Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

The Music Collectorship

Pau Almirall Ruiz

Tutors: Francesco Ciompi i
Santi Seguí Mesquida.
Realitzat a: Departament de Matemàtica
Aplicada i Anàlisi.

Barcelona, 17 de gener de 2013

Agradecimientos

Antes de documentar el proyecto “The Music Collectorship” me gustaría agradecer a una serie de personas las cuales han sido partícipes en la realización del proyecto de una manera u otra.

Primeramente agradecer a los tutores del proyecto, Francesco Ciompi y Santiago Seguí Mesquida, el apoyo dado durante todo el proceso, así como las ideas y las correcciones que han aportado que han hecho que la aplicación se haya podido realizar de la mejor manera posible.

Seguidamente me gustaría agradecer a mi familia su comprensión, sobretodo en los momentos más críticos, y apoyo incondicional en todo momento.

Por último me gustaría agradecer a los compañeros con los que he compartido todo este tiempo de realización del proyecto el haber podido hacer este trabajo en compañía, lo cual ha hecho que el proceso fuese mucho más ameno y divertido.

Índice

Abstract (Castellano)

El proyecto que va a ser documentado a continuación, llamado “The Music Collectorship”, es un servicio web destinado a recolectar y ofrecer toda la información disponible relacionada con un artista musical.

Dada la gran variedad de servicios existentes en el mercado ofreciendo diferente tipo de información relacionada con la música y siendo el internet y la música una dupla cada vez más unida, se decidió que se podría englobar varios de estos servicios en uno solo, que recogiera la información de dichos servicios y los mostrara por un único canal.

De esta manera se creó esta aplicación, con el deseo de aportar un valor añadido a las ya existentes, combinando la información recogida a través de estos canales diferentes para que resultase mucho más útil e interesante.

La principal función de la aplicación es guardar en una base de datos toda la información del artista que se obtenga utilizando una serie de APIs externas. Una vez el usuario busque un artista, este puede estar ya almacenado en la base de datos del servicio o no, aunque en cualquiera de los dos casos, se mostraría la misma información.

Al obtener su información a través de APIs de otros servicios web, ya estén relacionados con la música o no, se ha hecho especial hincapié en la robustez del servicio, para evitar problemas con los datos extraídos.

El usuario final que acceda a la aplicación, una vez haya introducido el nombre del artista tendrá acceso a diferentes tipos de información que se han separado en apartados que muestran datos como las últimas noticias relacionadas con el artista, su información general, los conciertos que van a realizar, las *setlists* de los conciertos que han realizado y los álbumes que han publicado con información detallada.

Abstract (Català)

El projecte que serà documentat a continuació, anomenat "The Music Collectorship", és un servei web destinat a recollir i oferir tota la informació disponible relacionada amb un artista musical.

Donada la gran varietat de serveis existents en el mercat oferint diferent tipus d'informació relacionada amb la música i sent l'internet i la música una dupla cada vegada més unida, es va decidir que es podria englobar diversos d'aquests serveis en un de sol, que recollís la informació d'aquests serveis i els mostrés per un únic canal.

D'aquesta manera es va crear aquesta aplicació, amb la intenció d'aportar un valor afegit a les ja existents, combinant la informació recollida a través d'aquests canals diferents per a que resultés molt més útil i interessant.

La principal funció de l'aplicació és guardar en una base de dades tota la informació de l'artista que s'obtingui utilitzant una serie d'APIs. Un cop l'usuari busqui un artista, aquest pot estar ja emmagatzemat a la base de dades del servei o no, tot i que en qualsevol dels dos casos, es mostraria la mateixa informació.

Al obtenir la seva informació a través d'APIs d'altres serveis web, ja estiguin relacionats amb la música o no, s'ha posat èmfasi en la robustesa del servei, per evitar problemes amb les dades extretes.

L'usuari final que accedeixi a l'aplicació, un cop introduït el nom de l'artista tindrà accés a diferents tipus d'informació que s'han separat en apartats que mostren dades com les últimes notícies relacionades amb l'artista, la seva informació general, els concerts que realitzaran, les *setlists* dels concerts que han realitzat i els àlbums que han publicat amb informació detallada.

Abstract (English)

The project that will be documented below, called "The Music Collectorship" is a web service designed to collect and provide all available information related to a music artist.

Given the wide range of services available in the market offering different types of information related to music and being the internet and music a couple more close than ever, we decided that it could be great to englobe some of these services into one, collecting the information of these services and show them through a single channel.

This application was created with the desire to add value to the existing ones, combining the information gathered through these different channels and make it much more useful and interesting to the user.

The main function of the application is to store in a database all the artist information that is obtained using some external APIs. Once the user searches for an artist, it may be already stored in the database of the service or not, but in either case, it will display the same information.

By getting the information through web services APIs, whether music related or not, has placed particular emphasis on the robustness of the service, to avoid problems with the extracted data.

The end user who accesses the application, once the artist is searched has access to different types of information separated into sections that show data like the latest news related with the artist, his overview, concerts to be performed, the setlists of the past shows and the released albums with detailed information.

1 - Introducción

1.1 - Objetivo

El objetivo inicial de este proyecto fue desarrollar un motor de búsqueda que partiendo de los artistas musicales, fuera recolectando información a través de diferentes servicios existentes en la red y rellenando una base de datos con toda esa información preparada para un futuro uso.

Se mantuvo dicho propósito hasta que la aplicación ya empezó a disponer de una cantidad destacable de información por cada búsqueda que se realizaba. En ese momento se propuso que, aunque fuese para comprobar que los resultados coincidían con los deseados se podría incluir una interfaz web.

Con el hecho de incluir la interfaz, el proyecto pasó de ser un motor de búsqueda a ser un servicio web, y nos planteamos la posibilidad de añadir algunas funcionalidades que no se contemplaban con los objetivos iniciales. Se incluyó la posibilidad de hacer peticiones a otros servicios desde *Javascript*, como *Google News* o *Twitter*, ya que al ser informaciones muy inmediatas no debían guardarse en la base de datos.

Así que con todo esto, los objetivos del proyecto pasaron a ser los de desarrollar un servicio web de búsqueda de artistas musicales, los cuales podían estar previamente almacenados en una base de datos o no, y mostrar toda su información en una página web que organizase toda esta información.

Aparte del objetivo principal, dos de las premisas que se han tratado de seguir a lo largo del desarrollo de la aplicación y que han marcado claramente el diseño de esta son, por un lado, tratar de disponer de cuantos más artistas mejor y que estos incluyan toda la información posible y por otro lado utilizar cuantos más servicios externos mejor, ya que dichos servicios son los que nutren la base de datos y aportan valor a la aplicación.
Estos objetivos se han marcado dado que ya existen aplicaciones en el mercado que ofrecen un servicio similar, una de ellas es *Last.fm*, de la cual se

nutre la aplicación para conseguir parte de la información. Así que para que "The Music Collectorship" tuviese un valor añadido debía tener otras funcionalidades y aportar mejoras que no estuvieran en *Last.fm*, y esto ha sido algo que ha estado en el punto de mira durante el desarrollo de todo el proyecto.

1.2 - Motivación

Este proyecto plantea la posibilidad de tener un servicio capaz de contener información detallada de la gran mayoría de artistas existentes. Combinando los tipos de informaciones a las que tiene acceso la aplicación, esta podría convertirse en un servicio de consulta de referencia para los aficionados a la música, debido a su sencillez de manejo y la completa base de datos que contiene, permitiendo incluso ser ampliada en un futuro.

Es un servicio que requiere un mantenimiento muy bajo, debido a que todo su contenido se obtiene automáticamente, y que su resultado final resulta muy atractivo para los usuarios, ya que son capaces de ver todas las publicaciones de los artistas consultados y ver o escuchar un video para cada canción, hecho que no es posible de hacer en muchos servicios similares.

A nivel personal tenia claro que el proyecto debía tratarse sobre un tema que fuese una afición para mi y que pudiese desarrollarlo en un lenguaje en el cual me sintiese cómodo.

La música siempre ha sido una de mis grandes aficiones, así que cuando encontré un proyecto relacionado con la música el cual podía desarrollar en *Python*, supe que no me iba a faltar motivación para realizar el proyecto.

Una vez hecha la primera toma de contacto y haber definido los objetivos del proyecto se podía observar que la aplicación duplicaría funcionalidades ya existentes en uno de los servicios externos tomados como referencia, *Last.fm*. Con este hándicap una de las motivaciones principales tenía que ser el aportar al usuario información que *Last.fm* no aporta a sus usuarios para darle a nuestra aplicación un valor añadido, y así fue durante el desarrollo del proyecto.

También fue una motivación para mi el hecho de poder diseñar desde cero la aplicación, pudiendo elegir desde el lenguaje en el cual desarrollar el servidor,

hasta el estilo de la página web, pasando por el *framework* utilizado para lanzar el servidor, el ORM (Object Relational Mapper) para conectar el servidor con la base de datos, etcétera.

Por último, y aunque fue una situación que se planteó un poco más adelante, el hecho de poder desarrollar parte de las funcionalidades de la aplicación en *Javascript* fue una motivación extra, ya que podía implementar parte de mis conocimientos en *Javascript*.

1.3 - Estado del arte

1.3.1 - Música e internet

La música e internet han tenido una relación muy complicada debido a las descargas de canciones, las cuales eran las culpables de la perdida de beneficios según el sector de las discográficas.

Esto llevó a la industria musical a tener una posición de escepticismo respecto a lo que internet le podría aportar. Por este motivo, tuvieron que dar un paso adelante los propios usuarios de internet y crear herramientas que aportasen un contenido adicional a la música en si.

Con ello nacieron los primeros servicios que permitían por ejemplo escuchar radio por internet (RealAudio, 1995), “Fan Pages” de artistas de todo tipo, páginas con largas colecciones de letras de canciones, etcétera.

Con la web 2.0 se produjo la autentica revolución de internet, y por supuesto también en el mundo de la música.

En 2003 se lanzó *Myspace*, el primer gran servicio 2.0 que permitía que los usuarios tuviesen un contacto más directo con los artistas. Este portal permitía a los usuarios hacerse un perfil con sus gustos, fotos, canciones favoritas.

También podían hacerse “amigos” de sus artistas favoritos, los cuales también tenían sus propios perfiles donde publicaban sus canciones más conocidas para ser escuchadas en *streaming*, sus conciertos, fotos promocionales, etcétera.

Myspace revolucionó tanto el sector, que en Estados Unidos llegó a tener más visitas que *Google* en junio del 2006 y fue la red social más visitada del mundo entre 2005 y 2008, hasta ser desbancada por *Facebook*.

Otro gran suceso que unió aun más a la música y el internet fue la aparición de los reproductores MP3. Con ellos empezaron a emerger cantidad de tiendas online de venta de canciones, como *iTunes*, *Amazon MP3* o *MP3.com*.

