Sistema de recomendación de música

Lothar Soto Palma Daniel López García Ignacio Cordón Castillo

31 de mayo de 2017

Índice

1. Descripción

Nuestro sistema pide al usuario un determinado número de grupos afines (grupos 'modelo'). A partir de dichos grupos obtenemos una predicción, basándonos en grupos afines obtenidos a partir de las APIs de Youtube, Spotify y a partir de un scraping de MusicBrainz. Para los grupos sugeridos además, proporciona información biográfica, álbumes de dichos grupos, etc.

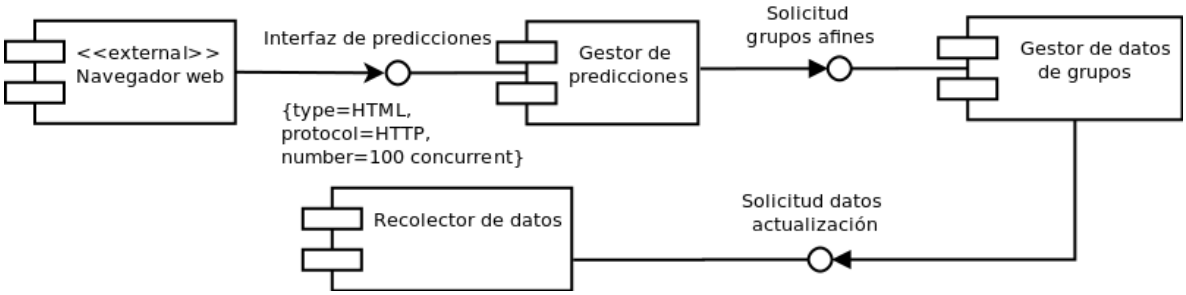
2. Vistas

2.1. Punto de vista funcional

En nuestro punto de vista funcional reseñamos los siguientes elementos:

- **Navegador web**: Elemento funcional externo, que se ejecuta del lado del cliente. Accede a través de la interfaz de predicciones, mediante peticiones HTTP al *gestor de predicciones*.
- **Gestor de predicciones**: Se ha pensado en este módulo como un elemento que sintentiza todos los grupos afines sugeridos para unos grupos modelo de entrada, escoge cuáles mostrarle al usuario y devuelve la respuesta al navegador web. Solicita al *gestor de datos* la información disponible para grupos afines a los introducidos por el usuario en el navegador web.
- **Gestor de datos de grupos**: Se encarga de acceder a los datos de grupos, y en caso de no existir o estar desactualizados, llama al *recolector de datos*. Caso opuesto devuelve la información al *gestor de predicciones*.
- **Recolector de datos**: Se encarga de acceder a los datos a través de las APIs o el *scrapeo* y los devuelve al gestor de datos de grupos para la actualización en la base de datos.

En nuestro caso se ha decidido separar en un único módulo funcional el acceso a la base de datos (escritura y lectura de información de grupos).

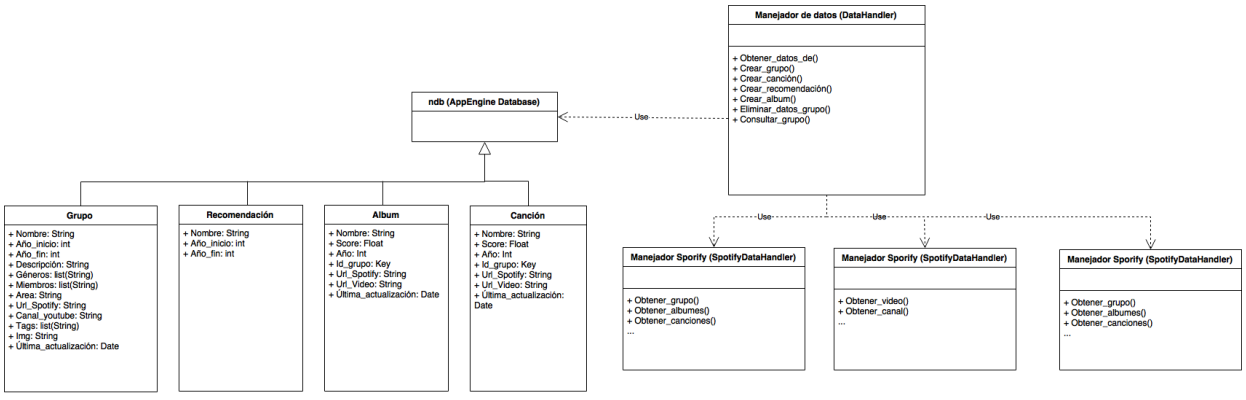


2.2. Punto de vista de informaciónn

La información del sistema se almacenará de la siguiente forma:

- **Grupo**: Clase que representa la información relativa a un grupo.
- **Album**: Clase que representa la información relativa a los albums de un grupo. Cada album está relacionado con su grupo correspondiente.
- **Canción**: Clase que representa la información relativa a las canciones de un Album. Cada canción está relacionada con su album correspondiente.
- **Recomendación(Base de datos)**: Clase que representa la información relativa a las recomendaciones de los grupos consultados en el sistema.
- **Manejador de datos de Spotify**: Clase que se encarga de la recolección de datos de Spotify, haciendo uso de una API determinada.

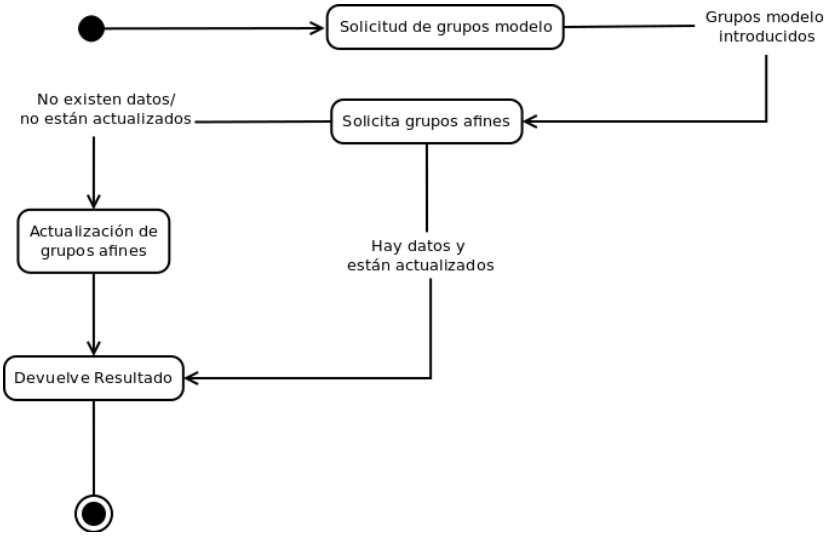
- **Manejador de datos de Youtube**: Clase que se encarga de la recolección de datos de Youtube, haciendo uso de una API determinada.
- **Manejador de datos de MusicBrainz**: Clase que se encarga de la recolección de datos de Musicbrainz, realizando un scraping sobre la misma.
- **Manejador de datos**: Clase que se encarga de la gestión de los datos de la base de datos, es la encargada de realizar consultas e inserciones sobre la misma.
- **Recomendación**: Clase para representar la gestión de predicciones. Dado que solo usaremos una instancia del recomendador usaremos una clase singleton.



