Sergio Rivas Delgado

s.rivasd@alumnos.urjc.es

Descripción breve

Aplicación realizada utilizando la API de Spotify con el objetivo de mostrar canciones relacionadas a una dada según una serie de parámetros.

SISTEMA DE RECOMENDACIÓN DE CANCIONES DE SPOTIFY

TRABAJO DE FIN DE GRADO DE INGENIERÍA INFORMÁTICA

ÍNDICE

[1. INTRODUCCIÓN 2](#_Toc514503239)

[A) MOTIVACIÓN 2](#_Toc514503240)

[B) DESCRIPCIÓN DEL TRABAJO DE FIN DE GRADO 2](#_Toc514503241)

[C) ESTADO DEL ARTE (OTROS TFGS EN GOOGLE ACADEMICO) 2](#_Toc514503242)

[2. OBJETIVOS (listado de los pasos que he ido siguiendo) 2](#_Toc514503243)

[3. DESCRIPCIÓN ALGORÍTMICA (explicar la idea de la aplicación sin entrar en detalle de código) 2](#_Toc514503244)

[4. DESCRIPCIÓN INFORMÁTICA 2](#_Toc514503245)

[A) LENGUAJES 2](#_Toc514503246)

[B) BBDD 2](#_Toc514503247)

[C) SERVICIOS FLASK (CADA UNO DE LOS MÉTODOS DE LAS RUTAS) 2](#_Toc514503248)

[D) FUNCIONES JAVASCRIPT 2](#_Toc514503249)

[5. RESULTADOS 2](#_Toc514503250)

[A) CASOS DE EJEMPLO 2](#_Toc514503251)

[I. GRAFO PEQUEÑO 2](#_Toc514503252)

[II. GRAFO MEDIANO 3](#_Toc514503253)

[III. GRAFO GRANDE 3](#_Toc514503254)

[B) TABLA RESUMEN (CON TIEMPOS) 3](#_Toc514503255)

[6. CONCLUSIONES Y TRABAJO FUTURO 3](#_Toc514503256)

[7. BIBLIOGRAFÍA 3](#_Toc514503257)

# INTRODUCCIÓN

## MOTIVACIÓN

Hoy en día disponemos de una gran cantidad de datos que generan millones de dispositivos electrónicos que tenemos a nuestro alrededor, desde relojes inteligentes con los que podemos llevar el control de nuestra ruta al correr hasta poder mandar un correo electrónico a nuestros compañeros de trabajo. De la misma manera que es interesante toda esta recolección de datos, cobra una gran importancia el poder compartirlo con los demás para darle un mayor valor o importancia a los objetivos conseguidos.

El crecimiento de la cantidad de información de la que disponemos está creciendo en los últimos años de manera exponencial. De esta manera nuestro conocimiento tiene que avanzar para equipararse con la enorme cantidad de datos de los que disponemos.

Aunque disponemos de gran cantidad de dispositivos y redes sociales para poder interactuar con todos estos datos, nos faltan herramientas apropiadas para poder procesarlos correctamente. En la actualidad vivimos en la época del *“Big Data”* en la que se suben una enorme cantidad de vídeos a *Youtube* o se publican millones de *Tweets* acerca de un suceso de actualidad en sólo minutos.

*“Big Data”* es el término que usamos para referirnos a una gran cantidad de datos, de manera que estos no se pueden procesar con las herramientas tradicionales. Sin embargo, hay que tener precaución a la hora de procesar estos datos y para conseguir un buen resultado debemos tener en cuenta la regla de las 4 V’s. La primera de ellas y la más evidente es el **Volumen**, debemos tener en cuenta la capacidad de cómputo ya que se prevee que en el año 2020 los datos se multipliquen por 44 con respecto de los que disponíamos en 2009. La segunda de ellas es la **Variedad**, porque a mayor cantidad de datos, más tipos de los mismos puede haber, por lo que debemos de tenerlo en cuenta a la hora de procesarlos. La tercera V es la **Velocidad** ya que es importante ofrecer un buen servicio al usuario y proporcionarle la información en un tiempo aceptable. La última es la Veracidad, porque tenemos que saber discriminar entre los datos que nos van a aportar valor y los que no.

Ante esta falta de herramientas surgió mi motivación para llevar a cabo este trabajo de fin de grado, ya que me gusta que sea algo que aporte valor y que sea de utilidad para que en un futuro algún desarrollador más lo pueda usar o mejorar para proyectos muy interesantes.