Más relacionado con el presente proyecto, se puede mencionar *Last.fm*, un completísimo portal musical que dispone de información muy detallada de la mayoría de artistas existentes y que dispone de unas funcionalidades muy atractivas para el usuario, como una radio en *streaming* personalizada según los gustos o la posibilidad de conocer personas con un gusto musical muy parecido.

Aunque *Last.fm* se fundase en 2002, no fue hasta 2005 que adquirió presencia en la web, y fue gracias a su fusión con otro proyecto, el llamado *Audioscrobbler*. Con esto, el usuario de *Last.fm* podía tener contabilizadas todas las canciones que escuchaba y eso permitía que el sistema de recomendación fuese muy eficiente.

1.3.2 - Estado actual

Hoy en día, y gracias a la web social, se han multiplicado y especializado los servicios web relacionados con los contenidos musicales. Desde aplicaciones para conocer/ligar con personas con gustos similares hasta servicios que te informan de todos los conciertos que hay a tu alrededor dependiendo de la música que escuches. Se podrían separar incluso por categorías, las cuales serían las siguientes:

- Reproductores en streaming

Spotify, *Grooveshark* y *Musicuo* son 3 servicios que permiten escuchar cualquier canción en *streaming* que esté almacenada en sus servidores. Permiten hacerse un usuario, crear listas y otras muchas funcionalidades.

De estos tres servicios *Spotify* es el que más ha perdurado, ya que algunos otros aparecieron con anterioridad pero fueron cerrados debido a que no se ceñían a la legalidad.

Otro tipo de solución en *streaming* es la que nos ofrecen servicios como *Amazon Cloud Player*, *Google Music* o *iTunes Match*, donde el usuario puede tener su colección disponible en la nube, pero ha de disponer de ella previamente, o haberla comprado.

- Radios por internet

Desde *RealAudio* a *Musicovery*, pasando por el propio *Last.fm* o *Spotify*, son muchos los servicios de estaciones de radio por internet. El servicio más popular y también de los más longevos, aunque no está activo en España, es

Pandora. Dicha aplicación decide que canciones debe escuchar el usuario dependiendo de una aproximación inicial introducida por el usuario y con el *feedback* positivo o negativo que el usuario puede darle a cada canción.

- Otros servicios

Aparte de las grandes comunidades como *Myspace* o *Last.fm*, las cuales reúnen muchas de las funcionalidades aquí explicadas por separado, hay páginas o aplicaciones enfocadas a ofrecer un servicio más especifico.
Es el caso de *TuneWiki*, *Shazam*, *Songkick*, *Setlist.fm*, *Tastebuds*, etcétera.

Algunos de estos servicios han sido utilizados para realizar el proyecto de “The Music Collectorship” gracias a que proporcionaban una API con la que trabajar.

1.4 - Tecnologias empleadas

1.4.1 - Programación web

Para la parte de cliente se ha implementado una página web con un buscador y otra con los resultados del artista seleccionado. Para desarrollar estas páginas se ha intentado diferenciar bien las capas de contenido, presentación y lógica, y se ha hecho uso de las siguientes tecnologías:

- **Flask:**

  Flask es un *framework* para aplicaciones web desarrolladas en *Python* y distribuido bajo una licencia BDS (Berkeley Software Distribution). Se le denomina *microframework* dado que se caracteriza por ser muy ligero, su núcleo es muy sencillo, pero a la vez es extensible por tal de poder abarcar la mayoría de las necesidades del desarrollador.

  Este *framework*, que lanzó su primera versión en abril del 2010 es hoy en día, junto a Django, uno de los más empleados para el desarrollo web en *Python*.

- **HTML5:**

  Se ha procurado que la página sea lo más semántica posible aprovechado todas las nuevas etiquetas de marcado que proporciona la última versión *HTML*.

La quinta versión de este lenguaje de marcado apareció como “Working Draft” en enero del 2008. En diciembre de 2012 el *W3C* (World Wide Web Consortium) cambió su estado a “Candidate Recommendation”, así que ya está en su fase final de desarrollo, aunque todavía no se ha establecido como estándar. Esta versión supone una pequeña revolución en la web añadiendo muchas funcionalidades.

La primera de ellas, que es la que se ha aprovechado durante el desarrollo del proyecto, es la incorporación de nuevas etiquetas por tal de hacer la web mucho más semántica y accesible (un lector automático ha de ser capaz de interpretar correctamente cada elemento de la página). Por este motivo, se ha marginado el uso de etiquetas como el *div* por otras que tienen un mayor significado (*header, article*, *aside*, *section*).

El resto de novedades que aporta *HTML5* se engloban en un conjunto de APIs accesibles desde *Javascript*, algunas de ellas muy útiles que aportan funcionalidades relacionadas con la geolocalización, la posibilidad de trabajar offline, tener una pequeña base de datos en el cliente, ejecutar hilos de ejecución en paralelo, utilización de *websockets*, etcétera.

- **CSS3:**

Se han utilizado propiedades en las hojas de estilo exclusivas de esta última versión, todavía en desarrollo, por tal de hacer un diseño más vistoso y funcional.

Esta versión del lenguaje lleva en desarrollo desde 1999, cuando el *W3C* lanzó su primer "Working Draft". Hoy en día todavía hay muchas propiedades en desarrollo que no pueden ser utilizadas desde los navegadores convencionales.

En *CSS3* se han añadido o mejorado muchas propiedades, y solo algunas de ellas han sido implementadas en la aplicación, como se detalla en el apartado 3.4.3.

Algunas de las propiedades que más se utilizan entre los desarrolladores web son las *Media-Queries*, que han permitido que haya páginas con *responsive design* (diseño sensible), los nuevos tipos de selectores, la posibilidad de aplicar transparencias con la propiedad *opacity*, sombreados o bordes redondeados han hecho posible que las páginas puedan tener un diseño mucho más vistoso de una manera mucho más adecuada (evitando utilizar imágenes que simulen propiedades que no existían).

- **Javascript:**

Se ha utilizado *Javascript* tanto para controlar el comportamiento de la página, como para añadir funcionalidades a la propia aplicación realizando peticiones *AJAX cross-domain*.

Javascript fue lanzado por *Nestcape* en 1995 debido a rápida expansión del mundo web, que produjo que hubieses aplicaciones web más complejas. Al no haber ningún lenguaje interpretable por el navegador, toda la gestión se debía hacer desde servidor, y eso producía unas

dificultades que la empresa que dominaba el sector de los navegadores decidió solucionar.

Este es un lenguaje interpretado, orientado a objetos, débilmente tipado, con lo que resulta fácil de programar, y que actualmente cuenta con muchas l ibrerías y *plugins* para que usuarios no expertos puedan aprovechar al máximo las posibilidades del lenguaje.

Dado que *Javascript* fue creado en 10 días, es un lenguaje no demasiado robusto, y tiene problemas de base en su interior, algunos de ellos bastante curiosos (`0.1 + 0.2 = 0.30000000000000004`). Estos defectos, junto con otros problemas derivados al uso de librerías de terceros, pueden producir problemas o vulnerabilidades en la página.

Por estos problemas, junto con el hecho que es un código que se ejecuta desde el cliente, se ha tenido especialmente cuidado a la hora de desarrollar esta parte de la aplicación como se detalla en el apartado 3.4.4.

1.4.2 - Bases de datos

En el lado de servidor, hay una constante comunicación con la base de datos. Esto motiva a que la base de datos esté diseñada en un lenguaje específico y a que se utilicen herramientas para agilizar la asimilación entre las clases diseñadas en la aplicación y las tablas de la base de datos.

La base de datos está desarrollada en *MySQL*, ya que es un sistema con el que ya tenia experiencia y tiene la capacidad para asumir todo el trabajo que pueda tener que llegar a soportar.

Para agilizar la comunicación se ha utilizado *SQLAlchemy*, una herramienta que proporciona un ORM (Object relational mapping) entre la aplicación y la base de datos. Se optó por esta opción ya que no se utiliza ningún *framework* que incorpore el ORM, y se consideró necesario debido a que había una considerable comunicación entre la aplicación y la base de datos.

1.4.3 - Entorno de desarrollo

El entorno donde se ha desarrollado este proyecto ha sido un sistema *Linux* (Ubuntu 11.10) dado que es el sistema operativo en el cual me siento más cómodo para desarrollar y es totalmente compatible con todas las herramientas que se han utilizado.

El editor de texto llamado *Sublime Text 2* ha sido el elegido para programar, ya que estoy familiarizado con su uso y permite añadir atajos y *plugins* para facilitar la tarea al escribir el código.

Otra aplicación que se ha utilizado es *Terminator*, un terminar de sistema muy configurable. Se ha utilizado para lanzar el servicio o trabajar directamente con la base de datos cuando ha sido necesario.

A la hora de hacer pruebas de código se ha utilizado el interprete interactivo *iPython*, el cual incluye funcionalidades extra al interprete por defecto del lenguaje.

Las pruebas en el navegador se han realizado desde *Firefox* y *Chromium*, ya que son los dos navegadores disponibles para el sistema *Ubuntu* que más se utilizan.

1.4.4 - APIs

El trabajo fundamental de este proyecto es trabajar con APIs de otros servicios, cuantos más mejor, así que se ha intentado añadir el máximo número de APIs para ofrecer un mayor valor a la aplicación. A continuación van a ser enumerados todos los servicios que han sido utilizados, ya sea en forma de librería en la propia aplicación o a través de peticiones *http*.

El siguiente gráfico muestra una abstracción del funcionamiento general de la aplicación, mostrando su relación con las APIs que a continuación van a ser detalladas.