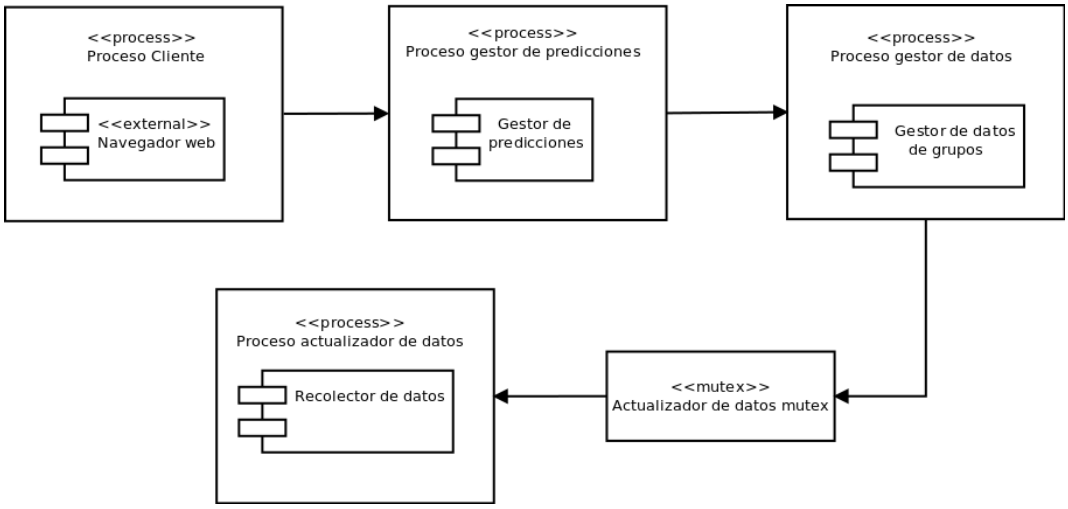
2.3. Punto de vista de concurrencia

Hay varios estados por los que tiene que pasar el sistema:

- **Solicitud de grupos modelo**: El usuario debe introducir una serie de grupos que ya le gustan.
- **Solicitud de grupos afines**: Una vez se han introducido los grupos modelo, se solicitan los grupos afines al sistema.
- **Devuelve resultado**: En caso de que haya datos para los grupos modelo introducidos, se devuelve el resultado y llegamos al estado final.
- **Actualización de grupos afines**: Caso de que no existan datos, o lleven sin actualizarse un determinado tiempo, se recolectan los datos y se escriben en la base de datos, y se procede al estado *Devuelve resultado*.