A todo el mundo le gusta la música, ya sea *Pop*, *Rock*, *Electrónica*… En este sentido la aplicación más conocida mundialmente para poder escuchar, compartir, crear listas con tus canciones favoritas y muchas cosas más es Spotify.



Ilustración 1. Logo actual de la compañía Spotify.

Spotify se trata de una aplicación multiplataforma que es usada para la reproducción de música vía *streaming.* Cuenta con dos versiones, una gratuita y otra premium mediante la cual tenemos algunas ventajas como evitar publicidad y anuncios en la aplicación, mejorar la calidad del audio, poder escuchar tu música sin necesidad de una conexión a internet, poder tener más libertad a la hora de escuchar una canción, artista o lista de reproducción completa, o tener disponibles modos adicionales como el modo radio.

Se implantó el 7 de octubre de 2008 en el mercado europeo, pero no fue implantado en el resto del mundo hasta el año 2009. Podemos tener acceso a *Spotify* desde prácticamente todos los sistemas operativos del mercado (lo que es bastante poco común entre la mayoría de las aplicaciones en la actualidad) entre los que se encuentran *Microsoft Windows, Mac OS X, Linux, Windows Phone, Symbian, iOS, Android* y *BlackBerry*.

En cuanto a su origen cabe mencionar que es una empresa de origen sueco, concretamente de Estocolmo, aunque a lo largo de su desarrollo a firmado convenios con importantes discográficas como pueden ser *Universal Music, Sony BMG, EMI Music, Hollywood Records, Interscope Records* y *Warner Music.*

Se trata de una aplicación con una enorme cantidad de usuarios por lo que de esta manera cobra más sentido nuestro proyecto como ayuda al procesado de esta enorme cantidad de información. Según datos de junio de 2015, un total de 75 millones de usuarios están activos en la plataforma (de los cuales 20 millones son usuarios *premium*).

Por último, es interesante mencionar cómo funciona la transferencia de archivos de audio en *Spotify*. Se trata de una combinación de servidores dedicados al *streaming* y en la transferencia de red de pares (P2P) que serían los propios usuarios entre sí. No se necesita nada más que una conexión de 256 kbit/s y utiliza el códec de audio Vorbis (Ogg). Los usuarios premium, como ventaja tienen una calidad de audio superior a los usuarios básicos (q9).

## DESCRIPCIÓN DEL TRABAJO DE FIN DE GRADO

El proyecto consiste en la implementación de una aplicación que actúe como recomendador de canciones de Spotify de una manera visual e intuitiva. Basta con que el usuario, una vez arrancada la aplicación, introduzca la canción con la que quiere comenzar su búsqueda (también se le da la posibilidad de introducir el nombre del artista o grupo para hacer la búsqueda más rápida y concisa). Esta zona está situada en la parte superior de la aplicación de manera que es fácilmente identificable en cuanto el usuario entra en la aplicación, aunque sea la primera vez que la está usando.

Una vez el usuario ha realizado la primera búsqueda, la aplicación se pone a trabajar con el API de Spotify para proporcionar un grafo que visualmente le proporciona una gran experiencia al usuario. Siguiendo los colores de la aplicación original de Spotify, los artistas se representarán mediante nodos grises, y las canciones del color verde característico de Spotify. En cuanto a la identificación, simplemente con pasar el cursor por encima de cada nodo podemos saber fácilmente de que artista o canción se trata. Mientras que la aplicación está trabajando con el API de Spotify nos aparecerá un icono de carga en la parte superior del grafo. Una de las dudas que surgió en esta parte fue cómo diferenciar canciones o artistas que habían sido incluidas en el grafo en una llamada anterior o posterior a la API. Esto fue resuelto de una manera muy fácil e intuitiva para el usuario, a medida que aumentamos las llamadas a la API, las canciones y artistas más recientes tendrán un mayor tamaño en la aplicación, quedando las canciones y artistas más antiguos en la aplicación de un tamaño inferior.

En relación con el grafo, se ha implementado una funcionalidad de gran utilidad para que el usuario se sienta cerca de su aplicación original de Spotify y nuestra herramienta le resulte realmente útil. Si realizamos *click* en un artista del grafo una vez se ha cargado totalmente en nuestra página, se nos abre una nueva pestaña en el navegador, que nos indica en la versión web de Spotify la página del grupo que hemos seleccionado (y por lo tanto el resto de sus discos, canciones o incluso próximas fechas y lugares de giras). Sin embargo, si hacemos *click* en una canción del grafo, además de continuar con la expansión del mismo buscando nuevas canciones relacionadas, se nos abre otra pestaña en el navegador con la reproducción de la canción en la versión web de Spotify desde donde podemos acceder a las demás canciones del disco. Hay que indicar que, para acceder a esta funcionalidad, el usuario tiene que iniciar sesión en Spotify para que el comportamiento y la experiencia sea la más adecuada.