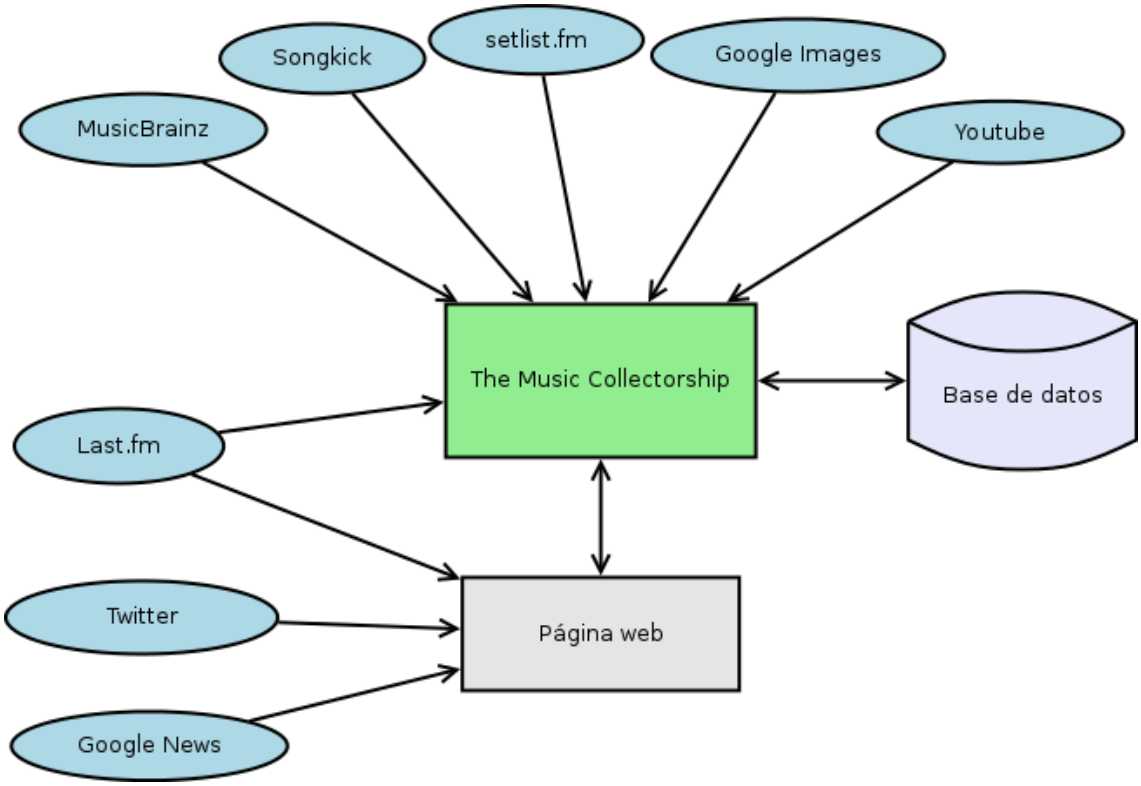


Diagrama 1: Relación de la aplicación con las APIs externas

- **Last.fm:** Esta gran comunidad es el servicio principal en el cual se basa la aplicación a la hora de recoger la información detallada del artista así como sus álbumes y canciones. Se accede a él a través de peticiones *http* que devuelven un objeto *JSON* (Javascript Object Notation) el cual es tratado desde la aplicación.

El uso de su API, en especial en su modalidad de uso no comercial, tiene muchas limitaciones, así que su uso se ha complementado utilizando otros servicios como *MusicBrainz*. Alguna de estas limitaciones han sido imposibles de solventar, como el hecho de no poder hacer más de una petición al segundo, ya que se necesita acceso a una API menos restrictiva que la de uso no comercial.

Se ha utilizado esta API tanto en el servidor, para recolectar la información necesaria, como en el cliente, donde se ha necesitado para realizar comprobaciones en directo de artistas mientras el usuario escribe el nombre del artista a consultar.

- **MusicBrainz:** Creada por la fundación sin ánimo de lucro *MetaBrainz*, esta es una gran base de datos de contenidos musicales que se construye a base de las aportaciones de usuarios anónimos. Esto hace que la base de datos tenga una capacidad muy grande y disponga de tanta música que entidades como la BBC hagan uso de sus servicios.

El uso de su API puede realizarse a través de peticiones *http* como a través de librerías diseñadas para agilizar su uso. En este caso se optó por hacer uso de la librería, ya que había una diseñada para *Python*.

MusicBrainz es empleado para la búsqueda de álbumes de un artista concreto, ya que Last.fm no permite acceder a todos los álbumes de un artista dado.

- **SongKick:** Este servicio web lanzado en 2007 dispone del calendario de conciertos de la mayoría de artistas del momento. Su API permite

acceder a los conciertos próximos de un artista concreto y disponer de información detallada sobre el evento.

A la API se accede a través de peticiones *http*, que devuelve en formato *JSON* la información solicitada.

- **Setlist.fm:** Es un servicio web que contiene una gran cantidad de listas de canciones interpretadas en conciertos pasados por artistas musicales de todo tipo. Es un concepto innovador que complementaba muy bien a la muestra de conciertos. Su API también se accede a través de peticiones *http* y permite acceder a todas las *setlists* de un artista concreto, siendo devueltas con un objeto *JSON*.

  Este servicio, al igual que *MusicBrainz* se nutre de las aportaciones de sus usuarios, así que tratar la información obtenida a veces resulta complejo.

- **Youtube:** Para disponer de los videos colgados en *youtube.com* relacionados con las canciones de cada álbum se ha hecho uso de su API utilizando una librería diseñada para *Python*. En este caso se hace una búsqueda de la canción y el artista en concreto y se guarda la información relevante de los hasta 3 videos más relevantes que devuelva.

- **Google Images:** Para complementar la información del artista se decidió añadir algunas imágenes obtenidas a través de Google imágenes. Esto se consigue a través de una librería no oficial que accede a la API de Google y se guardan las *urls* de las imágenes más relevantes resultantes de la búsqueda formada por el nombre del artista y “artist”.

En la parte de cliente también se ha hecho uso de algunas APIs para complementar la información. Por un lado se ha utilizado la API de *Last.fm* realizando una petición *cross-domain* en *AJAX* desde *Javascript* cada vez que se escribe una letra, de esta manera los resultados de los posibles artistas encontrados se muestran a medida que se escribe la búsqueda.

Por otro lado, en la página con la información del artista resultante se ha añadido información en tiempo real que no debe de ser guardada en la base de datos, así que se hace la petición desde *Javascript*, la información en concreto es:

- **Google News**: Se hace uso de una librería en *Javascript* que permite obtener la información de las noticias de *Google News* relacionadas a una búsqueda en concreto.

- **Twitter**: A través de una petición Ajax se accede a la API d *Twitter* donde se obtienen y se muestran los *tweets* que contengan el nombre del artista seleccionado.

  Al ser peticiones asíncronas no añaden más tiempo de espera para cargar la página, cosa que dados los problemas con los artistas que no están en la base de datos, facilita el acceso.

En la parte de *front-end* también hay librerías adicionales para facilitar el trabajo desde *Javascript*, todo y que se ha procurado trabajar siempre con propiedades nativas de *Javascript* y no tener dependencias de librerías externas.

Es el caso de *Jquery*, que se ha utilizado sobretodo para los selectores de elementos *HTML*, y para poder trabajar con otra librería , también de *Jquery*, para mostrar un dialogo emergente cuando se han de visualizar los vídeos.

2 - Estudio de requerimientos

2.1 - Análisis de requisitos y funcionalidades

Para el correcto funcionamiento de la aplicación se tuvo que analizar una serie de requisitos que debía cumplir, y unas funcionalidades que la aplicación debía aportar. Los requisitos y las funcionalidades se fueron construyendo a medida que avanzaba el *backlog* del proyecto.

Uno de los requisitos fundamentales era que el sistema encontrase el artista introducido por el usuario. Con lo cual debía apoyarse sobre una colección de artistas lo más completa posible, como se hizo con *Last.fm*.

También había que entender que el usuario podría cometer algún error al introducir el nombre. Con el sistema de búsquedas que se implementó se trató de evitar todo tipo de problemas realizando la búsqueda a medida que se escribe, así que el usuario solo podrá seleccionar un artista existente, evitando tener que comprobar la existencia del artista desde servidor.

Otro de los requisitos que se marcó para la aplicación fue que estuviesen disponibles todas las publicaciones del artista seleccionado. Este fue uno de los mayores retos debido a las limitaciones de la API de *Last.fm*.

2.2 - Requisitos funcionales

Una vez analizados los requisitos generales de la aplicación se pueden definir los requisitos funcionales, que se basan en las funcionalidades que el servicio aporta al usuario final.

Las funcionalidades de la aplicación van todas relacionadas con poder ofrecer más y mejor información. Así que las funcionalidades que han sido diseñadas finalmente serían las siguientes:

- **"Live search"**: Mostrar resultados a medida que se realiza la búsqueda.
- **Página de resultados personalizada**: La página con la información del artista ha de tener el aspecto de un *microsite* del propio artista.
- **Información organizada**: Dado que hay mucha información por mostrar, esta debe estar cuanto más organizada mejor.
- **Vídeo por canción**: Una de las funcionalidades que le debe dar un valor añadido a la página es el hecho que cada canción disponga de uno o varios vídeos para reproducir, ya sea con el videoclip, solo la canción o alguna versión en concierto de dicha canción.
- **"Responsive design"**: Se debe intentar que la web sea adaptable a diferentes resoluciones.

2.2.1 - Casos de uso de usuario

La aplicación solo interactúa con un tipo de usuario, que es el usuario final que utilice la página. En este diagrama se muestran todos los posibles casos de uso que puede haber entre el usuario y el sistema, a continuación serán detallados uno por uno.

Diagrama 2: Casos de uso del usuario final de la aplicación

2.2.1.1 - Seleccionar artista

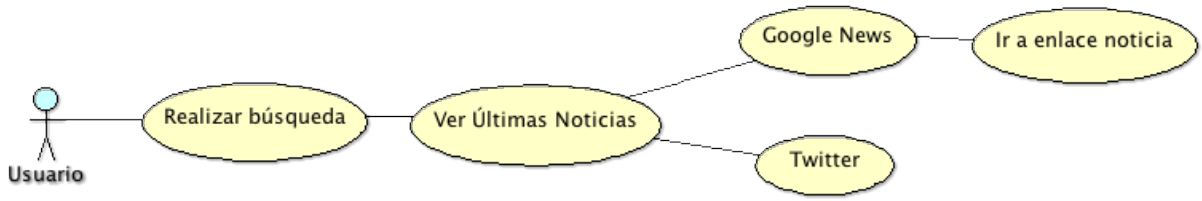
Caso de uso: Seleccionar un artista.

En este caso de uso inicial, el usuario deberá escribir en el buscador el nombre o parte del nombre del artista. A medida que se vaya escribiendo el nombre se realizará la búsqueda con los artistas con un nombre similar al escrito y se mostrarán en una cuadrícula con su nombre y una imagen.

Los artistas mostrados en la cuadrícula serán artistas que estando o no en ese momento en la base de datos del sistema podrán ser buscados y habrá información disponible sobre ellos.

Una vez el usuario ha encontrado el artista deseado, haya escrito su nombre completo o no, deberá clicar sobre el elemento que corresponda al artista deseado. Es entonces cuando se envía la petición al servidor y empieza el trabajo para recolectar su información.

2.2.1.2 - Últimas noticias



Caso de uso: Visualizar las últimas noticias.

Una vez realizada la búsqueda se muestran diversos apartados con información sobre el artista seleccionado. La primera de ellas, que es la única que no obtiene información del servidor, es “Last News”.

Este apartado, que se titula “Last news and updates” tiene un título clicable que colapsa o despliega el contenido del mismo.

