




9 DE NOVIEMBRE DE 2022

# SPOTIFY TO SQLITE

DESARROLLO DE APLICACIONES PARA CIENCIA DE DATOS

GRADO EN CIENCIA E INGENIERÍA DE DATOS, 2º

Escuela de Ingeniería Informática, Universidad de Las Palmas de Gran Canaria





## Resumen

Este trabajo consiste en usar los conocimientos adquiridos en clase relacionados con las API REST y con el manejo de bases de datos SQLite. Debíamos obtener los datos, álbumes y canciones de al menos 5 artistas solistas y almacenarlos en una base de datos que contuviese las siguientes tablas: Artists, Albums y Tracks. Desde este proyecto se ha intentado que el código sea lo más legible y entendible posible, sin recaer en ningún momento en el uso de comentarios para la explicación de este. Ahondando en la metodología, el programa toma un array que contiene id's de artistas y en primera instancia los convierte a POJO para luego introducir los datos más relevantes en las tablas correspondientes (mas adelante se especificará de manera más detallada el proceso, así como las clases e interfaces utilizadas). La aplicación no se puede decir que sea rápida, ni eficiente, ni que esté al 100% optimizada, aunque he intentado que, con mis conocimientos actuales, ésta no incurra en acciones innecesarias o tenga código zombie; por ende, se puede decir que, aunque no será la más rápida, si se puede decir que realiza bien su función. En el caso de que no se lean la memoria al completo (que lo puedo entender), hay dos apartados MUY IMPORTANTES que deben tener en cuenta:

- Para probar el programa, deberán introducir sus propios Tokens de Spotify
- Deben cambiar el path donde se va a crear dicha base de datos

## Índice

Resumen.....	2
Recursos Utilizados .....	2
Diseño.....	3
Conclusiones .....	5
Líneas Futuras .....	5
Bibliografía .....	6

## Recursos Utilizados

Para la realización de este proyecto se ha utilizado el entorno de programación “IntelliJ Idea Community Edition 2022.2.3”, así como las dependencias de gson (para convertir JSON en clases POJO) y sqlite-jdbc, que permite a Java ejecutar código SQL. Para el controlador de versiones se ha utilizado Git (aunque he de admitir que algunos commits tienen unos nombres no adecuados, ya que no ayudan a identificar el contenido actualizado; y algunos al principio eran de prueba, ya que, como le comenté también a Octavio, este apartado del Git me había estado dando problemas en un principio). Para la realización de esta memoria se ha utilizado “Microsoft Word”.

## Diseño

En el apartado de diseño, se explicarán las distintas clases existentes en el proyecto, así como su organización y utilidad, y a su vez se nombrarán las interfaces implementadas. Finalmente, explicaré de forma más específica el funcionamiento de la aplicación.

Antes de hablar de clases y funcionamiento, me gustaría comentar que decidí convertir el JSON proveniente de la API en clases POJO (en vez de obtener los datos con JsonObject) ya que me parece una forma más elegante, simple y ordenada de almacenar temporalmente dicha información. De igual manera, me gustaría argumentar que la introducción de datos a la base de datos se realiza mediante PreparedStatement, ya que algunos nombres de álbumes/canciones contenían la comilla simple ('), y no me funcionaba bien cuando intentaba escaparla (comenté este problema con Octavio, quien me animó a utilizar PreparedStatement).

Dentro del paquete downloader, se encuentran a su vez, otro dos paquetes: spotify y sql; además de encontrarse la clase Main y las interfaces MusicStreaming y DataBase, que las dejaremos para el final.

Empezando por el paquete spotify, éste incluye a su vez un paquete accesss, que contiene las clases necesarias para abrir una conexión con la API de Spotify, de forma que, si en un futuro la forma de acceder a la API cambiase, solo habría que centrarse en dicho paquete. Aprovecho para informar de que, para probar el programa, deberán introducir sus propios Tokens de Spotify, ya que me pareció la forma más fácil y cómoda tanto para ustedes (docentes) como para mí, ya que la otra forma era pidiendo los tokens por teclado, pero al fin y al cabo es lo mismo.