Además de la zona de representación del grafo, contamos con una zona que nos muestra los detalles de la última canción seleccionada. Estos datos son el título de la canción, el artista de la misma, la duración de la canción, la popularidad dentro de la aplicación de Spotify y la fecha de lanzamiento del álbum, así como la foto del mismo de manera que sea una experiencia enriquecedora por parte del usuario a la hora de utilizar la aplicación.

Como mencionaré más adelante, durante el desarrollo del trabajo, he implementado varias mejoras en la experiencia de interacción del usuario con la aplicación. Con estas mejoras, tanto la zona de representación del grafo como la de los detalles de la canción seleccionada se actualizan en un corto periodo de tiempo para evitar esperas innecesarias.

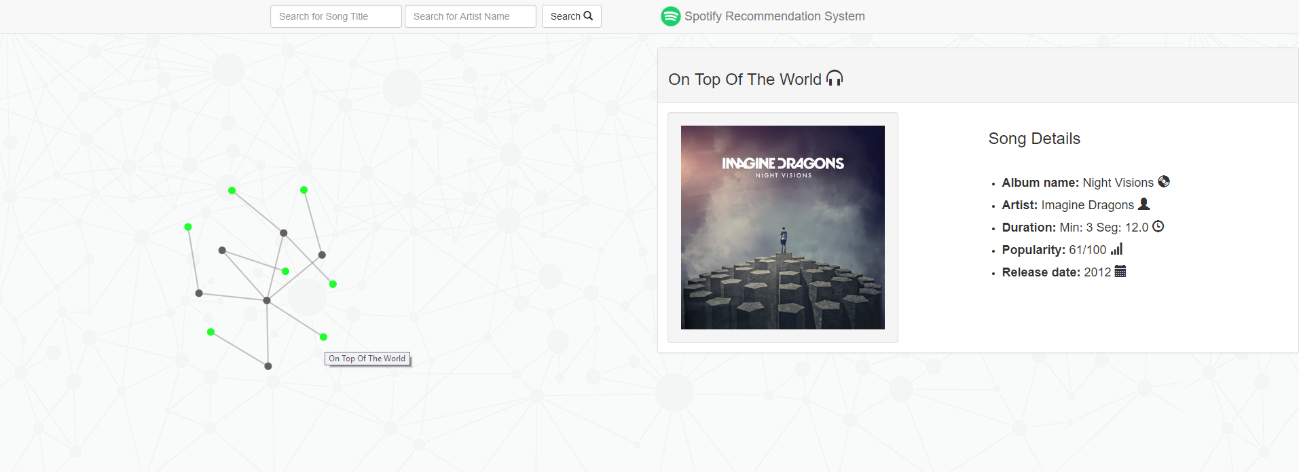


Ilustración 2. Captura de pantalla de un ejemplo de ejecución de la aplicación.

## ESTADO DEL ARTE

Una vez te dispones a comenzar un proyecto que pueda ser útil para que otros usuarios puedan aprovechar tu trabajo para tener una buena experiencia, una parte importante del tiempo de preparación debe ir a visualizar otros trabajos anteriores que puedan estar relacionados con el tuyo.



Ilustración 3. Logo de la herramienta Google Académico.

Una herramienta muy útil para poder investigar de un solo vistazo información relevante para tu propósito es Google Académico. Esta versión del famoso buscador Google nos permite extraer información relacionada con el ámbito de estudio e investigación proporcionándonos PDFs, libros, *e-books* y todo tipo de archivos de interés.