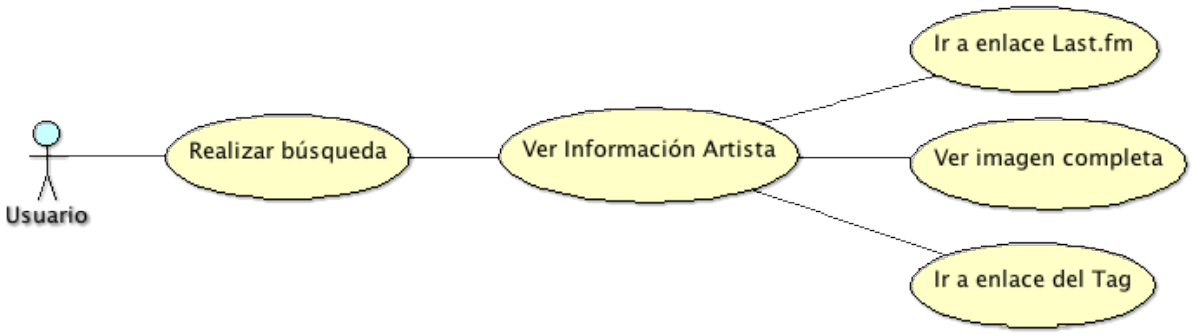
A este apartado se puede acceder a través del enlace de la barra de navegación o navegando por la página. En “Last News” se muestran dos elementos con información de última hora sobre el artista.

El primer elemento son noticias extraídas de *Google News*, se muestra una lista con los titulares y un abstracto de la noticia, la cual nombra el nombre del artista. El título es clicable y redirige a la *url* de la noticia original.

El otro elemento con información relevante muestra los últimos *tweets* publicados que mencionen el nombre del artista. Aparte del mensaje se puede ver el usuario con su foto y la fecha y hora de la publicación.

Si el nombre del artista es un nombre común puede darse el caso que haya noticias o *tweets* que no estén relacionados con el artista, ya que puede que dichas palabras se hayan mencionado pero no para referirse al artista musical, son un ejemplo los grupos con nombres de ciudades o países.

2.2.1.3 - Información del artista



Caso de uso: Visualizar información del artista.

Otro de los apartados que el usuario puede visualizar desde la página de resultados es la información del artista, en ella se muestra diversa información extraída de *Last.fm* y *Google Images* y el usuario puede interactuar con algunos de los elementos mostrados.

El apartado se titula con el nombre del artista más la palabra “Info”. Dicho título, que es clicable al igual que el apartado anterior, también colapsa o despliega el contenido del apartado.

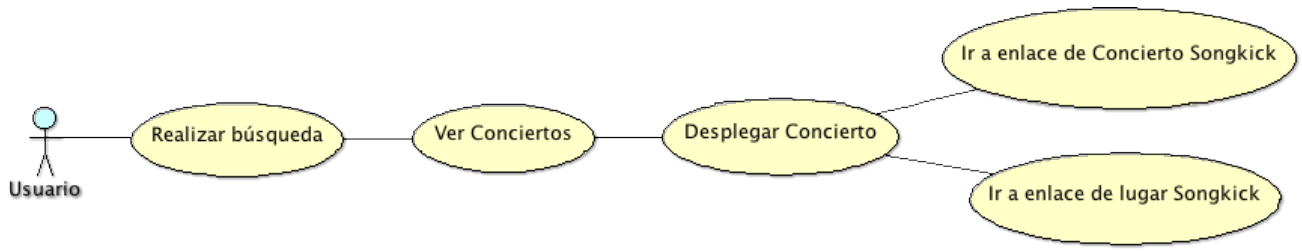
Debido a que gran parte del contenido de este apartado se obtiene de *Last.fm*, hay un enlace que redirige directamente a la página del artista de la conocida comunidad musical.

También se muestra una serie de fotos, una extraída de *Last.fm* y las otras obtenidas a través del buscador de *Google*. Estos elementos, que modifican su comportamiento al pasar el ratón por encima, actúan como un enlace que redirige a la ruta absoluta que contiene la imagen, permitiéndola visualizar en su tamaño original.

Otro de los elementos con los que puede interactuar el usuario son las etiquetas que definen el estilo musical del artista. Estos enlaces redirigen a la página reservada para dicha etiqueta de *Last.fm*.

Por último, se muestra la biografía del artista, en ella puede haber menciones a discográficas o artistas. Estas menciones son enlaces que señalan a páginas de *Last.fm* también.

2.2.1.4 - Próximos conciertos



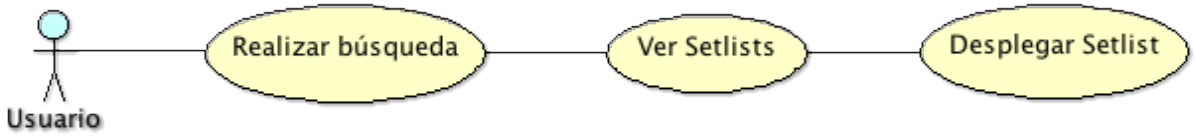
Caso de uso: Visualizar información de los próximos conciertos.

Una vez realizada la búsqueda, y en el caso de que el artista tenga conciertos pendientes, se muestran en el apartado que lleva por título “Next concerts”, dicho título es clicable y permite ocultar o mostrar el contenido del apartado. En el caso de que no haya información de futuros conciertos este apartado no aparecerá y el botón de la barra de navegación estará desactivado.

En este apartado, donde aparece una lista ordenada por fecha de los siguientes conciertos, muestra para cada concierto la fecha y hora y el título del concierto, obtenido del servicio web *Songkick*.

Dichos títulos son clicables, y muestran o ocultan información adicional del concierto. De esta información especifica se puede interactuar con el enlace a la página del concierto en *songkick.com* y con el enlace a la página del lugar donde se celebrará el concierto, también en *songkick.com*.

2.2.1.5 - Visualizar Setlists



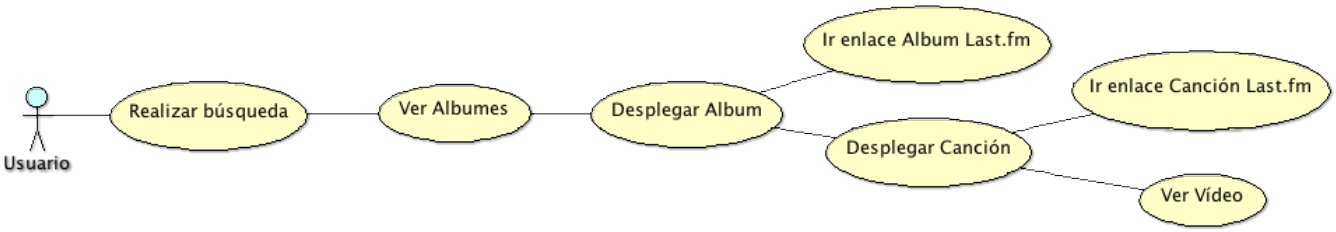
Caso de uso: Visualizar las canciones interpretadas en conciertos anteriores.

Las *setlists* del artista seleccionado se muestran en el caso de que se encuentre al menos una de ellas a través de la API de *setlists.fm*, en caso contrario esta sección no será mostrada y su enlace en la barra de navegación aparecerá deshabilitado.

Este apartado, el título del cual se titula con el nombre del artista más la palabra “Setlists” y tiene el mismo comportamiento que las secciones detalladas anteriormente, muestra una lista con las *setlists* disponibles ordenadas por su fecha de utilización.

Cada elemento de esta lista es clicable y despliega o esconde información detallada de dicha *setlist*. Entre esta información, aparte de la dirección y la lista de canciones utilizadas por el artista en ese evento aparece el enlace que redirige a la página de la *setlist* de la página *setlist.fm*.

2.3.1.6 - Visualizar Álbumes



Caso de uso: Visualizar la información de los álbumes del artista.

El último de los apartados de la página corresponde al dedicado a los álbumes del artista. Dicha sección, titulada con el nombre del artista más la palabra “Albums” aparece siempre que se hayan encontrado álbumes a través de la API de *Last.fm*.

Se muestran en una cuadrícula todos los álbumes encontrados como elementos con la portada y el nombre del álbum. Al clicar en alguno de estos elementos se despliega información detallada del álbum, entre las cuales hay un enlace a la página correspondiente a dicho álbum en *Last.fm*.

Dentro de esta información más detallada sobre el álbum aparece la lista de canciones, las cuales también son clicables y despliegan información detallada sobre la canción. Una de los datos mostrados es un enlace a la página de *Last.fm* dedicada a la canción desplegada.

Por último puede haber la posibilidad que el sistema haya encontrado videos en *Youtube* relacionados con la canción desplegada. En ese caso aparece un enlace con el nombre del video el cual al ser clicado lanza un dialogo con un reproductor de *Youtube* incrustado con el video en cuestión.

2.3 - Requisitos no funcionales

Los requisitos no funcionales de la aplicación están especificados por tal de poder evitar errores, ya que se trabaja con datos de otros servicios, y para poder cumplir con las funcionalidades propuestas. Los requisitos que se han marcado para que la aplicación funcionase correctamente serian los siguientes:

- Evitar la duplicidad de artistas en la base de datos.
- Cadena de dependencias en la base de datos para que todos dependan de la tabla de artistas.
- Desacoplamiento total entre clases en la aplicación para que no haya dependencias a la hora de mostrar los datos.
- Solo contar con los datos externos que tengan el formato deseado.
- Minimizar la carga de la página de resultados.

3 Arquitectura del sistema

3.1 - Base de datos

La base de datos contiene toda la información que la aplicación necesita para un artista, a excepción de aquella que se obtiene a través de *Javascript* (*Last News*).

El contenido de la base de datos se rellena automáticamente a través de peticiones a los servicios externos anteriormente mencionados. Es por ello que los campos y los tipos de datos almacenados en la base de datos tienen que coincidir con las informaciones obtenidas de las APIs utilizadas.

Dado que la aplicación iba a tener que interactuar mucho con la base de datos, se decidió desde el primer momento en utilizar alguna herramienta que facilitara su uso implementando un mapeo objeto-relacional (ORM).

3.1.1 - ORM

El “Object Relational Mapping” es una técnica de programación que permite un nivel de abstracción sobre la base de datos creando una base de datos virtual orientada a objetos.

Esta técnica surge a raíz del problema de similitud entre la programación orientada a objetos, donde la gestión de datos se implementa a través de objetos que tienen atributos los cuales pueden ser dinámicos y los modelos de bases de datos donde se almacenan valores escalares organizados en tablas normalizadas.

Para conseguir este nivel de abstracción se buscó una herramienta que permitiese hacer dicho mapeo en *Python* y así ahorrarse tener que hacerlo de manera manual.

Finalmente se decidió utilizar *SQLAlchemy*. Esta herramienta de código abierto, lanzada en 2006 y que sigue en continuo desarrollo, es una de las más utilizadas, junto a Django, por los desarrolladores en *Python*.

La decisión de usar *SQLAlchemy* en vez de *Django* fue sencillamente debido a que se necesitaba una herramienta que permitiese hacer el ORM y fuese lo más ligera posible. Puesto que *Django* dispone de muchas otras funcionalidades que no iban a ser utilizadas, como el hecho de ser usado de *framework* web, que ya era implementado con otra herramienta llamada *Flask*, se escogió esta opción.

