SPOTIFY TO SQLITE

Mariana Bordes Bueno 2ºGCID DACD EII ULPGC

Contenido

RESUMEN

El objetivo del proyecto es adquirir los datos de la base de datos de Spotify a través de la API e inyectarlos en una nueva base de datos de SQLite utilizando Java como lenguaje de programación para adquirir los datos en formato Json, extraer la información deseada y almacenarla en la nueva base de datos que hemos creado con JDBC.

He creado las clases POJO de las principales variables (Artist, Album, Track) para poder extraer la información de los Jsons que nos provee la API de Spotify y fijar los valores de cada atributo de las variables. He usado este tipo de clases porque considero que es una manera de simplificar la implementación del código principal y el manejo de los datos dado que dichas clases destacan por su simplicidad.

Primero que nada, para utilizar la base de datos debemos saber que vamos a utilizar el lenguaje SQL que está diseñado para la gestión de base de datos relacionales. Este tipo de bases de datos se caracterizan por almacenar los datos en tablas facilitando la comprensión y visualización de los datos. Para esto, importamos todas las clases de la librería java.sql donde se encuentran los métodos de JDBC que tendremos que utilizar.

La implementación del código se basa en una clase Main que hace una llamada al Controller, que como su nombre indica, se encarga de controlar y gestionar la interacción entre las clases del modelo y los accesos a la base de datos.

RECURSOS UTILIZADOS

El entorno de desarrollo utilizado ha sido Intelllj. Me he sentido muy cómoda con él principalmente porque facilita mucho el trabajo que el entorno resalte todos los errores automáticamente y que con Alt + Intro te sugiera cómo solucionarlo. Además, gracias a los shortcuts he podido programar de una manera bastante rápida y eficiente.

Con respecto a las herramientas de control de versiones, he utilizado Git. Este software diseñado por Linus Torvalds es uno de los mejores aliados de los programadores ya que te permite acceder a versiones anteriores de tu código y llevar un registro de la evolución del proyecto. La versión final del proyecto la he subido al repositorio de GitHub desde Intelllj generando un Token.

Como herramienta de documentación he utilizado la documentación de Java 11.

DISEÑO

Con respecto al patrón de diseño, he tratado de seguir el estilo arquitectónico MVC (Model View Controller) separando por una parte las clases POJO que pertenecen al modelo porque contienen lo equivalente al núcleo del código siendo estas clases la representación de los datos que se almacenan y por otra parte las clases Connect, Table, Controller, SpotifyAccessor y SpotifyAuthorization que pertenecen al Controller ya que se encargan de solicitar los datos al modelo y comunicarles a la vista en caso de haberla, pero en este caso el código no cuenta con una vista explícita como puede ser una interfaz gráfica pero el Main realiza las impresiones de pantalla que se asocian a la vista aunque tenga otras funciones como la llamada al controller.

En cuanto a los principios de diseño, he utilizado los principios SOLID como el de Single Responsability ya que cada clase tiene una única responsabilidad como por ejemplo la clase Tables se encarga única y exclusivamente de las tablas o el de Open for Extension, Closed for Modification aunque no cumple con otros como el principio de Sustitución de Liskov ya que no trabajo con herencia.

CONCLUSIONES

Considero que esta actividad me ha aportado muchísimo ya que no sólo he adquirido conocimientos sobre trabajar con APIs y programación en Java, sino que también he aprendido muchísimo sobre cómo enfrentarme a proyectos de programación ya que ha sido el primero que he tenido que hacer.

El mayor fallo que tuve fue empezar a programar el controller sin saber qué es lo que quería hacer y sin las ideas claras; el resultado fue un “código espagueti” con el que ni yo misma podía trabajar, así que empecé un nuevo proyecto. Antes de empezar este nuevo proyecto hice una lluvia de ideas para organizar todo lo que tenía pensado y poder empezar con las ideas ya estructuradas.

Algo que cambiaría si tuviera que empezar el programa de nuevo sería utilizar más el Git, ya que me parece una herramienta muy útil y su uso me hubiese facilitado el trabajo en más de una ocasión en la que sentí la necesidad de revisar versiones anteriores.

También empecé teniendo muchos fallos que eran más bien despistes como equivocarme con los nombres o faltas de comas que al principio me costaban más pero ya al final aprendí a detectarlos mucho más rápido.

LÍNEAS FUTURAS

Creo que una manera de comercializarlo sería expandiendo el código para que admita también información sobre ventas y estadísticas en la tabla de los álbumes y artistas y añadiendo también más características en los Tracks con el método GetTrack's Audio Features de la API con el que se puede adquirir muchísima información técnica sobre las canciones y vender el resultado a las productoras musicales para que sepan qué es lo que más vende y que tengan una guía para futuros contratos con artistas. Por ejemplo, si viene un nuevo artista, comparar sus tracks con los de un artista con las características de los tracks parecidas y ver la popularidad que podría tener para saber si es rentable contratarlo o no. Otro ejemplo sería si un artista va a sacar un nuevo álbum ver los datos de sus álbumes anteriores para analizar qué impacto puede tener antes de sacarlo y diseñar estrategias de marketing basándonos en las teorías que hemos concluido basándonos en los datos. También me gustaría en un futuro si se llegara a comercializar el código, añadirle una interfaz gráfica en la que poder reflejar gráficos que ayuden a la comprensión de los datos.

WEBGRAFÍA

Java Documentation
Maven Repository
SQLite tutorial
Campus DACD
API Spotify

Mariana Bordes Bueno

09/11/2022