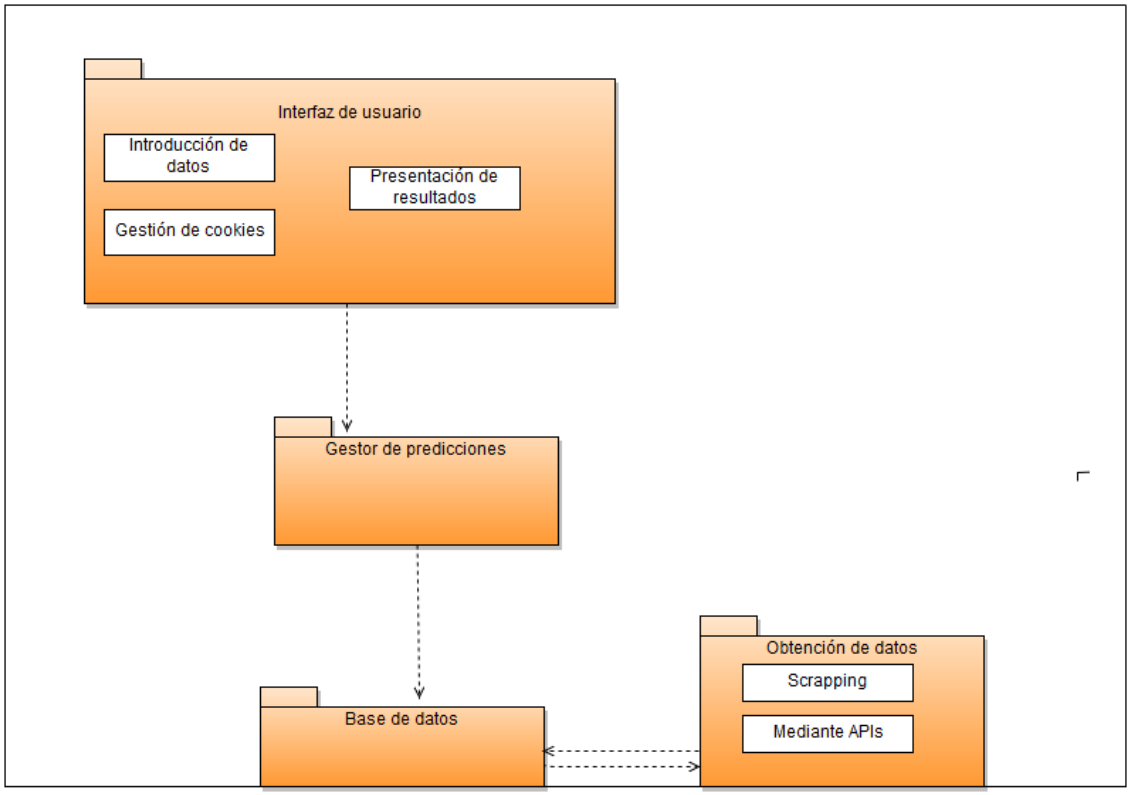


Debería existir una actualización concurrente de los datos de grupos, caso de que no existan en el sistema o estén desactualizados. Hemos tratado de modelar esto en el siguiente diagrama:



Finalmente el uso de un proceso *cron* para actualizar la base de datos no es viable haciendo uso de de la capacidad aportada de manera gratuita por google app engine, debido a que

2.4. Punto de vista de desarrollo



2.5. Punto de vista de despliegue

En esta vista se describe el entorno en el que el sistema será desplegado, incluyendo cada una de las dependencias del mismo. Para este apartado es necesario considerar los siguientes puntos:

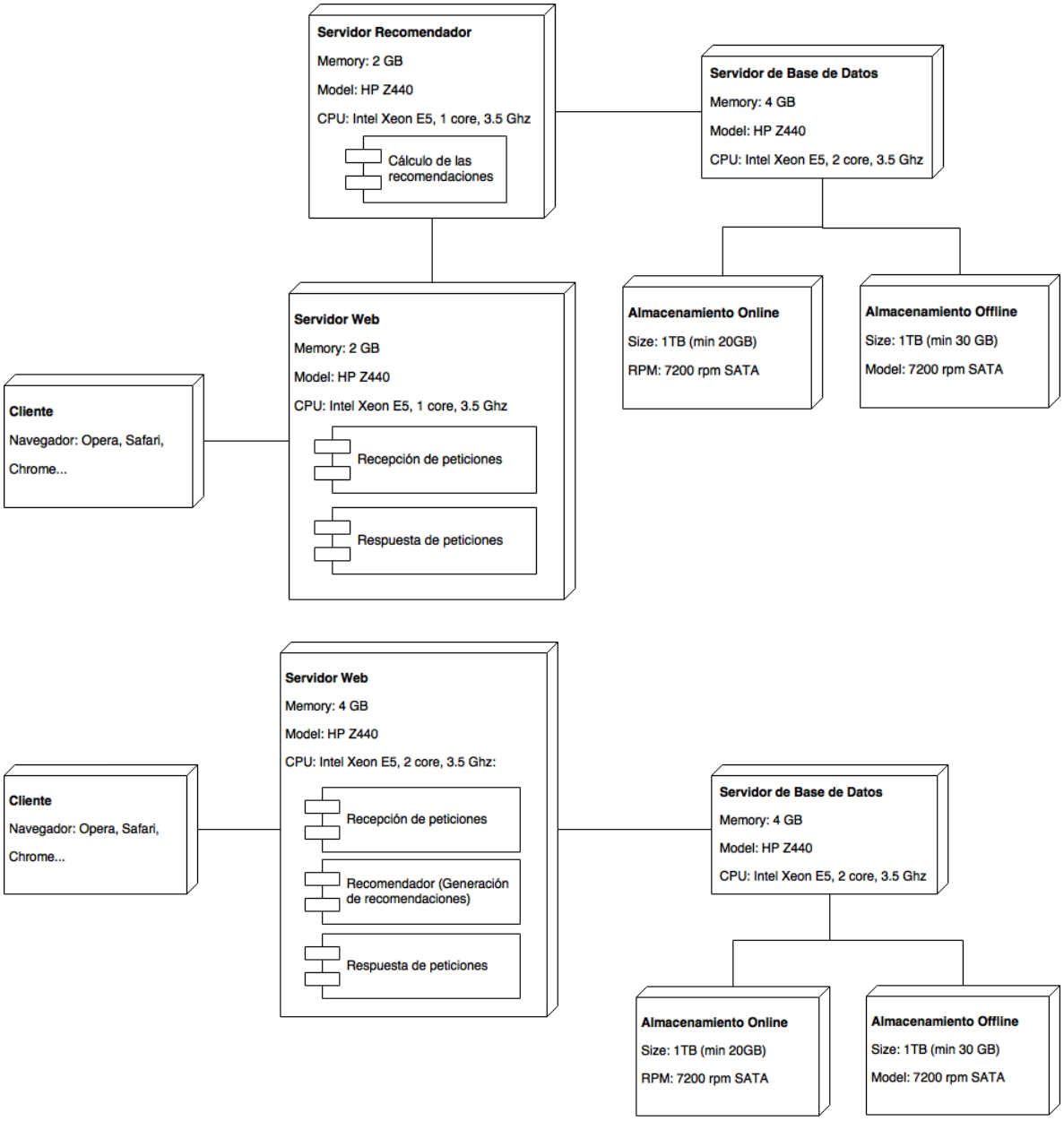
- Tipo de Hardware requerido.
- Especificaciones y precios.
- Software de terceros utilizado en el sistema.
- Compatibilidad de las tecnologías utilizadas en el sistema.
- Requisitos y capacidad de la red.