SQLAlchemy se lanza al arrancar la aplicación, inicialmente trata de crear las tablas si no están definidas, y a continuación mapea cada campo de dichas tablas con las clases y sus atributos definidos en la aplicación. Una vez está todo mapeado, cuando se necesita guardar alguna información en la base de datos se hace una llamada a la librería de SQLAlchemy para que guarde dicho objeto en la base de datos virtual e internamente ya se encarga de guardar cada atributo en su campo correspondiente con el tipo de dato adecuado.

3.1.2 - Definición de la base de datos

A medida que se fue definiendo la aplicación se fue definiendo la base de datos, la cual se amplió a medida que se ampliaron los servicios utilizados para proporcionar información sobre los artistas.

La relación que se establece entre la aplicación y la base de datos es de guardar la información obtenida a través de las APIs o de obtener dicha información desde la base de datos. Es por ello que se estableció que cada clase de la aplicación correspondería a una tabla en la base de datos.

La base de la aplicación es el objeto artista, elemento sobre el cual se realiza la búsqueda y que dispone de un identificador único del cual dependen todos los demás objetos (álbumes, conciertos, *setlists*).

Para que la base de datos fuese congruente con la gestión de objetos de la aplicación se estableció que en la base de datos hubiese una cadena de dependencias desde artista y así asumir que toda esta información perteneciese a un mismo ente abstracto, el artista, aunque estuviese distribuida en diferentes tablas.

3.1.3 - Modelo Entidad-Relación

A continuación se va a exponer un diagrama con el modelo entidad-relación de la base de datos, donde se pueden observar las dependencias especificadas previamente, en forma de entidades débiles, debido su dependencia en forma de claves foráneas.

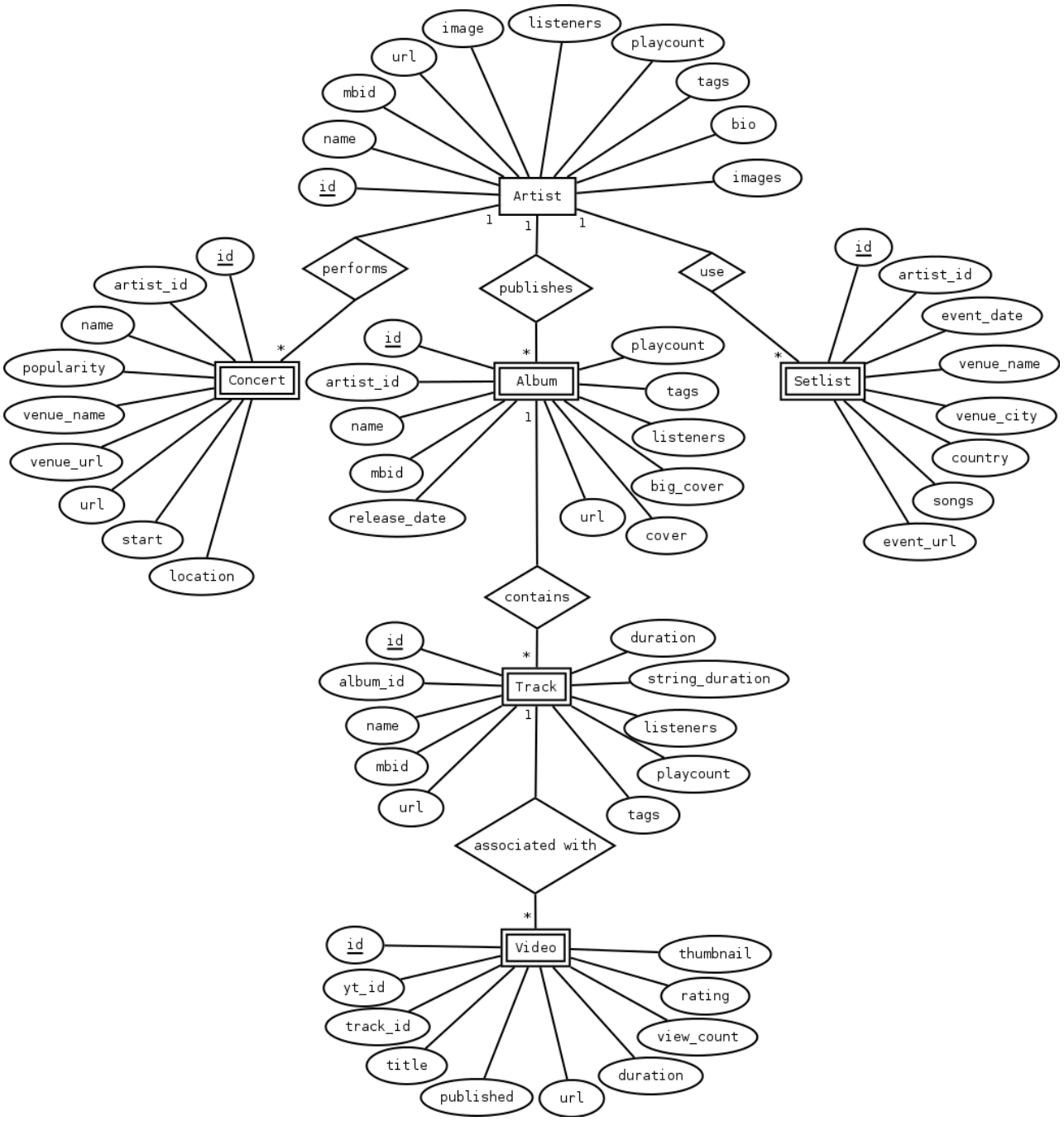


Diagrama 3: Modelo Entidad-Relación de la base de datos

3.2 - Diagrama de clases

Una vez el usuario ha elegido un artista válido en la página del buscador es cuando se interactúa con el servidor. Este, desarrollado en *Python*, consiste en un *script* el cual se encarga de recolectar la información disponible del dicho artista.

Si ha sido buscado previamente, solo tendrá que recoger su información de la base de datos y enviarla al cliente en formato *JSON*. En el caso de que todavía no se haya buscado antes, antes de recoger los datos de la base de datos, se recolectará toda su información a través de las APIs con las cuales trabaja, y es en este momento en el cual se hace uso de todos los elementos de la aplicación.

Siguiendo la estructura de Modelo-Vista-Controlador, el *front-end* de la aplicación actuaria como la vista de esta, el *script* que lanza el servidor actúa como controlador, ya que gestiona las peticiones, actualiza la información de los modelos y formatea la información de dichos modelos a datos reconocibles por la vista. Los modelos son las clases definidas, que equivalen posteriormente a tablas en la base de datos.

La arquitectura de la aplicación quedaría de la siguiente manera:

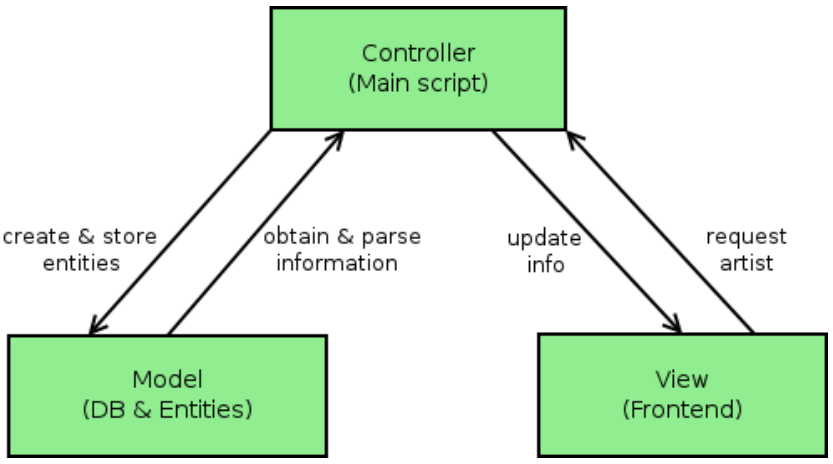


Diagrama 4: Modelo-Vista-Controlador de la aplicación

Dada esta implementación de la arquitectura, no hay más clases que las necesarias para implementar los modelos de la aplicación, que son las entidades necesarias para contener toda la información del artista.

Estas clases son gestionadas directamente por el controlador, con lo que no tienen una dependencia directa entre ellas en el código, aunque si que hay dependencias teóricas (no puede haber canciones si no hay álbumes).

Es por ello que se ha definido el diagrama de clases aplicando estas dependencias. Se puede observar que el diagrama, mostrado a continuación, donde se mantienen las mismas relaciones que el modelo ER de la base de datos, ya que los atributos de cada clase equivalen a campos en las tablas definidas.

También se puede observar que estas entidades no tienen ninguna funcionalidad, solo disponen de un constructor para definir cada tipo de objeto. Toda la lógica queda en el lado del controlador.