En cuanto comencé a buscar me di cuenta de la falta que hacía una aplicación como la de mi proyecto ya que no abundan este tipo de trabajos de investigación acerca de cómo poder ofrecer al usuario una buena recomendación de canciones siguiendo sus gustos. No obstante, mostraré tres trabajos bastante relacionados y que me han parecido de gran interés:

***Combining Spotify and Twitter Data for Generating a Recent and Public Dataset for Music Recommendation*** [(PDF)](https://dbis.uibk.ac.at/sites/default/files/2017-03/gvdb14.pdf)

En primer lugar, tenemos esta aplicación realizado por *Martin Pichl, Eva Zangerle* y *Günther Specht.* Básicamente, se basa en sacar datos de la red social *Twitter* para interpretarlos y poder utilizarlos como huella digital para saber qué tipo de música les puede resultar afín a determinados usuarios con el fin de tener un recomendador de *Spotify*. Como comentan los autores en sus conclusiones se trata de un sistema limitado ante un gran número de recomendaciones además de tener que usar la relación entre dos redes sociales (lo que obliga a que el usuario de la aplicación esté dado de alta en ambas).

***Deep content-based music recommendation*** [(PDF)](http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf)

Por otro lado, tenemos el proyecto de *Aaron Vandenoord, Sander Dieleman* y *Benjamin Schrauwen.* En este caso, nos hablan de la dificultad concreta de recomendar una canción y no tanto en la forma. Tienen en cuenta la gran cantidad de plataformas disponibles en la actualidad y la disparidad de criterios en cuanto a recomendar una u otra canción al usuario final. En su memoria nos intentan aportar soluciones acerca de la base de datos a utilizar o como predecir los factores de latencia a través del audio de la canción determinada. También es de gran importancia el aporte sobre la factorización de matriz ponderada, ya que como nos comentan los autores, muchas veces los usuarios no votan o valoran expresamente todas las canciones que escuchan. Todos estos aspectos a tener en cuenta nos ayudan bastante a entender el problema de la complejidad de recomendar canciones y como poder afrontar las posibles soluciones.

***A Survey of Music Recommendation Aids*** [(PDF)](https://pdfs.semanticscholar.org/d904/0ff891c0afe9cb275419bcf4a6d6f12c5fc0.pdf)

Por último, me gustaría destacar la entrevista realizada por *Pirkka Åman* y *Lassi A. Liikkanen.* En esta encuesta nos informan acerca de la importancia que tienen los anuncios que nos recomiendan los sistemas de *streaming* online como pueden ser *Spotify* o *Last.fm.* Centrándonos en el caso de *Spotify*, destacan que se trata de una aplicación con interfaz muy intuitiva, facilita el acceso a nuestra música favorita al ser multiplataforma y tiene una gran calidad de audio. Pero por otro lado comentan que posee un sistema de recomendación bastante limitado. Lo que nos muestra una vez más la gran importancia y utilidad de nuestro proyecto.

# OBJETIVOS

Una vez asignado el trabajo lo primero fue aprender las tecnologías que iban a ser empleadas para interactuar con el API de Spotify.

Por un lado, consistiría en aprender el lenguaje de programación *Python* que es el utilizado para hacer las llamadas a la API. Al haber ya desarrollado ya con lenguajes de todo tipo en la carrera se me hizo bastante fácil aprender la sintaxis de *Python*.

Por otro lado, debía aprender cómo funcionaba la base de datos *Neo4j*. La base de datos *Neo4j*, se trata básicamente de un grafo formado por nodos y aristas que representan la información y relaciones guardadas en la misma.

Comencé por aprender *Python* poco a poco mediante problemas y ejercicios simples como realizar funciones, bucles y algunas estructuras de control de flujo para irme familiarizándome con el lenguaje. En esta parte me resultó un poco complejo la manera de llamar a mis programas con los argumentos por consola, pero una vez solventados estos problemas se volvió mucho más rápido y sencillo el desarrollo.

Una vez ya con el lenguaje interiorizado me dispuse a realizar mis primeras pruebas con el API de *Spotify*. Me resultó fácil el registro para sacar los identificadores propios de mi cuenta de desarrollador, pero un poco más complejo el tema de realizar peticiones al API una vez que me autenticase ya que no estaba muy bien indicado el tema de las variables de entorno en *Windows*. El poder realizar muchos de estos ajustes en *PyCharm* ha hecho que sea un IDE con el que me he sentido muy cómodo a lo largo del desarrollo.

Posteriormente comencé a realizar diferentes peticiones basándome en ejemplos de la API y propios míos para entender y comprender de primera mano cómo funciona dicha API. Estos ejemplos fueron tales como los álbumes publicados por un artista, la búsqueda de canciones dentro de un álbum, la búsqueda de los propios artistas mediante la URI o simplemente su nombre, la búsqueda de artistas relacionados con uno dado y la búsqueda de las propiedades de una canción. Éste último ejemplo nos acerca bastante a lo que buscamos para poder implementar nuestra aplicación sobre Spotify.

