3. Importación de los datos 11](#_Toc57559789)

[4. Procesamiento de los datos 12](#_Toc57559790)

[5. Estructura de la base de datos 13](#_Toc57559791)

[4. Descripción de la aplicación 16](#_Toc57559792)

[5. Explicación del algoritmo 17](#_Toc57559793)

[6. Análisis de resultados 18](#_Toc57559794)

[7. Dafo 19](#_Toc57559795)

[8. Líneas de futuro 20](#_Toc57559796)

[9. Lecciones aprendidas 21](#_Toc57559797)

[10. Bibliografía 22](#_Toc57559798)

1. Descripción del problema

Tal vez si pensamos en un sistema de recomendación, el primero que se nos viene a la mente es un sistema de recomendación de películas. Todas y cada una de las plataformas de cine en streaming emplean un complejo sistema de recomendación encargado de analizar gigantescos volúmenes de datos y realizar sugerencias a sus usuarios en tiempo record. Aunque mucha gente lo desconozca, dichos sistemas no sólo tienen en cuenta los gustos del usuario para efectuar una recomendación, sino que analizan una gran cantidad de variables tales como el género, la edad, la raza, el dispositivo en el que sueles ver contenido, las horas del día en las que acostumbras a visualizar contenido o incluso el lugar de residencia con el fin de obtener recomendaciones lo más certeras posibles.

Estos sistemas, que se encuentran en constante desarrollo y evolución, son mucho más que simples recomendadores de películas. Si bien es posible que aunque utilices dichas plataformas con asiduidad no hayas reparado en ello, aunque estos sistemas recomienden la misma película o serie a dos personas diferentes, la portada mostrada no tiene por qué coincidir. Esto tiene una explicación muy sencilla, esos dos usuarios no son iguales, o al menos no lo son a ojos del sistema. Según la asunción que el sistema tenga del usuario, la portada que se le muestra difiere. Aunque desde luego este tipo de comportamientos no se pueden generalizar para toda la población, en la práctica ocurre que es infinitamente más probable que acabemos visualizando un contenido si la portada (que no deja de ser nuestra primera impresión sobre el contenido) es de nuestro agrado. Debido a esto, estas plataformas intentan que dicha primera impresión sea lo más fructífera posible, mostrando al usuario una portada que se asemeje al estilo de contenido que al usuario le guste visualizar, así como mostrando personajes que concuerden en raza y género con la asunción que el sistema tenga de esas dos características del usuario.

Tras esta breve descripción del estado del arte, supongo que lo más lógico es pensar ¿por qué iba a querer alguien desarrollar un sistema de recomendación de películas dado el extremo desarrollo y complejidad que caracteriza a los sistemas existentes en el mercado? Pues, aunque parezca increíble, tiene su respuesta. Vayamos por partes:

1. **Extensión del catálogo:** Como es evidente, los sistemas de recomendación de las grandes plataformas de streaming única y exclusivamente trabajan sobre el conjunto de películas que, en ese momento, están disponibles en la plataforma. Esto supone una gran diferencia con mi aplicación, que consta con una base de datos con información y valoraciones de algo más de 9.000 películas, mientras que por ejemplo Netflix apenas cuenta con unas 2.500, Prime Video con unas 4.400 y HBO con unas 900.
2. **Antigüedad de las películas:** En las plataformas de streaming, las películas que más abundan son aquellas más modernas, que por lo general tienden a gustarle mucho más al gran grueso de la población. Sin embargo, precisamente por la necesidad de tener en el catálogo películas que agraden a la mayoría, quedan en el olvido un sinfín de clásicos como *Con la muerte en los talones* de Alfred Hitchcock, auténticos clásicos del séptimo arte discriminados únicamente por su antigüedad. En mi sistema, esta discriminación no existe. Pueden encontrarse películas desde el año 1903 hasta la actualidad, recomendándose todas ellas por igual, todo en base a ciertos criterios y reglas matemáticas que serán perfectamente descritas en capítulos posteriores de este mismo documento.
3. **Ausencia de intereses externos:** Con el paso de los años, las plataformas de streaming han dejado de ser simples intermediarias entre productoras cinematográficas y usuarios, pasando a convertirse cada vez con más frecuencia en productoras del propio contenido que ofrecen. Aquí se plantea un dilema, ¿acaso alguien considera que los sistemas de recomendación no van a valorar especialmente bien las películas que sus propias compañías han producido? Esas películas no se pueden ver en una sala de cine, tampoco se pueden adquirir en un centro comercial. Que tengan éxito o no depende solamente de que una gran cantidad de usuarios las visualicen, y ahí, los sistemas de recomendación tienen la misma importancia que una buena campaña de marketing. Mi sistema, por el contrario, no se ve influenciado por ningún tipo de vicio ni interés externo. Es, si se quiere ver así, totalmente puro e imparcial.