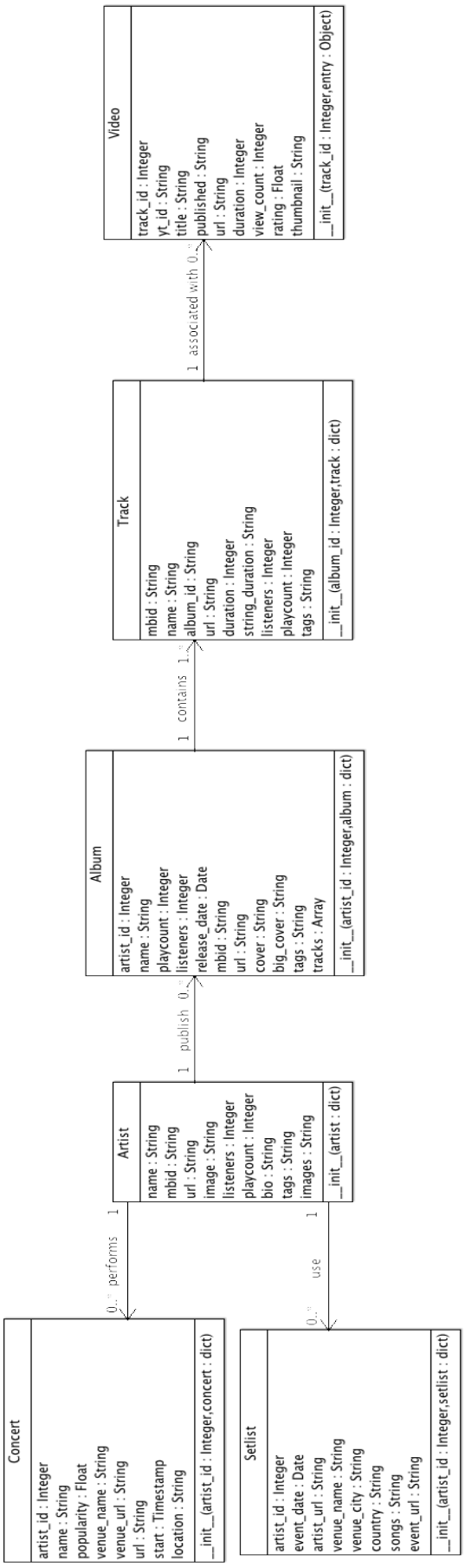
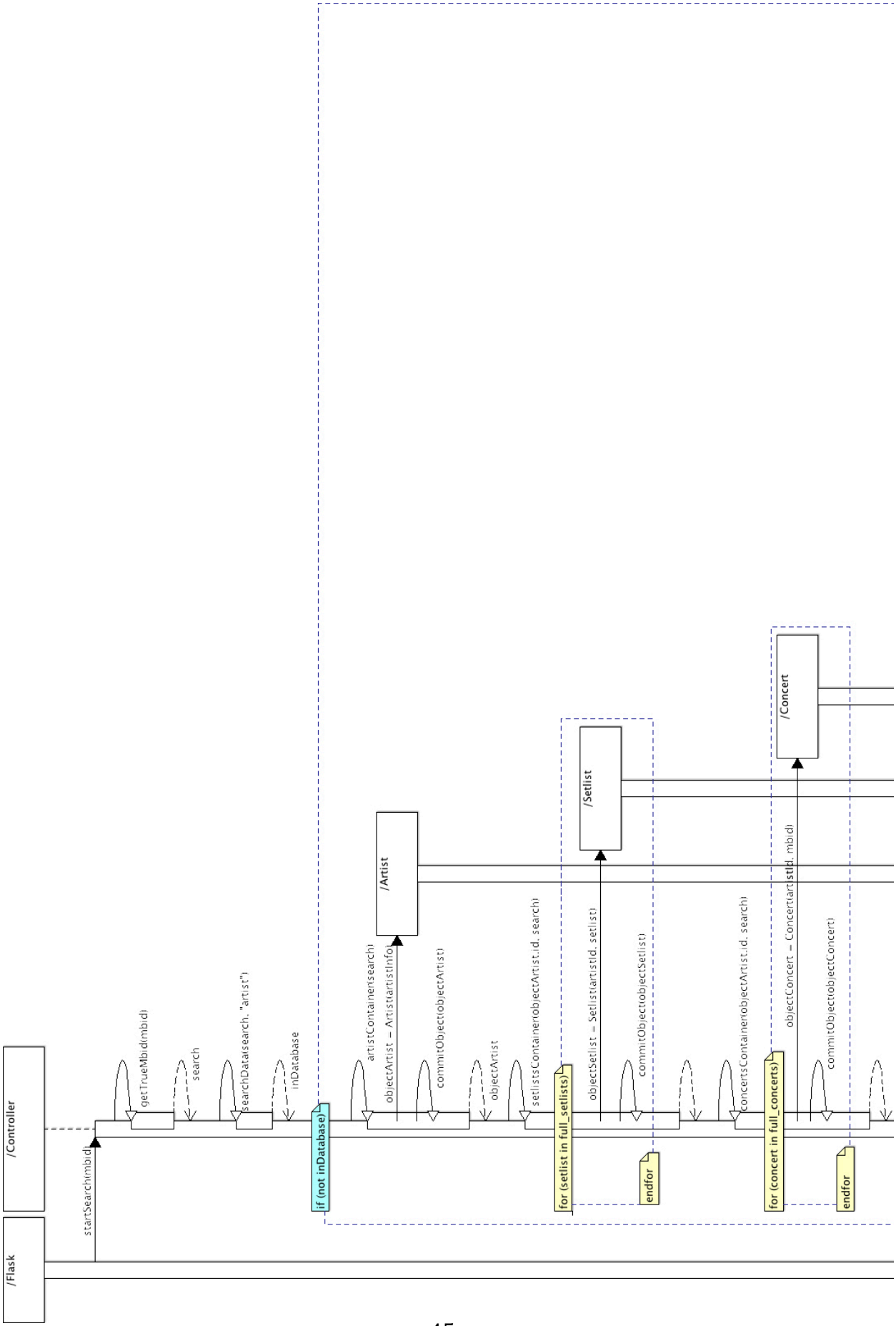


Diagrama 5: Diagrama de clases de la aplicación

3.3 - Diagrama de secuencia



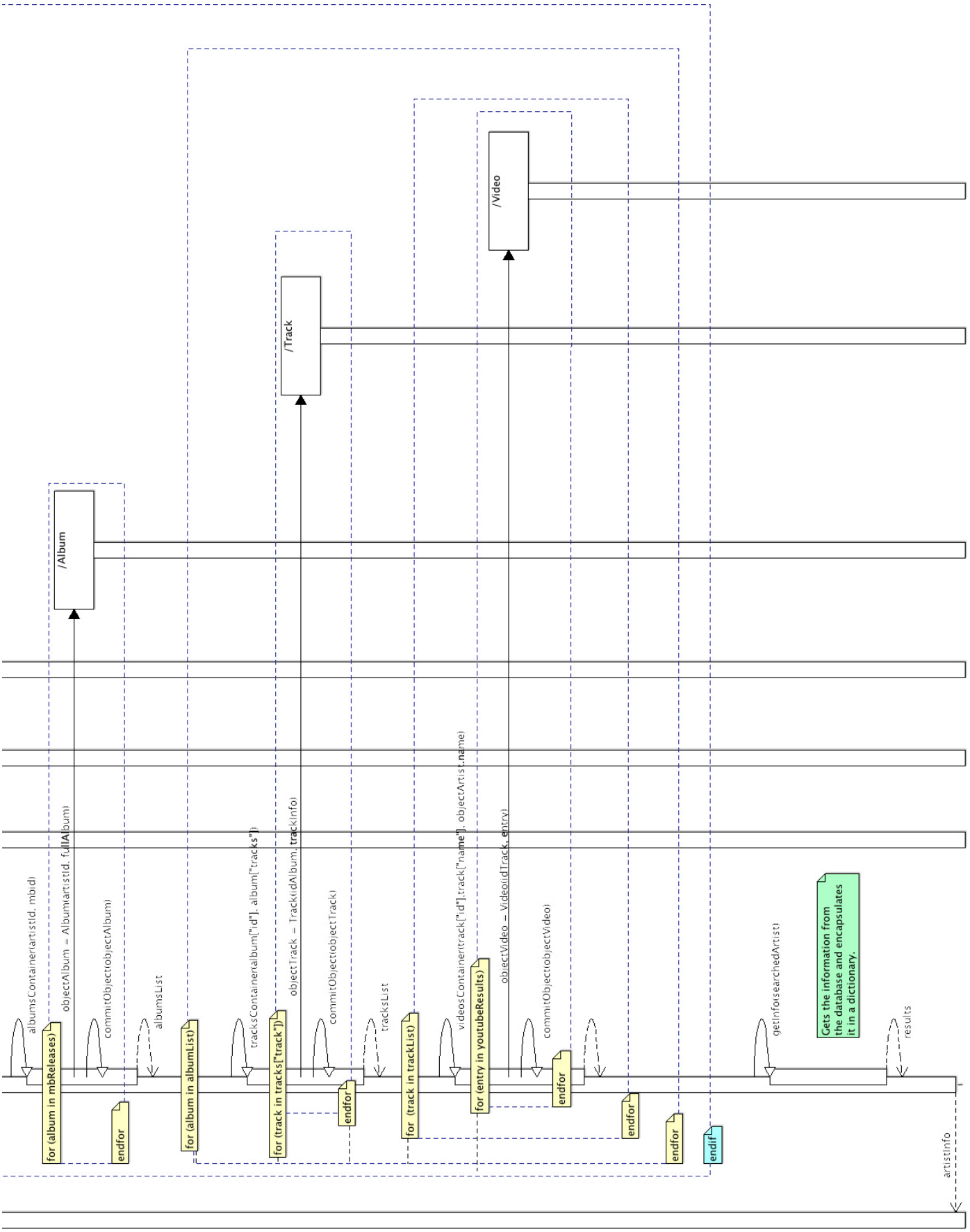


Diagrama 6: Diagrama de secuencia de la búsqueda de artista

Se han dividido los diagramas en dos imágenes aunque ambas pertenecen al mismo hilo de ejecución. En este diagrama se define la tarea principal del servidor, desde que le llega una petición hasta que devuelve la información deseada.

Como podemos observar en el servidor se recibe un identificador del grupo (MBID) que es verificado con el correspondiente en el servicio *MusicBrainz*. Esta operación se realiza porque hay casos en los que no coincide y se necesita el auténtico (definido por *MusicBrainz*) para obtener los álbumes del artista.

Tras ello, intenta buscarlo en la base de datos para recoger la información del artista. En caso que haya encontrado el artista, se dirige directamente a la función llamada *getInfo(searchedArtist)* donde a partir del artista obtiene el resto de información necesaria y la introduce en un diccionario el cual será enviado al cliente. En caso de que no esté ya almacenado en la base de datos debe obtener la información del artista a través de las APIs y almacenarlo para poder luego mandar su información al cliente.

Este proceso sigue una estructura similar para cada tipo de información. Primeramente se recoge la información desde la API correspondiente, normalmente son más de un elemento los recogidos (álbumes, conciertos). Para cada elemento del tipo que corresponda se crea un nuevo objeto utilizando la información obtenida desde la API, una vez creado se guarda en la base de datos con la función *commitObject(obj)*.

Cuando ya se han guardado todos los elementos de un tipo se pasa al siguiente. Una vez finalizado, y utilizando el objeto de tipo *Artista* que hemos creado, se llega al punto que se ha mencionado en el caso de que ya estuviese almacenada la información del artista y se puede recopilar toda su información para ser enviada al cliente.

3.4 - Interfaz de usuario

En la definición inicial del proyecto no se contempló la posibilidad de añadir una interfaz de usuario, ya que el objetivo no era otro que hacer un motor de búsqueda que recogiese información relacionada con artistas musicales a través de diferentes APIs.

A medida que se desarrolló la aplicación, se planteó la necesidad de mostrar los resultados para que pudiesen ser comprobados. Dado que muchos de estos datos eran imágenes o enlaces a páginas web se propuso mostrar dichos datos en una página web.

Una vez se decidió añadir una interfaz para mostrar los datos hubo que redefinir los objetivos añadiendo todo lo referente a la parte de cliente y a su comunicación con el servidor.

Se decidió componer la interfaz en dos páginas, una que fuese un buscador y otra que mostrase toda la información relacionada con el artista seleccionado. Para la comunicación entre cliente y servidor se optó por utilizar peticiones tipo *GET* que gestionaría *Flask*, el *framework* que gestiona toda la parte de cliente.

La arquitectura del *front-end* ha intentado ser fiel separando tres capas en su desarrollo y tratando de no mezclarlas siempre que fuese posible, como muestra el siguiente gráfico.

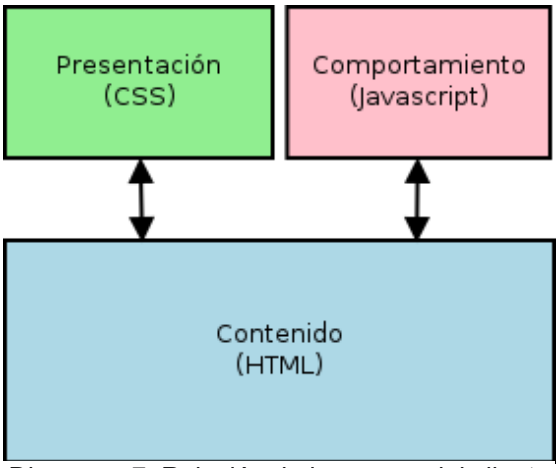


Diagrama 7: Relación de las capas del cliente

3.4.1 - Flask

A la hora de implementar el servicio web se necesitaba trabajar con un *framework* que gestionara la comunicación del cliente con el servidor. Dado que el servidor esta desarrollado en *Python* se plantearon dos posibilidades, *Django* o *Flask*.