Ya entrando un poco más en materia, implementé un par de ejercicios más complejos. El primero consistía en buscar con una profundidad 3 los artistas relacionados con uno dado, para saber de esta manera un conjunto de artistas relacionados entre sí. El segundo consistía en sacar todas las canciones de todos los álbumes de un artista determinado y de cada una de estas canciones sus principales propiedades como la bailabilidad, ritmo, positivismo...

Una vez llegados a este punto, comencé a familiarizarme con la base de datos *Neo4j* y su lenguaje para realizar consultas y demás operaciones con la base de datos, *Cypher*. El aprendizaje me resultó muy sencillo ya que es similar a *SQL* y ya estaba acostumbrado a trabajar con bases de datos relacionales. Realicé algunos ejemplos de grafos sencillos con libros y películas relacionándolos entre sí.

Una vez aprendida las tecnologías a emplear, me pareció buena idea conectarlas en forma de ejercicio realizado anteriormente sobre las propiedades de las canciones de un artista dado. De forma que toda esta información se muestre en forma de grafo en la base de datos *Neo4j*.

En este paso del aprendizaje resultó problemático el tema de las relaciones entre nodos para ciertos ejemplos ya que estas relaciones son más apropiadas para asociar distintos tipos de entidades como pueden ser álbumes y canciones o canciones y artistas.

Una vez que hemos conseguido almacenar la información en la base de datos *Neo4j*, el próximo paso es representarla correctamente. Debido a la poca información y documentación en internet con respecto al grafo de *Neo4j*, me he visto obligado a utilizar la herramienta *Bokeh* para representar dicha información almacenada.

Debido al desconocimiento de dicha librería de Python he procedido a instalarla, que gracias al IDE *Pycharm* ha sido muy sencillo de realizar. Antes de ponernos con el problema real y cohesionarlo todo, he ido poco a poco implementando aplicaciones más sencillas para familiarizarme con la librería.

Teniendo en cuenta las exigencias del problema, decidí utilizar el *framework* de Python llamado *Flask* mediante el cual se pueden crear fácilmente interfaces gráficas utilizando la herramienta de D3JS en lugar de *Bokeh* como había planteado en un principio. Una vez encontrada una aplicación de ejemplo el siguiente paso es adaptarla a nuestro problema concreto. Cuando haya completado esta tarea, el siguiente paso será documentarse acerca de la maquetación con D3JS.

Tras unas semanas familiarizándome con *Flask*, conseguí hacer una aplicación de prueba que guardaba la información de la API en *Neo4j* y posteriormente mostraba el grafo en la pestaña del navegador. Esta aplicación relacionaba un álbum dado con las canciones de este. El siguiente paso es profundizar más con la API y con *Flask* para representar relaciones más complejas.

*Flask* me planteó varios problemas al principio ya que los métodos tenían que devolver un objeto *JSON* para que *D3JS* pudiese interactuar con los datos de manera correcta. Una vez solventados estos problemas solo planteó ciertas dudas el tema del algoritmo de la aplicación en sí. Una vez en este punto, la primera aproximación de la aplicación sólo permitía mostrar las canciones y artistas relacionados a uno dado sin poder profundizar más. Otro de los problemas que me dio *Flask* fue a la hora de transmitir información desde la interfaz y cuantas URLs iba a necesitar de cara a realizar búsquedas de canciones o artistas. Sin embargo, se solucionó de una manera muy simple, a través de parámetros. De esta manera, si el usuario sólo introduce el nombre de la canción y no el del artista, también se realiza la búsqueda. La cuestión determinante de esta aplicación es que también se puede utilizar ese método al realizar las siguientes búsquedas haciendo *click* en una canción del grafo, ya que se obtiene el nombre de la misma y se llama a esa URL.

El próximo paso fue poder mostrar en la herramienta web de Neo4j todas las relaciones correctamente antes de pasar a *D3JS* en el navegador web. Lo más difícil en este apartado fue distinguir entre todos los niveles de llamada a la API para poder formar correctamente la estructura. Además, como suele pasar con el tema de las APIs siempre suele haber ciertas excepciones con algunos datos como puede ser que una canción esté compuesta o representada por varios artistas, y eso nos puede llevar a error a la hora de poder guardar la información en la base de datos. Esto fue solucionado escogiendo al artista con más relevancia o importancia en el álbum correspondiente a la hora de conformar las relaciones.

