Spotify Downloader – Memoria

Desarollo de Applicaciones para Ciencia de Datos

Grado en Ciencias e Ingeniería de Datos

Escuela de Ingeniería Informática

Universidad de Las Palmas de Gran Canaria

Resumen

Spotify Downloader es un programa capaz de acceder al API REST de Spotify, obtener datos sobre artistas, albums, y canciones, y guardarlos en una base de datos. Dado la naturaleza IO del problema, es obvio de que el límite de la aplicación será lo rápido de que puede realizar peticiones. Por esto el programa esta diseñado de forma de que el acceso a la API sea concurrente, mandando los valores que obtiene a la base de datos a medida que las peticiones se completan.

Esta concurrencia se obtiene con el uso de ReactiveX[2], más concretamente, con la librería de Java **RxJava**. Esta librería proporciona la clase `Observer<T>` y `Observable<T>`. En el que `Observable<T>` representa una corriente asíncrona de valores con una interfaz monádica, permitiendo una manera sencilla de manipular y combinar estas.

Índice

1. Recursos utilizados

Los siguientes recursos se utilizaron para la realización del proyecto:

- El IDE usado es **IntelliJ IDEA**[1]
- Para la versión de control se utilizó **git** con **GitHub**
- Para la escritura de la memoria, se utilizó LaTeX
- Para los diagramas de clase se utilizó **StarUML**

2. Diseño

2.1. Observables

Una gran parte de este programa está centrado sobre los observables. Estos se pueden pensar como si fuesen una combinación de futuros e iterables. Un iterable decora un típo `T` de forma de que en vez de tener un único valor, se tiene una corriente de valores. Y un futuro decora un típo `T` de forma de que en vez de existir en la actualidad, es un valor de que vendrá en el futuro. De esta forma, un Observable decora a un tipo en ambas de estas formas. Un observable de un típo implica de que es una corriente de valores futuros.

		Número	
		Singular	Plural
Sincronisidad	Síncrono	`T`	`Iterable<T>`
	Asíncrono	`Future<T>`	`Observable<T>`

Además, dado que este objeto dispone de métodos como `just` y `flatMap`, se pueden razonar sobre estos típos como si fuesen monadas, permitiendo una sencilla y elegante composición entre todos los diferentes observables que se pueden aparecer durante la ejecución del programa, sin tener que preocuparse por los detalles de los efectos secundarios que ocurren detras de estas.

2.2. Patrones de diseño

Se utilizan los siguientes patrones de diseño:

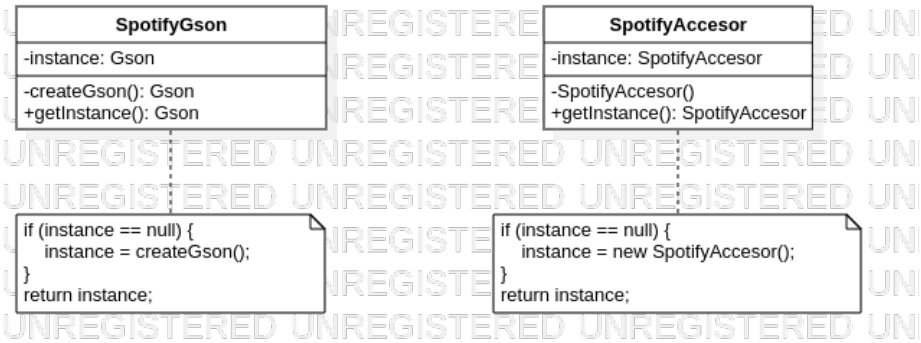
- Singleton
- Visitor
- Template

2.2.1. Singleton

Hay dos instancias en las que se utiliza el patrón de Singleton en el proyecto.

Una de estas es para la creación de un Gson que tiene registrado deserializadores personalizados para las diferentes clases que sirven para modelar el dominio.

La segunda aparición del patron de Singleton aparece en el uso de SpotifyAccesor. Esto se justifica con el hecho de que hace falta un tóken OAuth 2.0 para acceder a este, con lo que mantener el SpotifyAccesor detrás de un Singleton significa de que en cada momento solo habrá una única instancia que usa el token de acceso que devuelve Spotify al pasar por la autorización de OAuth 2.0. Además, la API de Spotify se accede desde varias partes del programa. De forma de que esto cancela la necesidad de pasar la instancia de SpotifyAccesor creada por todo el árbol de las clases a las clases que lo necesita.