Se han considerado los siguientes nodos para el diagrama de despliegue:

- Nodos de procesamiento principales:
  - **Servidor recomendador:** Parte del sistema encargado de procesar a partir de la información de la base de datos y la proporcionada por la interacción del usuario final a través del cliente.
  - **Servidor de base de datos:** Parte del sistema encargado de almacenar los datos de la música obtenida a través de software de terceros, en particular +2 APIs y al menos un scrapping. Es necesario la extracción de datos e integración de los datos.
  - **Servidor Web:** Parte del sistema encargado de la recepción y respuesta de las peticiones generadas por el usuario final a través del cliente. El servidor web procesa la petición del usuario y la envía al sistema recomendador con la finalidad de generar una respuesta.
- Nodos de cliente:
  - **Navegador:** El acceso al sistema por parte del usuario final se realiza a través de cualquier navegador web (p.e.: Opera, Edge, Safari, Chrome, Firefox,...)
- Nodos de almacenamiento:
  - **Almacenamiento online:** Parte del sistema que se encarga de almacenar la información principal generada por el sistema.
  - **Almacenamiento offline:** Parte del sistema que se encarga de almacenar las copias de seguridad del sistema, para evitar pérdida de funcionalidad a la hora de una posible mala migración de datos o ataques al servidor.
- Conexión de red.

Reflexionando sobre los principales cuellos de botella de una aplicación web, estos son claramente los servidores y su actividad, por lo que es muy posible que el servidor que se encarga de la recomendación y el servidor web se integren en uno solo, ya que la interacción entre ellos debe ser rápida.

En el primer diagrama se considera el servidor recomendador de manera independiente del servidor web y en el segundo caso el servidor web tiene un módulo que realiza las tareas del recomendación, sin embargo estas agrupaciones pueden llegar a ser lógicas y los tres servidores pueden actuar en una única máquina en la que a cada servidor se le asigna una cantidad determinada de recursos físicos.



Esta interpretación del sistema desde el punto de vista del despliegue se realiza sin tener en cuenta el servicio Google App Engine que se usará en la práctica para trabajar, puesto que el servicio sustituirá esta parte del diseño.

2.6. Punto de vista operacional

En esta vista se describe como el sistema funciona, será administrado y mantenido cuando el sistema está funcionando en su entorno de producción. Es necesario considerar los siguientes puntos:

- Instalación y actualización.
- Migración de funcionalidades y datos.
- Backup y restauración del sistema.

2.6.1. Instalación y actualización

El modelo de instalación para el sistema de recomendación de música debe constar de los siguientes elementos instalados en el siguiente orden:

- Python 2.7: Contiene todo el software necesario para desarrollar el sistema de recomendación, la instalación depende del sistema:
  - Windows: Es necesario descargar e instalar el software a partir del archivo binario obtenido de la web oficial de python: https://www.python.org/downloads/windows/
  - Mac OS X: Es necesario descargar e instalar el software a partir del archivo binario obtenido de la web oficial de python: https://www.python.org/downloads/mac-osx/
  - Linux/UNIX: Es necesario descargar e instalar a través de la terminal el software a partir de los archivos obtenido de la web oficial de python: https://www.python.org/downloads/source/
- Librería flask para python: Una vez instalado python la librería se instala ejecutando el comando `pip install flask`.
- Librería spotipy para python: Una vez instalado python la librería se instala ejecutando el comando `pip install spotipy`.
- Librería google-api-python-client para python: Una vez instalado python la librería se instala ejecutando el comando `pip install google-api-python-client`.
- Librería requests para python: Una vez instalado python la librería se instala ejecutando el comando `pip install requests`.
- Librería fuzzywuzzy para python: Una vez instalado python la librería se instala ejecutando el comando `pip install fuzzywuzzy`.
- Librería python-Levenshtein para python: Una vez instalado python la librería se instala ejecutando el comando `pip install python-Levenshtein`.
- Librería python-dateutil para python: Una vez instalado python la librería se instala ejecutando el comando `pip install python-dateutil`.

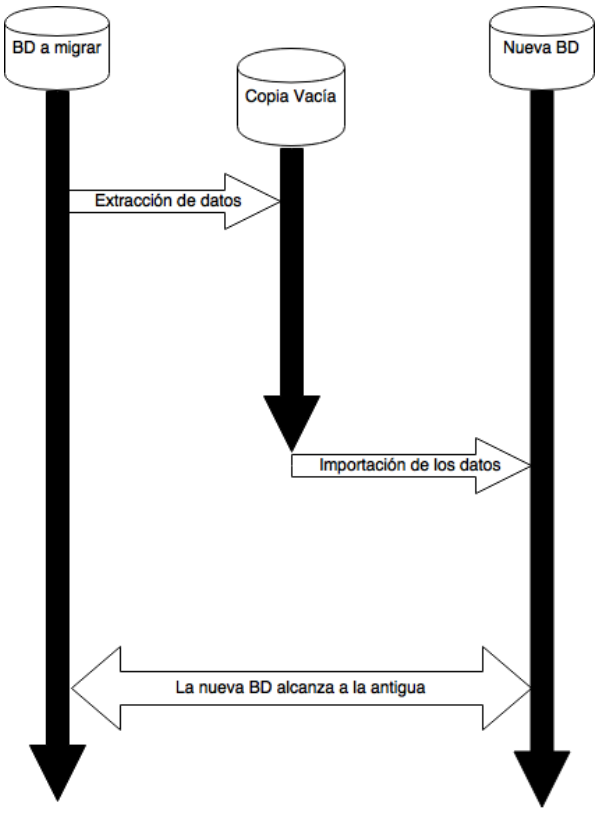
- Base de datos: Google App engine trabaja directamente con una base de datos noSQL, tan solo sería necesario instalar el esquema de base de datos en el sistema.
- Librería del sistema: Inclusión de las librerías desarrolladas para realizar la recomendación, recolección y gestión de datos.

Se ha desarrollado un archivo makefile que permite realizar la configuración de las librerías de python necesarias para la ejecución del sistema, además de permitir montar localmente el sistema.

2.6.2. Migración del sistema y los datos

La migración de las funcionalidades podrían realizarse de manera que la antigua versión del sistema se continúe ejecutando de forma paralela a la del sistema con las nuevas o mejoradas funcionalidades, esto ocurre hasta que este último este totalmente operativo entonces se para la ejecución del sistema anterior. Esto es debido a que en nuestro sistema no se realizan peticiones críticas que necesiten de constante rigor. Aunque también sería posible adoptar el modelo big bang, es decir reinicios programados cada cierto tiempo que reinicien las funcionalidades del sistema para aplicar las actualizaciones o añadir nuevas funcionalidades.