Aún dentro del paquete spotify, nos encontramos con la clase Spotify, que pide a la API ciertos recursos (artistas, canciones o álbumes) y devuelve un objeto o un ArrayList de objetos (solo en el caso de álbums y canciones). Esta clase tiene tres métodos con nombres distintos, ya que no encontré la forma de realizar sobrecarga de métodos, pues estos tenían todos el mismo parámetro (id) y al menos yo, no conseguí hacerlo. A pesar de que esta clase tiene dos responsabilidades (traerse el JSON de la API y convertirlo en un objeto), decidí dejarlo así ya que para mí tenía sentido que devolviese un objeto en vez de una string JSON.

A continuación, al mismo nivel nos encontramos todas las clases necesarias para transformar JSON en objetos: Artist, GetAlbum, Album, GetTrack y Track. La clase Artist está claro lo que contiene, pero seguramente no esté tan claro por qué existen las clases GetAlbum y GetTrack, así que procederé a explicarlo: cuando realizas un GET a la API para que te devuelva los álbumes de un artista, o te devuelva las canciones de un álbum, dichas canciones o álbumes se encuentran en un atributo llamado ítems, que es un array que contiene los dichos. Por ello, para mí, dichos ítems son ArrayLists de tipo Album, o de tipo Track, ya que eso es lo que realmente nos interesa. Como consecuencia, tuve que pensar un nombre para las otras dos clases, y lo mejor que se me ocurrió fue GetAlbum y GetTrack. Puede que sea un disparate (y seguramente lo sea), pero en mi cabeza todo esto tenía y tiene sentido. Hablando un poco de los atributos de las clases principales (Artist, Album, Track), los más interesante para mí fueron la popularidad del artista, el nombre, mercados disponibles y algunas características específicas como la fecha de lanzamiento (álbum), duración de la canción, si contiene contenido explícito...

Pasando al paquete sql, encontramos la clase SQLite, que contiene toda la lógica de dicha base de datos. Contiene un método estático para la creación de tablas y 3 métodos sobrecargados para la inserción de datos en la base de datos. A su vez, la clase contiene 3 variables de clase que indican

si las distintas tablas están o no creadas. Como ya comenté antes, la inserción se hace mediante PreparedStatement. Las distintas tablas tienen una Primary Key que se autoincrementa y que sirve para indexar los álbumes, artistas y canciones; y que además son de utilidad para buscar en otras tablas: mediante ArtistID, puedes buscar en la tabla Albums los distintos álbumes del artista y mediante AlbumID puedes buscar en la tabla Tracks las distintas canciones que conforman dicho álbum.

Por último, en el nivel más alto, al mismo nivel que los paquetes spotify o sql, encontramos dos interfaces y la clase Main. En relación con las interfaces, una de ellas es MusicStreaming (que proporciona los métodos getTrack, getAlbum y GetArtist) y la otra es DataBase (que proporciona los métodos create y add). La clase Main se encarga de tomar el array de artistas y mediante 3 for (a mi también me duele decir esto, pero no tenía ni idea de que otra cosa podía haber hecho), se encarga de controlar que todo funciona de forma correcta y que todo se introduce en la base de datos.

Con respecto a la base de datos, deben cambiar el path donde se va a crear dicha base de datos, debido a que seguramente, el path mío sea distinto a los suyos.

Ya para finalizar con el diseño (juro de verdad que ahora sí), me gustaría remarcar que cada vez que algo se introduce en la base de datos, dicha operación se muestra al usuario por pantalla según el siguiente formato: “Introducing into Tracks/Albums/Artists: {JSON recurso}”.