Una vez finalizada esta aproximación había que transmitir la información de la base de datos *Neo4j* a la representación en el navegador web mediante *D3JS.* La información en internet acerca de esta librería es bastante extensa y diversa, ya que se usa para multitud de diferentes proyectos de diferente índole. Sin embargo, gracias a un aporte de mi tutor a modo de diapositivas de ciertos ejercicios me pude familiarizar más rápidamente con la librería.

Como ya comentaré más adelante en detalle, en este punto del trabajo he implementado dos aproximaciones diferentes. Una mediante *Flask* devolviendo información a través de *JSON* y otra accediendo a la base de datos desde código *JavaScript*.

La primera aproximación podemos decir que es la más modularizada y estructuralmente correcta. Aunque el principal problema venía en la pobre experiencia del usuario ante grandes volúmenes de datos. La segunda, solventa estos problemas de esperas no deseadas, pero de cara a la estructura sea la menos habitual.

Con la versión de *Flask* use *jQuery* para traer la información de la API para el módulo de detalles de la canción en la interfaz web. Me resultó bastante fácil porque ya había tratado con ello en el grado, más concretamente en la asignatura de desarrollo de aplicaciones web.

Una vez implementada la funcionalidad básica de la aplicación, pensé que sería de gran ayuda para el usuario mejorar la relación de nuestra herramienta con *Spotify* en sí (más allá de su obvia conexión con la API). En este aspecto se me ocurrió el tema de abrir una pestaña nueva en el navegador cada vez que el usuario quisiera interactuar con un artista o una canción del grafo. Tenía dos formas de hacerlo, mediante el nombre de uno u otro, o a través del código URI de Spotify. Comencé a implementar la primera alternativa, pero la descarté por lo siguiente. La pestaña que se habría nos llevaba a una página que realizaba automáticamente la búsqueda en la base de datos de *Spotify* por nosotros. El problema viene cuando es una canción con un título bastante común como puede ser *Tatoo.* Si nos encontramos en este caso puede ser que nuestra canción se encuentre de las últimas en el listado y se nos haga bastante tedioso encontrarla.

Por las anteriores razones opté por la segunda opción. La única dificultad añadida era traer la información del URI en cuestión desde la base de datos. Una vez hecho esto sólo teníamos que asignar la URL en función de si se trataba de un artista o de una canción teniendo en cuenta el tipo de nodo seleccionado. De esta forma la pestaña que se nos abre es bastante más directa y relacionada con lo que podemos esperar desde un principio. En el caso de que hagamos *click* en un artista, se nos abre directamente la página del artista en Spotify con toda su información de relevancia. En el caso de una canción (además de realizar la búsqueda de las canciones relacionadas a través de la API) se nos abre una pestaña con la canción implicada reproduciéndose y el disco al que pertenece con las demás canciones del mismo.

Para finalizar y darle un toque para que el usuario tenga más control de la aplicación, he añadido un par de barras de carga para que el usuario sepa en todo momento cuando la aplicación está trabajando para traerle toda la información. Esto es debido a que la velocidad de la API es limitada y para un proyecto que busca tanta información es bastante trabajoso.

A continuación, muestro un esquema para indicar más visualmente los pasos u objetivos que he ido cumpliendo en el desarrollo del proyecto.

# DESCRIPCIÓN ALGORÍTMICA

La estructura que sigue el grafo es la siguiente. En primer lugar, busca la canción indicada por el usuario y la introduce en el grafo. Posteriormente encuentra el artista relacionado con dicha canción buscada. Este artista puede ser introducido por el usuario a mano, aunque si no lo hace, la herramienta lo busca en la API, aunque dificulta el rendimiento.

Una vez tenemos en nuestra base de datos la canción y el artista iniciales, el siguiente paso es buscar todos los artistas relacionados con el que poseemos actualmente según el criterio de *Spotify*. A priori no se introducen todos ya que puede ser que no sea necesario y esto empeoraría el funcionamiento de nuestra aplicación. Según vamos recorriendo los artistas relacionados, vamos mirando en los discos que tienen publicados y en sus correspondientes canciones para ver si estas canciones están relacionadas con la que hemos comenzado nuestra búsqueda.