Habiendo ya explicado en qué se diferencia de los sistemas de recomendación convencionales, queda por explicar exactamente cómo funciona el sistema, qué problemas resuelve y, en definitiva, cuál es su razón de ser.

En su función más elemental, es un sistema de recomendación híbrido capaz de recomendar películas en base a dos aspectos totalmente diferentes que son aunados finalmente para efectuar una recomendación final. Estos dos aspectos son los siguientes:

* **Algoritmo de recomendación basado en contenido:** En base a las características que definen a las películas valoradas por el usuario, se efectúan recomendaciones de películas similares a las valoradas.
* **Algoritmo de recomendación de filtrado colaborativo:** En base a las valoraciones asignadas a las películas, se recomiendan al usuario las películas mejor valoradas por otros usuarios, en este caso aquellos que más se parecen a él.

Además de como sistema de recomendación, la aplicación también puede funcionar como una agenda en la que apuntar aquellas películas que hemos visto y la puntuación que les hemos asignado, aunque claramente se trata de una función secundaria.

1. Herramientas utilizadas

En lo relativo a las tecnologías utilizadas, se pueden dividir en 3 grupos perfectamente diferenciados: base de datos, interfaz de usuario y servidor.

1. Base de datos: Entre las diversas tecnologías de bases de datos que pueden encontrarse en el mercado, hay dos que se llevan toda la atención: las bases de datos relacionales (MySQL, Oracle, MSSQL) y las bases de datos orientadas a documentos (MongoDB, CouchDB). Aunque sus usos son extremadamente variados (no en vano copan casi todo el mercado) para el problema que deseaba resolver no eran, ni en el mejor de los casos, lo más mínimamente viables.

El motivo es sencillo, no son capaces de computar cálculos sobre el más de millón de relaciones de las que dispone la base de datos del sistema en un periodo de tiempo aceptable. Por ello, me decidí a utilizar una base de datos orientada a grafos de conocimiento, Neo4j para ser más exactos. Su estructura y funcionamiento, en el que las relaciones son el aspecto más importante, las convierten en la base de datos más apropiada para resolver este tipo de problemas, así como otros problemas relacionados con detección de fraudes o incluso investigaciones genéticas.

Habiendo decidido el tipo de base de datos a utilizar, tan sólo quedaba una cosa, aprender a interactuar con ella. Neo4j implementa un lenguaje propio de consultas basado en SQL denominado Cypher, lo que hace que, si bien no sea necesario aprender la lógica que opera detrás de las consultas, sí que resulte extremadamente necesario aprender la sintaxis que emplea Cypher. Para ello, decidí completar el curso **Introduction to Neo4j 4.0** [1]**,** un curso de unas 16 horas de duración en el que pude aprender todas las funcionalidades de Cypher que posteriormente tuve que utilizar en el desarrollo del sistema, es decir, funcionamiento de una base de datos orientada a grafos de conocimiento, creación de nodos y relaciones, queries, indexación, importación de datos…

Sobre la base de datos hay muchos más aspectos que comentar, de modo que he decidido crear un capítulo que hable en exclusiva del origen del base de datos, procesamiento, importación, estructura, etc.