Se escogió *Flask* en vez de *Django* por su simplicidad de uso, por el enrutamiento de páginas y por como funciona su sistema de *templates* (*Jinja*). Las funcionalidades que permite implementar son más que suficientes para el uso que se le iba a dar en esta aplicación.

Otra característica que ha facilitado el desarrollo del servicio ha sido la posibilidad de relanzar el servicio automáticamente cada vez que se producían cambios en algún fichero del proyecto, por lo que durante las pruebas el servicio permanecía siempre encendido sin necesidad de cerrarlo y lanzarlo manualmente.

3.4.2 - HTML5

Los *templates*, que utilizan el motor de *Jinja*, están escritos en *HTML5*. De este lenguaje de marcado, que incluye muchas mejoras respecto a su versión anterior, se ha hecho uso de parte de la gran variedad de etiquetas que ofrece, por tal de hacer que la página sea más accesible, orientando el contenido hacia la web 3.0 (web semántica).

Otra de las cuestiones que se ha tenido en cuenta, por tal de respetar la separación entre capas, es no añadir código *Javascript* directamente en los ficheros *HTML*, lo mismo para los estilos, que se añaden todos desde su correspondiente hoja de estilos.

Dado que es un proyecto académico y se ha tratado de utilizar las últimas tecnologías existentes, se han penalizado los navegadores antiguos no añadiendo “fallbacks” en los casos que se utilicen etiquetas solo reconocibles por navegadores que soporten la última versión de este sistema de marcado.

3.4.3 - CSS3

Puesto que finalmente se decidió hacer una interfaz de usuario y esta está compuesta por solamente dos páginas, se ha intentado que la presentación de estas páginas resulte atractiva para el usuario e incluso añadirle un plus de funcionalidad a través de las propiedades *CSS*.

Siguiendo con el criterio explicado en el apartado anterior, se ha decidido utilizar esta última versión de *CSS* por tal de aplicar el máximo de propiedades que aporta, aunque se hayan penalizado navegadores antiguos. Dado que es un lenguaje aun en desarrollo, algunas de las propiedades han tenido que ser añadidas con prefijos por tal de que puedan ser mostradas en la mayoría de navegadores. Para comprobar su disponibilidad en estos navegadores se ha utilizado como referencia la página www.caniuse.com.

También se ha tenido en cuenta la jerarquía de pesos a la hora de definir los selectores que iban a ser utilizados para añadir los estilos. Dando por hecho que todos los estilos se aplican desde las hojas de estilo, se ha procurado que los selectores tuviesen el menor peso posible, es por ello que si se podía definir un selector con el tipo de etiqueta, se aplicaba de esta manera. En caso contrario, se le añadía una clase al elemento por tal de definirle el estilo.

En algunos casos los elementos tienen identificadores únicos. Estos se han definido por requerimientos de *Javascript*, y se han intentado evitar siempre que fuese necesario. De esta manera se ha conseguido evitar tener selectores muy largos o con un peso muy alto.

De las nuevas funcionalidades, y por tal de mejorar la presentación, se han añadido propiedades sólo disponibles en CSS3, como *border-radius*, *box-shadow*, *text-shadow*, *transition*, *transform*, *media-queries* y alguna más. La mayoría de ellas todavía no están definidas por el W3C (World Wide Web Consortium) como estándar, así que para definir dichas propiedades en la hoja de estilo se han añadido tres veces, las dos primeras utilizando un prefijo de

navegador (*Firefox* y *Chrome*), para poder utilizar dichas variables aunque estén todavía en "Working Draft".

Una de las propiedades que cabe mencionar son las *media-queries*. Esta propiedad, que tiene un formato diferente al resto, permite aplicar unos estilos diferentes en el caso de que se cumpla la condición que marque. En el caso de esta aplicación se han aplicado las *media-queries* para modificar las propiedades de presentación en algunos elementos una vez el ancho de la página sea más pequeño del marcado por esta propiedad.

Tanto en la página inicial como en la de resultados se han aplicado *media-queries* que modifican los estilos a partir de un ancho de página determinado. Esto, junto con la manera que se han definido las propiedades, usando valores relativos en vez de absolutos, ha permitido que la página tenga un diseño sensible a varias resoluciones (*Responsive Design*), tema muy de moda debido al cada vez mayor porcentaje de usuarios que acceden a internet a través de móviles o tabletas.

Por último, se ha hecho uso de fuentes diferentes a las genéricas definidas en *CSS*. Para aplicar el uso de estas fuentes se ha hecho uso de *Google Web Fonts*, que dispone de una larga colección de fuentes de código abierto que pueden incluso ser importadas desde el *HTML*. Esto facilita la carga de la página, ya que el servidor no ha de enviar los ficheros correspondientes a las fuentes.

3.4.4 - Javascript

"The Music Collectorship" se ha apoyado mucho en *Javascript*, ya sea para modificar el comportamiento de los elementos *HTML* o para añadir nuevas funcionalidades a la aplicación, como en el caso del apartado "Last News" o la funcionalidad "Live Search".

Como se ha explicado anteriormente, *Javascript* es un lenguaje frágil que fácilmente puede causar problemas o producir vulnerabilidades. Es por ello que a la hora de realizar el código se han seguido una serie de normas por tal de minimizar dichos problemas y para conseguir una mayor eficiencia al ejecutar el código, ya que corre por parte de cliente.

En primer lugar se ha tenido en cuenta que el código se ejecuta por partes (primero se declaran las variables como "undefined" y posteriormente se les asignan sus valores, en la fase de ejecución) y los problemas de *hoisting* que pueden provocar. Es por ello que se han declarado todas las variables al inicio de cada función, aunque no se inicialicen hasta más adelante. Se ha actuado así en todos los casos a excepción de los índices de bucles, para mantener el código entendible.

Para mantener la coherencia a través de todos los métodos y variables se ha decidido utilizar nombres descriptivos e implementar el estilo "lowerCamelCase" en cada declaración.

Siguiendo con los apartados anteriores, y por tal de separar las diferentes capas del *front-end*, se ha evitado la modificación de estilos desde código *Javascript*. Para ello, las funciones que responden a eventos del DOM (Document Object Model) como *mouseover*, *mousedown* o *click* añaden o eliminan clases en los elementos que activan dichas funciones. En las hojas de estilo se han definido dichas clases que modifican la apariencia de los elementos *HTML*.

Ha habido casos donde se ha tenido que inyectar código *HTML* desde *Javascript*, como a la hora de mostrar los resultados en buscador o para mostrar los elementos de “Last News”, que contienen datos que se obtienen una vez está la página cargada. En estos casos, y siendo conscientes de la carga que supone la operación de añadir elementos al DOM, siempre se ha tratado de añadir un solo bloque de código con toda la información necesaria.

Dado que hay muchas funciones en los ficheros de *Javascript* se ha tratado de encapsular dichas funciones usando el patrón *namespace* (espacio de nombres virtual) por tal de no ensuciar el espacio de nombres global. De esta manera en el espacio global solo hay un objeto llamado “namespace” que contiene tres posibles objetos llamados *main*, *results* y *utilities*, siendo este último común para las dos páginas que hay definidas.

Dentro de cada uno de estos objetos se han definido las funciones correspondientes utilizando el patrón modulo por tal de mantener el código limpio, organizado y separado del resto, aunque sigan estando accesible desde el espacio global.

Para evitar problemas con los nombres de algunas variables claves, y dado que la aplicación puede ampliarse en un futuro añadiendo otros ficheros *Javascript* que pueden utilizar o modificar dichas variables se han encapsulado todos los elementos en funciones anónimas autoejecutables. Utilizando esta técnica, y pasando por parámetro elementos como *window*, *document* o *Jquery* nos aseguramos que dentro de la función anónima estas variables serán las esperadas ya que, por ejemplo, *Jquery* no es la única librería que puede definir “$” en el espacio global.

Jquery ha sido utilizada por la aplicación por tal de simplificar el código, todo y que se ha tratado de utilizar métodos nativos de *Javascript* siempre que se ha podido ya que son más eficientes. El principal uso que se le ha hecho ha sido a la hora de crear selectores para modificar elementos del DOM, ya que el motor que utiliza, “Sizzle”, es muy potente.

Otro de los casos donde se ha hecho uso de *Jquery* es para hacer peticiones *AJAX*. Tanto en la página del buscador como en la página de resultados ha surgido la necesidad de realizar peticiones asíncronas *cross-domain*, por tal de obtener datos de APIs externas sin tener que recargar la página. Los métodos que realizan estas peticiones también se encargan de componer y añadir los elementos necesarios en el DOM.

En el caso de la página de resultados el hecho de obtener y añadir las últimas noticias desde *Javascript* permite que la carga de la página sea más rápida, ya que estos métodos no son llamados hasta que no esta el documento completamente cargado.

Para que el *Javascript* no llame a ninguna función hasta que no esté cargada la página se ha utilizado el método “onload” del elemento del espacio global *window*. Esto ha sido necesario también debido a que el código se genera dinámicamente con el motor de plantillas que proporciona *Flask*, y hasta que no esté todos los elementos del DOM añadidos no se pueden definir su reacción a los eventos.

Se han utilizado librerías externas aparte de *Javascript* para algunas de las funcionalidades de la aplicación. En el caso de *Google News*, se utiliza la librería de *Javascript* que proporciona el propio *Google* y un fichero que hace uso de dicha librería y que realiza la búsqueda y muestra sus resultados.

También se ha hecho uso de *Jquery UI* (Jquery User Interface) para añadir la posibilidad de insertar los videos en la página de resultados. Dado que se intentan añadir varios videos por cada canción de cada álbum estos debían añadirse una vez el usuario los selecciona, ya que cargar todos los visualizadores incrustados al mostrar los resultados colapsaría la página. Esta es la razón por la cual se ha utilizado un *plugin* de *Jquery UI*, que lanza un dialogo con el video incrustado una vez el usuario clica en el video deseado.

4 – Implementación del código

En este apartado se va a detallar como se han implementado algunas de las funcionalidades de la aplicación, poniendo énfasis en aquellos procesos que son claves para el correcto funcionamiento, por tal de mostrar su uso y su interacción con el cliente, la base de datos y las APIs que se han utilizado.