El criterio que hemos seguido para indicar que una canción es parecida a otra es el siguiente. Si observamos detenidamente la API de *Spotify* tenemos que cada canción dispone de una serie de atributos que la caracteriza y que podemos utilizar a nuestro propio criterio para diseñar nuestro algoritmo. Los atributos en los que nos hemos basado son los siguientes:

* *Danceability:* Esta propiedad nos indica como de apropiada es una canción para ser bailada de acorde a elementos como el tempo, la estabilidad del ritmo, la fuerza o la regularidad total. El valor de 0 indica la canción con menos capacidad para ser bailada y un valor de 1 la que más capacidad tendría.
* *Energy:* Nos representa una medida de la intensidad y la actividad. Por lo general, las canciones con más energía se sienten más rápidas, altas y ruidosas.
* *Liveness:* Detecta la presencia de público en la grabación. Un alto nivel aumenta la probabilidad de que la canción haya sido grabada en vivo (sobre todo si supera el valor de 0.8).
* *Mode:* Indica la modalidad (menor o mayor) de una canción. El tipo de escala a partir del cual se deriva su contenido melódico. El modo mayor se representa con un 1 y el menor con un 0.
* *Speechiness:* Detecta la presencia de palabras habladas en una canción. Cuanto más discurso tiene el audio (como puede ser el caso de un audiolibro) más cercano a 1 es el atributo.
* *Acousticness:* Es una medida acerca de cómo de acústica es la melodía. El valor de 1 representa una gran firmeza de que la canción es acústica.
* *Instrumentalness:* Predice si la canción no tiene contenido vocal. Cuanto más cercano a 1 es el valor, más probabilidad tiene esa ausencia de contenido vocal.
* *Valence:* Por último, este atributo mide cuánta positividad conlleva una determinada canción. Las canciones con un nivel elevado son más felices o eufóricas mientras que las que tienen un valor bajo son más tristes o depresivas.

Una vez que almacenamos todos estos valores de nuestra canción original, procedemos a compararlos con los que vamos obteniendo del resto de canciones que vamos recorriendo. Si la diferencia de valores entre todos los atributos de ambas canciones comparadas es inferior a 0.4 deducimos que se trata de una canción parecida. Este valor ha sido extraído a partir de un gran número de experimentos que han definido este valor cómo el más apropiado ya que no muestra demasiadas canciones, pero sí las necesarias.

Procedemos a recorrer las canciones del artista y una vez que hemos llegado al final es cuando tomamos la decisión acerca de incluirle o no en nuestro grafo. En el caso de que se haya añadido al menos una canción sí que debemos de incluir a dicho artista en nuestra representación. Por el contrario, si no se han encontrado canciones relacionadas a la dada en dicho artista tenemos que prescindir de él por el momento.

De esta forma vamos iterando por todos los artistas relacionados y su correspondiente discografía. Cuando hemos finalizado dicho proceso decimos que hemos llegado al final de la iteración y el grado tiene otro nivel más añadido. Para mostrarlo más gráficamente, una iteración o ciclo se representaría de la siguiente manera:

Esta estructura cambia de cierta manera con la segunda aproximación que realicé en el algoritmo. Como ya comenté anteriormente, el tema de devolver *JSON* con *Flask* y representarlos mediante *D3JS* producía un cuello de botella que era conveniente mejorar. De esta manera, el acceso desde *JavaScript* a *Neo4j* para ir actualizando periódicamente el grafo mejora notablemente la experiencia de usuario. La manera de actualizar el ciclo de trabajo de la aplicación con este cambio sería de la siguiente manera.

Paralelamente a la carga y actualización del grafo, también se realiza la descarga de los detalles de la canción seleccionada. Se reciben los datos de la API para mostrarlos paralelamente a la carga del grafo para mejorar notablemente la experiencia de usuario.

# DESCRIPCIÓN INFORMÁTICA

En este apartado vamos a entrar más en detalle acerca de por qué he escogido estos lenguajes y herramientas para sacar adelante este proyecto y los detalles y curiosidades de todos y cada uno de ellos. También volveré a hacer hincapié en las dos implementaciones realizadas (tanto la que incluye código *JavaScript* y *Flask* como la que sólo contiene *Flask* en relación con la base de datos *Neo4j*).

## LENGUAJES, IDE, FRAMEWORKS Y LIBRERÍAS

* ***PYTHON***



Ilustración 4. Logo del lenguaje de programación Python.

Se trata de un lenguaje de programación interpretado cuya filosofía es la realización de un código legible para cualquier programador que se enfrente a él. Cabe destacar la gran virtud de que se trata de un lenguaje multiparadigma, lo que significa que soporta orientación a objetos, programación imperativa y programación funcional. Además de todo esto es un lenguaje interpretado de tipado dinámico y multiplataforma.