1. Interfaz de usuario (Frontend): A la hora de diseñar la interfaz de usuario, que generalmente se denomina frontend, decidí utilizar las tecnologías que ya conocía previamente. Así, al no tener que aprender desde 0 ninguna tecnología, puede realizar el desarrollo de una forma mucho más eficiente, obteniendo además un mejor resultado. Las tecnologías en cuestión son las siguientes:
2. **Javascript** [2]**:** Lenguaje de programación que aporta interactividad dinámica a la página web.
3. **Vue** [3]**:** Framework de Javascript que permite la construcción de interfaces de usuario y aplicaciones de una única página.
4. **Vuetify** [4]: Framework de Vue que permite la creación de interfaces de usuario basándose en componentes prediseñados.
5. **Vue router** [5]: Librería encargada de gestionar los movimientos entre vistas en una aplicación creada con Vue.
6. **Vuex** [6]: Librería encargada de gestionar los estados de una aplicación creada con Vue. Además, sirve como repositorio central de datos de la aplicación.
7. **Vuex persistedstate** [7]: Librería encargada de persistir la información de Vuex al refrescar una página.
8. **Axios** [8]: Cliente HTTP basado en promesas que permite efectuar peticiones API a un servidor.
9. Servidor (Backend): A la hora de diseñar el servidor de la aplicación, generalmente denominado Backend, se me presentaron dos alternativas entre las que tuve que decidir. Por un lado, tenía la opción de desarrollarlo en nodejs [9], un entorno en tiempo de ejecución basado en Javascript [2] que permite el desarrollo de servidores. Por otro lado, dado que es en el servidor donde se ejecutan los algoritmos de recomendación, estaba la opción de desarrollarlo en Flask [10], un framework de Python [11] que permite el desarrollo de servidores de forma sencilla.

Tras valorar pros y contras de ambas implementaciones me decidí por la segunda opción, especialmente porque Python iba a ser muchísimo más eficiente que Javascript a la hora de realizar operaciones de análisis y procesamiento de datos.

Como no había desarrollado jamás una aplicación con Flask, me vi en la necesidad de aprender desde 0 su funcionamiento, el cual aprendí mediante dos vías diferentes:

1. **Visualización de un vídeo en Youtube** [12]: En este vídeo se explica a la perfección el funcionamiento del framework. Peticiones, respuestas, gestión de rutas, argumentos, validación de entradas… Tan solo visualizando los 50 primeros minutos de este vídeo ya pude ponerme sin ningún problema a desarrollar el backend del proyecto.
2. **Observación del siguiente proyecto en Github** [13]: En este repositorio, dentro de la carpeta flask-api puede encontrarse un ejemplo de desarrollo de una aplicación Flask con acceso a base de datos neo4j que me sirvió de ejemplo para observar cómo se hacen las consultas a la base de datos desde Python.

Tras aprender el funcionamiento del framework, proceso que me llevó aproximadamente un día, pude empezar a desarrollar el backend del proyecto sin ningún tipo de problema. Las tecnologías que empleé, además de por supuesto Python y Flask, fueron las siguientes:

1. **Pandas** [14]**:** Módulo de Python que sirve para la manipulación y análisis de datos.
2. **Numpy** [15]**:** Módulo de Python que da soporte a la creación de vectores y matrices, así como a la realización de una gran cantidad de operaciones matemáticas complejas.
3. **Neo4j** [16]**:** Módulo de Python que permite la comunicación con bases de datos Neo4j.
4. **Sklearn** [17]**:** Módulo de Python que aporta una gran cantidad de funcionalidades relacionadas con Aprendizaje automático.
5. **Scipy** [18]**:** Módulo de Python que aporta una gran cantidad de herramientas y algoritmos matemáticos.

1. Base de datos utilizada

A la hora de desarrollar un buen sistema de recomendación, resulta estrictamente necesario que la base de datos sobre la que se trabaje sea lo más completa posible. Por muy robusto y efectivo que pueda ser el sistema, si la calidad de los datos no es la adecuada, las recomendaciones en la práctica no serán casi nunca certeras. Debido a esto, he considerado oportuno crear un capítulo dedicado única y exclusivamente a la propia base de datos en la que se traten diversos temas como la selección, procesamiento y carga de los datos, así como de la estructura de la propia base.

1. Selección de los datos

Para seleccionar un conjunto de datos sobre el que trabajar, decidí buscar en la conocida comunidad de científicos de datos Kaggle [19]. Allí, pueden encontrarse una gran cantidad de dataset, cursos y competiciones de temáticas más que variadas. Entre los dataset de películas que encontré, debo destacar los siguientes:

1. **TMDB 5000 Movie Dataset** [20]: Aunque se trata de un dataset bastante completo con numeras películas y valoraciones, faltaban algunos campos en la base de datos que, en mi opinión, eran muy importantes, como por ejemplo el reparto de la película.
2. **The movies dataset** [21]**:** Si bien se trata de un dataset extremadamente completo que contiene una gran cantidad de información sobre cada película, valoraciones, etc. A la hora de explorar el conjunto de datos en profundidad me di cuenta de que todo estaba almacenado en formato json dentro de un archivo csv. En base a esto, y aunque el dataset en principio tenía bastante buena pinta, decidí seguir buscando.
3. **Movielens** [22]**:** El dataset de películas por excelencia sobre el que desarrollar un sistema de recomendación, que incluye millones de valoraciones de usuarios a decenas de miles de películas. El único problema que encontré al dataset es que no almacena información sobre el casting de las películas, lo que hace que, a la hora de mostrar información en mi aplicación sobre las películas, no pueda ser esta muy completa.
4. **Extensión de Movielens** [23]**:** Cuando ya había decidido utilizar Movielens, encontré por casualidad este dataset, una ampliación de Movielens realizada por algunos investigadores de la Universidad Autónoma de Madrid. En este dataset, además de toda la información existente en el dataset de Movielens, podemos encontrar otra información adicional como el país de origen de la película, el reparto, director, etc. Por si esto fuera poco, todo el dataset ha pasado previamente un proceso de limpieza, de modo que no hay que realizar un proceso de limpieza de datos muy inferior a los demás.

Tras analizar las ventajas e inconvenientes de seleccionar cada uno de los diversos dataset, decidí utilizar el último de todos (la extensión de Movielens) dado que contiene una gran cantidad de películas y valoraciones y, además, no es necesario realizar un proceso de limpieza demasiado exhaustivo.

1. Conversión de los datos

Para importar datos a una base de datos Neo4j desde un archivo, es necesario que éste se encuentre en formato CSV. En mi caso, el conjunto de datos elegido se encontraba en varios archivos en formato DAT, en el que las diferentes columnas se encuentran separadas por tabulaciones en lugar de por comas. Debido a esto, me vi en la obligación de desarrollar un pequeño script en Python que se encargara de transformar los archivos al formato deseado.

Dentro de este pequeño script, además de cambiar las tabulaciones por comas, también se realizan otro tipo de transformaciones como, por ejemplo, reemplazar todos los caracteres especiales por sus equivalentes en el lenguaje inglés, eliminando por tanto las tildes a las vocales, sustituyendo la letra ñ por la n, etc.

Aunque no lo había comentado hasta el momento, todos los archivos que forman la base de datos, un total de 6, deben ser transformados mediante el script en Python procesar\_csv.py que puede encontrarse en el repositorio de Github en la carpeta Scripts.

1. Importación de los datos

Con el fin de importar todos los datos en la base de datos, y aunque es posible hacerlo en un único script, decidí crear un script por cada archivo, de modo que fuera más sencillo tanto depurar errores y, además, el tiempo de carga fuera menor.

Aunque la estructura de dichos scripts difiere en base a cuál sea el archivo que se quiere cargar, aquí puede encontrarse un ejemplo de la carga de todas las películas que contiene la base de datos:

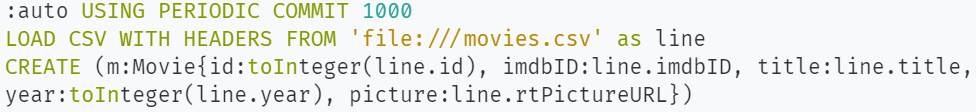


Figura 3.1.- Importación datos películas

Al efectuar la carga, decidí importar el contenido de los archivos en su totalidad, de modo que la carga de las valoraciones puede llegar a demorarse unas cuantas horas. En el caso de querer realizar esta carga en un periodo de tiempo mucho menor, bastaría con añadir una limitación en el script de carga, indicando que solo se quieren cargar las x primeras líneas.

Los scripts de carga pueden encontrarse dentro del repositorio de Github en la carpeta Scripts. Al ejecutarlos, es importante incluirlos dentro de la carpeta Import en Neo4j, de no ser así, el gestor no sería capaz de encontrarlos y, por tanto, no se podría efectuar la carga de manera satisfactoria.

Muy relacionados con la carga de los datos está la indexación. En vistas a que el sistema de búsqueda implementado en la interfaz de usuario resulte más efectivo, siempre es conveniente añadir una serie de índices que, bien implementados, son capaces de reducir los tiempos de carga de una manera increíble.

En mi caso, el índice principal y más importante es el siguiente, que permite indexar las películas en base a su título:



Figura 3.2.- Indexación películas en base a título

Al igual que todos los scripts, los archivos encargados de la creación de índices pueden encontrarse dentro del repositorio Github en la carpeta Scripts.