## Conclusiones

Ya que voy a hablar sobre mi experiencia propia, me voy a permitir hablar de forma más informal, si no les importa. Al principio, cuando se marcó este trabajo, estaba un poco perdido ya que no terminaba de entender que es lo que era una API o para que servía-mucho menos sabía cómo convertir lo que la API me devolvía en clases POJO. Algo de SQLite sabía gracias a lo que dimos de Python el año pasado, pero claro, sin información que meter en la base de datos, que supiese de SQLite no servía para nada. Pero gracias tanto a las clases de teoría como a las de prácticas (y gracias a Stack Overflow por resolverme algunos problemas), empecé a entender mejor qué era una API, y lo extremadamente útil que era. Poco a poco el proyecto fue tomando forma a medida que mis objetivos a corto plazo iban cambiando: en la primera semana ni me había descargado el código base de la práctica, lo que había hecho había sido probar a convertir JSON en clases POJO, jugar un poco con otras APIs y meter esos datos en una base de datos. Pero gracias a que empecé con un problema de escala relativamente pequeño, pasar a uno mayor como el de Spotify (donde las respuestas de la API son bastante densas), fue mucho más sencillo. He de decir que ir de menos a más era algo que nunca me había propuesto y que sin duda utilizaré en futuros proyectos, no sólo de esta asignatura, sino también en las demás. Por ello, mirando hacia atrás y preguntándome que cosas haría de otra manera, puedo decir que en líneas generales estoy bastante contento de cómo me he organizado y realizado cada paso del proyecto; aunque hay una cosa que sí que haría distinto: al principio, todo estaba en el main y este tenía todas las responsabilidades; sabía que finalmente tendría que repartir responsabilidades, pero por alguna razón, no se me ocurrió hacerlo desde un principio. Por ello, para próximos proyectos tendré esto en cuenta antes de poner de nuevo todo el código en la clase Main. Finalmente, pese a la frustración inicial de no entender bien que era un API, que tenía que hacer... Ahora que realmente lo entiendo, veo este proyecto como una situación de aprendizaje muy interesante y totalmente práctica, ya que utilizamos una API de verdad. Y esto puede parecer una bobería, pero este es el proyecto más “real” que he hecho desde que entré en esta carrera y la primera asignatura que nos muestra la utilidad real de las cosas que aprendemos con ejemplos reales (no es que las otras asignaturas sean malas o expliquen mal, sino que son mucho más teóricas). Y aunque se que mi proyecto no es perfecto, que seguramente recurra en 1523512200 cosas innecesarias, cosas que se podrían haber hecho mejor... no quería dejar de contar mi experiencia. Y, por cierto, si has leído todo el documento, gracias por tomarte la molestia; y si no, no pasa nada, lo entiendo.

## Líneas Futuras

Si miramos a líneas futuras, con un usuario final de a pie (que usa spotify sin saber lo que es una API, una base de datos SQLite...) este programa actual no tiene ninguna utilidad para él, ya que una persona que usa spotify no tiene la necesidad de guardar canciones en una base de datos. Pero creo que sí podríamos hacerles ver que este programa es útil si ayudásemos a dichos usuarios a automatizar ciertas tareas que resultan tediosas para ellos, como añadir canciones a una playlist o cambiar canciones de una playlist a otra. Navegando un poco por la web de la API de Spotify, me he dado cuenta de que no es nada complicado añadir canciones a una playlist, y es que podríamos ofrecerles a los usuarios meter en una playlist todas las canciones de los álbumes de cierto artista en muy poco tiempo. Y esto es algo que yo como usuario agradecería una barbaridad, ya que no hay nada que me aburra más que meter las canciones dentro de una playlist una por una. Investigando un poco por internet, no he sido capaz de encontrar (al menos yo), ninguna aplicación que haga esto, lo cual me sorprende pues no es extremadamente complicado.

Y ya con esto llegamos al fin de la memoria, esperando que no se le haya hecho extremadamente larga y agrediéndole su tiempo.

## Bibliografía

<https://stackoverflow.com/questions/41499935/how-to-convert-the-following-json-string-to-pojo>

Introduction to Spotify Api, Otavio Roncal; José Juan Hernández, 2022

<https://www.geeksforgeeks.org/static-method-in-interface-in-java/>

<https://developer.spotify.com/documentation/web-api/>