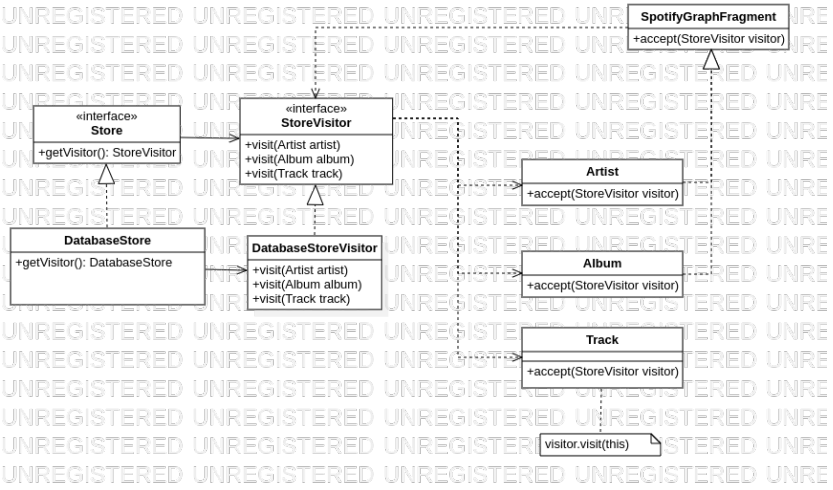


2.2.2. Visitor

El hecho de guardar los datos obtenidos de la API de Spotify se realiza mediante un patrón de Visitor.

Ya que el modelo está dividido entre autores, artistas y canciones, y además estos son pasados a la base de datos en cuanto sean generados, significa de que la base de datos tendrá que tratarlos individualmente.

Para manejar esto se utiliza el patrón de Visitor.

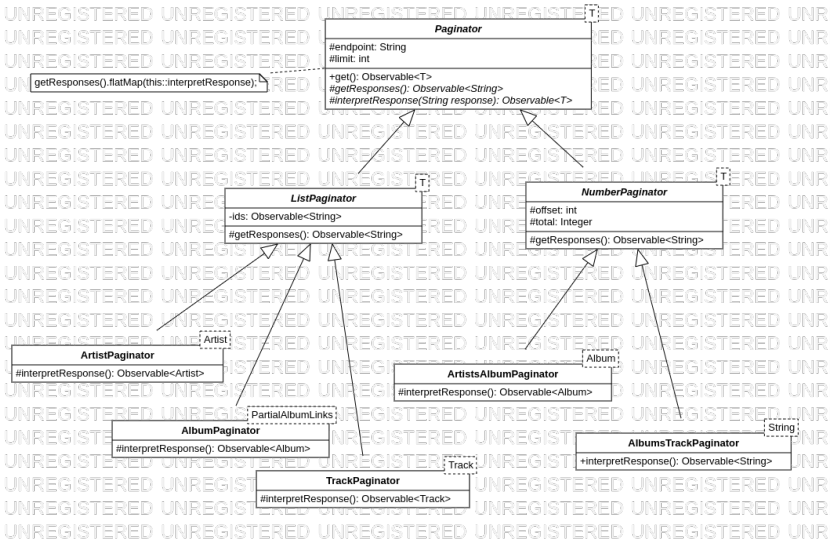


2.2.3. Template

Para acceder el API de la forma más eficiente, se hace uso de los recursos paginados que tiene.

Entre estos se pueden encontrar paginación con offset y limit, donde se pagina poniendo un offset y un limit; y paginación de listados, donde se pagina pasando todos los IDs en una lista.

Para modelar estos endpoints del API, se usa un patrón de templado para generalizar las partes comunes que tienen estos puntos de acceso paginados.



3. Conclusiones

Dado la inexperiencia de la programación concurrente en Java de los autores del programa. A lo largo de su vida de desarollo primero se implementó de una forma completamente síncrona, tal que una vez funcione este prototípo, se siga desarollando para obtener la concurrencia. Por esta historia, y la gran diferencia de paradigmas usados en el cambio de la sincronisidad del programa, la versión final puede que tenga detalles que sobrarían si se hubiera tenido en cuenta desde el principio el paradigma que se usaría para el desarollo de la aplicación.

En la actualidad, el problema más grande presente es la falta de una centralización en cuando al manejo del límite de peticiones, de forma de que si se llega al límite, todas las peticiones que se necesitan que realizar se van a ejecutar de forma que todas se ejecutarian de todos modos ignorando el hecho de que el límite ya está superado, solo descubriendo esto cuando la respuesta les indica esto. Una versión mejorada podría ser capaz de alentar peticiones aún no mandadas si se sabe de que el límite de peticiones se ha pasado.

4. Líneas futuras

Dado la baja iteractividad con un usuario que tiene la aplicación, no existe una obvia línea que tomar para comercializar el producto. A pesar de esto, el encapsulamiento y el bien rendimiento de la aplicación pueden encontrarse a ser útiles para otras aplicaciones más centradas al usuario. De forma de que pueden existir una plétora de aplicaciones comerciales que usan los datos obtenidos de Spotify, tal que todas estas pueden usar este proyecto como un módulo integral para el funcionamiento de estas.

Por lo que la mejor línea que puede tener este programa para poder comercializarse es desarollarse en una libreria bien diseñada y modular que se pueda integrar en otras aplicaciones comerciales.

Referencias

[1] *IntelliJ IDEA*. 2022. URL: `www.jetbrains.com/idea` (visitado 09-10-2022).

[2] *ReactiveX*. 2022. URL: `reactivex.io` (visitado 09-10-2022).

Aurora Zuoris

La memoria se ha creado el 9 de noviembre de 2022 con LaTeX

9 de noviembre de 2022