La migración de datos es una operación más crítica en caso de que deba realizarse durante el funcionamiento del sistema, en el caso del modelo big bang es tan sencillo como realizar una copia de la base de datos con el nombre deseado mientras que el sistema está apagado, estaría programado para que se realizara antes del inicio del sistema. En el caso de que la migración se produzca durante la ejecución del sistema se realiza creando una nueva base de datos, se extraen los datos actuales de la anterior base de datos y posteriormente se cargan en la nueva base de datos en el tiempo, se irán realizando las mismas operaciones en ambas bases de datos hasta que las dos llegan al punto en que son iguales y se continua con la nueva base de datos tal y como se refleja en el siguiente gráfico:



2.6.3. Backup y restauración del sistema

El sistema realizará de forma programada copias de la base de datos y se almacenará en un disco duro no conectada a la red, cuya única finalidad es almacenar los datos fechados de cada copia de la base de datos. En caso de que sea necesario una restauración de la base de datos se podría realizar la misma técnica empleada en la migración pero en sentido opuesto.

Para la restauración del sistema se barajan dos posibilidades:

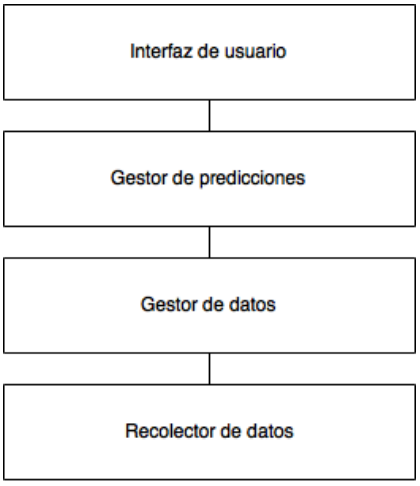
- La instalación de una réplica del servidor actual, para que en el caso de una posible desconexión del servidor principal, este entre en acción hasta la activación de este último.
- No dar servicio durante la caida del servidor, con la finalidad de disminuir el coste inicial del sistema.

3. Diseño arquitectonico

El sistema se diseñará siguiendo una arquitectura por capas. Esta decisión se debe a que se pueden identificar claramente varias partes del sistema que pueden ser desarrolladas independientemente. El

sistema consta de 3 capas:

- **Interfaz de usuario**: Su labor es mostrar información al usuario y comunicarse con él. Esta capa se comunica con la capa de gestión de predicciones obteniendo los datos que deben ser mostrados.
- **Gestor de predicciones**: En esta capa se lleva a cabo la funcionalidad del sistema. Se comunica con la capa de gestión de datos para construir una recomendación basandose en lo introducido por el usuario.
- **Gestor de datos**: Es la encargada realizar las consultas sobre la base de datos además de las inserciones de grupos, albumes y canciones. También incorpora funcionalidades para realizar la integración de los datos obtenidos a partir del gestor de recolección de datos.
- **Recolección de datos**: Almacena los manejadores de datos de Spotify, Musicbrainz y Youtube por lo que es el que se va a encargar de la recolección de los datos haciendo uso de las APIs y el scraping.



3.1. Justificación - Arquitectura basada en capas

La arquitectura basada en capas https://en.wikipedia.org/wiki/Multitier_architecture es una arquitectura cuyo objetivo primordial es la separación de la lógica funcional y de la lógica de diseño. El desarrollo del modelo puede llevarse a cabo en diversos niveles de forma que si se requiere modificar alguna parte del sistema, este no tiene que revisarse completamente, sino que solo se revisa la parte necesaria.

La selección de este modelo es debido a la clara diferenciación de las partes de nuestro sistema. No requerimos por ejemplo que varias partes de nuestro sistema modifiquen los datos, necesitamos un único módulo que obtenga e integre los datos.

Otros diseños considerados:

- Modelo no relacional: en principio se asemejaba mucho a la clase de sistema de información que queriamos desarrollar, pero consideramos que el uso de herramientas como mapReduce para este sistema se excede con respecto al uso final del mismo.
- Modelo multidimensional: descartado debido a las limitaciones sobre el cálculo y ancho de banda impuestas por AppEngine. Se decide implementar la actualización de datos de manera que se realice bajo demanda por parte de la aplicación de los mismos, y se descarta emplear un procedimiento ETL que por ejemplo actualicase los datos disponibles en base de datos durante el periodo de menor uso del sistema.
- Flujo de datos: a pesar de ajustarse bastante bien al diseño de nuestro sistema, su menor capacidad de modularización en caso de querer ampliarse la aplicación ha sido un factor determinante para escoger el modelo de capas.