1. Procesamiento de los datos

Habiendo importado todos los datos, llega el momento de efectuar una serie de transformaciones en los datos que nos ayuden a eliminar duplicados y a facilitar el trabajo a los algoritmos de recomendación que se ejecutarán en el backend.

Dentro de este procesamiento, destacan 3 tipos totalmente diferenciados:

* 1. **Eliminación nodos repetidos**: En vistas a una mayor robustez del sistema de recomendación, resulta necesario eliminar aquellos nodos que se encuentran repetidos en la base de datos. Aunque el dataset se encontraba ya bastante limpio en un principio, sí que se podían encontrar algunas inconsistencias que debían ser solventadas.

Entre los directores, el id asignado para su identificación se encontraba en algunas ocasiones repetido, de modo que existían dos o más nodos para un mismo director. Esto también ocurría con algunos actores y películas, que también tenían el mismo identificador o la misma portada en el caso de las películas.

Para eliminar estos nodos repetidos, decidí implementar un sencillo Script en Cypher. El siguiente ejemplo muestra como eliminar todas las películas con la misma portada a excepción de 1. Sin embargo, el script es perfectamente extrapolable a cualquier otro tipo de nodo

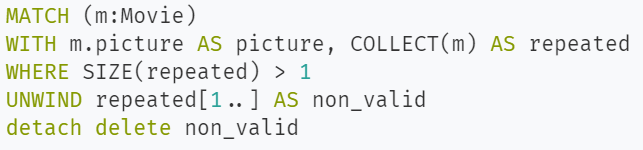


Figura 3.3.- Eliminar películas repetidas

* 1. **Eliminación nodos incompletos**: Al igual que con los nodos repetidos, los nodos incompletos también podían suponer un problema al sistema. Debido a esto, decidí eliminar todos aquellos nodos que carecieran de alguna propiedad, ya sirviera esta de identificación o fuera secundaria.
  2. **Procesamiento de texto**: Aunque en un principio no había reparado en ello, al ir desarrollando el algoritmo de recomendación basado en contenido reparé en que, por la forma de separar tokens en Python, no podían existir guiones ni espacios en los nombres de las características que fueran a ser utilizados como criterios de recomendación. Debido a esto, decidí desarrollar un script en Cypher que lo solucionara. El siguiente ejemplo muestra como sustituir los guiones y espacios contenidos en el texto por barras bajas:

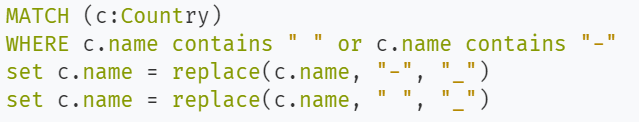


Figura 4.- Sustitución espacios y guiones por barra baja

1. Estructura de la base de datos

Para terminar el capítulo, considero oportuno explicar de forma detallada cuál es la estructura de la base de datos, es decir, por qué tipo de nodos y relaciones está formada.

Al haberse realizado una carga de la totalidad de los datos, el tamaño de la base es considerable, unos 110.000 nodos y 1.000.000 de relaciones. Dado que la gran mayoría de relaciones son valoraciones de los usuarios a las películas, y como ya comenté anteriormente, en el caso de querer que los tiempos de respuesta de la base de datos sean menores y el tiempo de importación de los datos sea también menor, tan sólo es necesario establecer una limitación a la hora de cargar las valoraciones en las que se indique el número de éstas que se quieran cargar.

En total, la base de datos está formada por 6 nodos diferentes y 5 relaciones entre ellos, siendo el nodo central de la base el que contiene el título de las películas. El esquema de visualización de la base de datos sería el siguiente:

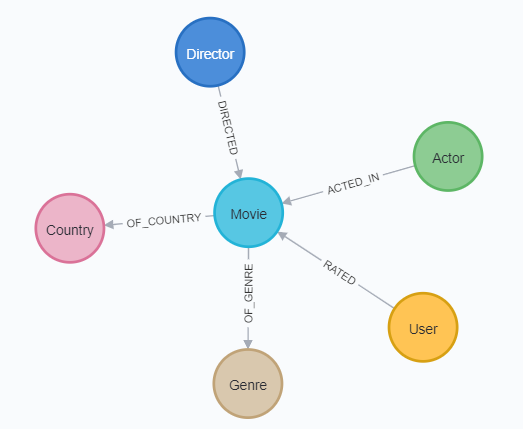


Figura 5.- Esquema visualización BBDD

A su vez, se podría definir cada uno de los nodos de la siguiente manera:

* **Movie**: Nodo que contiene la siguiente información sobre una película: identificador único, identificador de la película en imdb, título, año, link a la fotografía de la portada. En la base hay un total de 9.060 películas diferentes.
* **Genre**: Género al que puede pertenecer una película. En la base se pueden encontrar un total de 20 géneros diferentes.
* **Country**: País de origen de una película. En la base hay un total de 71 países diferentes.
* **Actor**: Nodo que contiene la información de cada actor en la base, identificador único y nombre del actor. Existen un total de 95320 actores diferentes.
* **Director:** Nodo que contiene la información de cada director en la base, identificador único y nombre del director. Existen un total de 4054.
* **User:** Nodo que contiene la información de cada usuario. Para los usuarios generados a partir del csv, la única información disponible será el identificador. Sin embargo, para los usuarios generados a partir de la aplicación web mediante el sistema de registro, también se encontrará almacenado su nombre, username y su contraseña debidamente cifrada.

Una vez descritos de forma correcta los distintos nodos existentes en la base de datos, creo necesario definir de igual modo las relaciones existentes entre ellos.

* **OF\_GENRE**: Relación entre Movie y Genre. Cada película puede pertenecer a un género o a varios. Existen un total de 18.366 relaciones de este tipo.
* **OF\_COUNTRY**: Relación entre Movie y Country. Cada película solo puede estar relacionada con un único país de origen. Existen un total de 9.060 relaciones de este tipo.
* **ACTED\_IN**: Relación entre Movie y Actor. Cada actor puede actuar en tantas películas como se desee. Existen un total de 207.985 relaciones de este tipo.
* **DIRECTED**: Relación entre Director y Movie. Cada director puede dirigir tantas películas como sea necesario y, además, cada película puede estar dirigida por más de un director. Existen un total de 9.060 relaciones de este tipo.
* **RATED:** Relación entre User y Movie. Cada User puede valorar tantas películas como quiera y cada película puede estar valorada por más de un User. La relación tiene un atributo que contiene la valoración que el usuario ha asignado a dicha película, un número entre 0 y 5. Existen un total de 758.836 relaciones de este tipo.

4. Descripción de la aplicación

1. Explicación del algoritmo
2. Análisis de resultados
3. Dafo
4. Líneas de futuro
5. Lecciones aprendidas
6. Bibliografía

|  |  |
| --- | --- |
| [1] | «Neo4j GraphAcademy,» [En línea]. Available: https://neo4j.com/graphacademy/online-training/v4/00-intro-neo4j-about/. |
| [2] | «API Javascript,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. |
| [3] | «API Vue,» [En línea]. Available: https://vuejs.org/. |
| [4] | «API Vuetify,» [En línea]. Available: https://vuetifyjs.com/en/. |
| [5] | «API Vue-router,» [En línea]. Available: https://router.vuejs.org/. |
| [6] | «API Vuex,» [En línea]. Available: https://vuex.vuejs.org/. |
| [7] | «API Vuex-persistedstate,» [En línea]. Available: https://www.npmjs.com/package/vuex-persistedstate. |
| [8] | «API axios,» [En línea]. Available: https://www.npmjs.com/package/axios. |
| [9] | «API nodejs,» [En línea]. Available: https://nodejs.org/es/. |
| [10] | «API Flask,» [En línea]. Available: https://palletsprojects.com/p/flask/. |
| [11] | «API Python,» [En línea]. Available: https://es.python.org/. |
| [12] | «Vídeo Flask,» [En línea]. Available: https://www.youtube.com/watch?v=GMppyAPbLYk. |
| [13] | «Código Flask,» [En línea]. Available: https://github.com/neo4j-examples/neo4j-movies-template/tree/master/flask-api. |
| [14] | «API Pandas,» [En línea]. Available: https://pandas.pydata.org/. |
| [15] | «API Numpy,» [En línea]. Available: https://numpy.org/. |
| [16] | «API Neo4j Python,» [En línea]. Available: https://neo4j.com/developer/python/. |
| [17] | «API Sklearn,» [En línea]. Available: https://scikit-learn.org/stable/. |
| [18] | «API Scipy,» [En línea]. Available: https://www.scipy.org/. |