Fue diseñado en 1991 por parte del holandés *Guido van Rossum,* es administrado por la *Python Software Foundation* y posee una licencia de código abierto denominada *Python Software Foundation License* que es compatible con la licencia pública general de GNU (a partir de la versión 2.1.1 del lenguaje).

Una de las características que podemos reseñar es la resolución dinámica de nombres. Esto quiere decir que relaciona un determinado método y el nombre de una variable durante la propia ejecución del programa. Otra característica bastante reseñable es la facilidad que nos proporciona este lenguaje para su extensión o modificación. Incluso es bastante sencillo de transcribir a otros lenguajes de programación en caso de necesitarlo como puede ser C o C++.

Una buena manera de medir la repercusión de un lenguaje de programación es a través del índice *TIOBE*. Este índice mide la popularidad e importancia de todos los lenguajes de programación utilizados. En mayo de 2018, se encuentra cuarto, solamente por detrás de lenguajes tan importantes como *Java, C* o *C++*. Sin embargo, es un lenguaje que se encuentra al alza como podemos observar en la siguiente gráfica que nos proporciona el índice *TIOBE*:

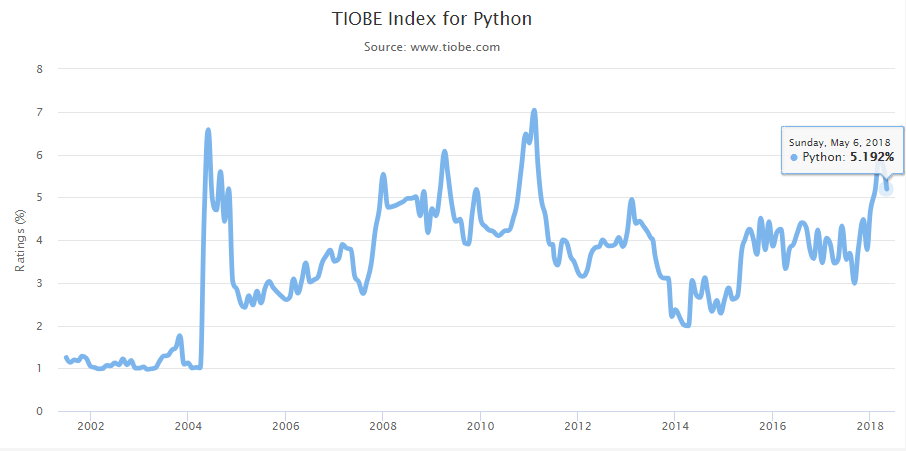


Ilustración 5. Gráfica de la importancia del lenguaje Python en el índice TIOBE.

Un aspecto más que relevante que hay que mencionar sobre Python es que muchos expertos en la materia lo han bautizado como el lenguaje del futuro. Esto es debido a su gran capacidad de utilización en ramas de la tecnología tan punteras como pueden ser el *Big Data*, la *Inteligencia Artificial*, el *Data Science* o la *Ciberseguridad* (como lenguaje de Scripting). Todo esto unido a que la comunidad del lenguaje es cada vez más grande, hay más consultas en *Stack Overflow* y hay más documentación disponible hace que los desarrolladores tengamos que seguir muy de cerca a este lenguaje en los próximos años.

En el aspecto personal, era un lenguaje de programación con el que no había trabajado y, una vez desarrollado el proyecto, sólo tengo buenas palabras para él. Sinceramente, una vez que has aprendido los primeros lenguajes de programación, el resto ya resultan más sencillos debido a la experiencia adquirida. No obstante, *Python* proporciona tal cantidad de funcionalidades intuitivas que hacen de la implementación a través de él una buenísima experiencia que repetiré en un futuro cercano sin duda.

* ***JAVASCRIPT***



Ilustración 6. Logo del lenguaje de programación JavaScript.

## BBDD

## SERVICIOS FLASK (CADA UNO DE LOS MÉTODOS DE LAS RUTAS)

## FUNCIONES JAVASCRIPT

# RESULTADOS

## CASOS DE EJEMPLO

### GRAFO PEQUEÑO

### GRAFO MEDIANO

### GRAFO GRANDE

## TABLA RESUMEN (CON TIEMPOS)

# CONCLUSIONES Y TRABAJO FUTURO

# BIBLIOGRAFÍA