4.1 - Live Search

```
searchArtist: function(event){
  var keyword = $("#inputArtist").val();
  keyword = keyword.replace(" ", "+");
  var apiKey = "e7dc908a4c5b80013ca0be4b64e3f696";
  var oRequest;
  oRequest = $.ajax({type: "POST",
                        url:
"http://ws.audioscrobbler.com/2.0/?method=artist.search&artist=" + keyword + "&api_
       key=" + apiKey + "&format=json",
                        dataType:"jsonp"});
  oRequest.success(function(){
    $("#resultsList").html("<p>Loading...</p>");
  });
  oRequest.done(function(msg){
    try{
      var artistNames = [];
      var sBlock = "";
      var sUrlImage, sMbid, sName;
      if (msg != undefined){
        for( var i = 0; i<msg["results"]["artistmatches"]["artist"].length; i++){
          if ( msg["results"]["artistmatches"]["artist"][i]["mbid"] != "" ){
            sUrlImage =
msg["results"]["artistmatches"]["artist"][i]["image"][2]["#text"];
            sMbid = msg["results"]["artistmatches"]["artist"][i]["mbid"];
            sName = msg["results"]["artistmatches"]["artist"][i]["name"];
            if (sUrlImage == ""){
              sUrlImage =
"http://static.tumblr.com/k9utpfa/tcom8wpif/default_cover_m.jpg";
            }
            sBlock += "<li><figure title = '" + sName + "' class = 'floated' id = '"+
sMbid +"'><img src = '"+sUrlImage+"'><figcaption>"+sName+"</figcaption></li>";
          }
        }
      }
    }
    catch(err){
      $("#resultsList").text("impossible to find artist");
    }
$("#resultsList").html(sBlock);
    $(".floated").mousedown(utils.opacBut);
    $(".floated").mouseup(utils.noOpacBut);
    $(".floated").mouseover(utils.focusIn);
    $(".floated").mouseout(utils.focusOut);
    $(".floated").click(ns.main.selectArtist);

      });
},
```

Método *searchArtist* incluido en el fichero *Javascript* main_script.js.

Esta función *Javascript* se encarga de actualizar la lista de resultados cada vez que se escriba una letra. Como se puede observar realiza una petición *AJAX* a *Last.fm* que devuelve los resultados que coindicen con la búsqueda.

Mientras se espera que estos resultados sean recibidos se muestra un mensaje (“Loading…”) para hacer patente que se está realizando la búsqueda.

Una vez recibidos los datos, un objeto *JSON*, son tratados para componer la cuadrícula que mostrará los artistas que coinciden con dicha petición. Para cada elemento resultante se crea un elemento *HTML* tipo *figure* que contiene la imagen y el nombre del artista. Como identificador se le asigna el *MBID* (MusicBrainz identificator), que se utilizará para realizar la petición al servidor en el caso de que sea seleccionado.

Por último, y una vez se ha compuesto todo el bloque de texto que se debe insertar en el código *HTML*, se inserta y se define los cambios de presentación que van a sufrir en cada tipo de acción.

4.2 – Gestión del cliente

```python
@app.route("/")
def index():
  return render_template('index.html')


@app.route("/search", methods=['GET', 'POST'])
def ask():
  if request.method == 'GET':
   mbid = request.args.get('artist', '')
   content = startSearch(mbid)
   return render_template("search.html",info = content)
```

Definición de las rutas en cliente definidas a través de *Flask* en el fichero *Python* main_controller.py.

En este fragmento de código se define la interacción entre cliente y servidor. Como se puede observar se han declarado dos posibles rutas, la raíz y "/search".

En el caso de la raíz se lanza un fichero *HTML* que contiene toda la información necesaria, ya que no se debe pasar ningún parámetro desde el servidor. Por otro lado, a la ruta "/search" se debe acceder a través de una petición tipo *GET* que contenga el identificador del usuario seleccionado. Si es así, se llama al método *startSearch* pasándole dicho identificador por parámetro.

Esta función devuelve un objeto con toda la información, que es enviado al cliente junto a la plantilla que se encarga de gestionar dichos datos.

4.3 – searchData

```
def searchData(dataId, dataType):
  db.connect()
  metadata = MetaData(db)
  Session = sessionmaker(bind=db)
  session = Session()

  try:
    if (dataType == "artist"):
      ret = session.query(Artist).filter(Artist.mbid == dataId)[0]
    elif (dataType == "albums"):
      ret = session.query(Album).filter(Album.artist_id == dataId)
    elif (dataType == "tracks"):
      ret = session.query(Track).filter(Track.album_id == dataId)
    elif (dataType == "videos"):
      ret = session.query(Video).filter(Video.track_id == dataId)
    elif (dataType == "concerts"):
      ret = session.query(Concert).filter(Concert.artist_id == dataId)
    elif (dataType == "setlists"):
      ret = session.query(Setlist).filter(Setlist.artist_id == dataId)

  except:
    ret = []

  session.close()
  db.dispose()
  return ret
```

Método *searchData* incluido en el fichero *Python* main_controller.py.

Este es el método que se utiliza para buscar información en la base de datos a través de *SQLAlchemy*, sea del tipo que sea. Recibe por parámetro dos variables, *dataId* y *dataType* y devuelve una *array* llamada *ret*.

Cada vez que es llamado se conecta a la base de datos creando una nueva "sesión". Con la conexión establecida comprueba que tipo de información se ha pedido por parámetro con el campo *dataType*. Una vez encontrado el tipo de dato se hace una petición a la base de datos buscando el objeto que contenga un identificador que coincida con *dataId*.

Para evitar un problema de conexión con la base de datos se ha definido este método con un *try/except* que fuerza la devolución de una *array* vacía en el caso de que no se puedan obtener los datos.

Por último, y justo antes de devolver el resultado, se cierra la conexión con el servidor.

4.4 – artistContainer

```
def artistContainer(mbid):
  f = urllib2.urlopen(
"http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&mbid="+mbid+"&api_key="+api_key
+"&format=json")
  artistInfo = json.loads(f.read())
  f.close()
  objectArtist = Artist(artistInfo)

  artistId = commitObject(objectArtist)
  objectArtist.id = artistId;
  return objectArtist
```

Función *artistContainer* incluida en el fichero *Python* main_controller.py.

Esta función es la encargada de buscar la información detallada del artista solicitado. Hay un método de estas características para cada tipo de objeto, aunque cada uno de ellos tiene sus particularidades y se han tenido que declarar por separado.

En este caso se recibe por parámetro el identificador de artista *mbid*. Con esta variable junto con *api_key*, que está declarada como global, se hace una petición *HTTP* al servicio de *Last.fm* pidiendo la información del artista.

Una vez finalizada se parsea el resultado para tener un diccionario y se crea un objeto de tipo *Artista* enviando por parámetro este mismo diccionario.

Este objeto se guarda en la base de datos mandándolo por parámetro a la función *commitObject*. Este método devuelve el identificador que se le asigna en la base de datos, el cual va a ser necesario para crear los objetos de tipo *Album*, *Concert* y *Setlist*.

Una vez ha finalizado esta función devuelve el propio objeto creado, ya que es utilizado más adelante por la aplicación.

4.5 – Ejemplo de objeto JSON

A continuación se va a mostrar el resultado de hacer una petición a *Last.fm* pidiendo la información detallada del artista “Metronomy”.


```
{"artist":
 {
 "name":"Metronomy",
 "mbid":"93eb7110-0bc9-4d3f-816b-4b52ef982ec8",
 "bandmembers":
  {"member":[
   {"name":"Joseph Mount","yearfrom":"1999"},
   {"name":"Gabriel Stebbing","yearto":"2009"},
   {"name":"Oscar Cash"},
   {"name":"Anna Prior"},
   {"name":"Gbenga Adelekan"}
   ]
  },
 "url":"http:\/\/www.last.fm\/music\/Metronomy",
 "image":[
  {"#text":"http:\/\/userserve-ak.last.fm\/serve\/34\/67947576.jpg","size":"small"},
  {"#text":"http:\/\/userserve-ak.last.fm\/serve\/64\/67947576.jpg","size":"medium"},
  {"#text":"http:\/\/userserve-ak.last.fm\/serve\/126\/67947576.jpg","size":"large"},
  {"#text":"http:\/\/userserve-
ak.last.fm\/serve\/252\/67947576.jpg","size":"extralarge"},
  {"#text":"http:\/\/userserve-
ak.last.fm\/serve\/500\/67947576\/Metronomy.jpg","size":"mega"}
  ],
 "streamable":"1",
 "stats":{
  "listeners":"441611",
  "playcount":"14544718"
 },
 "similar":
  {"artist":[{
   "name":"Hot Chip",
   "url":"http:\/\/www.last.fm\/music\/Hot+Chip","image":[
   {"#text":"http:\/\/userserve-ak.last.fm\/serve\/34\/78224120.png","size":"small"},
   {"#text":"http:\/\/userserve-ak.last.fm\/serve\/64\/78224120.png","size":"medium"},
   {"#text":"http:\/\/userserve-ak.last.fm\/serve\/126\/78224120.png","size":"large"},
   {"#text":"http:\/\/userserve-
ak.last.fm\/serve\/252\/78224120.png","size":"extralarge"},
   {"#text":"http:\/\/userserve-
ak.last.fm\/serve\/_\/78224120\/Hot+Chip.png","size":"mega"}]
   },
   {
   "name":"Late of the Pier",
   "url":"http:\/\/www.last.fm\/music\/Late+of+the+Pier",
   "image":[
   {"#text":"http:\/\/userserve-ak.last.fm\/serve\/34\/24585843.jpg","size":"small"},
   {"#text":"http:\/\/userserve-ak.last.fm\/serve\/64\/24585843.jpg","size":"medium"},
   {"#text":"http:\/\/userserve-ak.last.fm\/serve\/126\/24585843.jpg","size":"large"},
   {"#text":"http:\/\/userserve-
ak.last.fm\/serve\/252\/24585843.jpg","size":"extralarge"},
   {"#text":"http:\/\/userserve-
ak.last.fm\/serve\/500\/24585843\/Late+of+the+Pier+81996.jpg","size":"mega"}]
   }]
 },
 "tags":
  {"tag":[
   {"name":"electronic",
   "url":"http:\/\/www.last.fm\/tag\/electronic"},
   {"name":"new rave",
   "url":"http:\/\/www.last.fm\/tag\/new%20rave"},
   {"name":"experimental",
   "url":"http:\/\/www.last.fm\/tag\/experimental"},
```

```
    {"name":"electro",
    "url":"http:\/\/www.last.fm\/tag\/electro"},
    {"name":"glitch","url":"http:\/\/www.last.fm\/tag\/glitch"}
    ]
  },
  "bio":{
   "published":"Mon, 23 Jul 2012 17:12:45 +0000",
   "summary": Metronomy is an ...(biografia del artista)
   "placeformed":"Totnes, Devon, United Kingdom",
   "yearformed":"1999",
   "formationlist":{
    "formation":{
     "yearfrom":"1999","yearto":""}
    }
   }
  }
}
```


Objeto *JSON* con el cual trabaja la aplicación para obtener alguna de la información del artista.

Como se puede observar, es mucha la información que se puede extraer de servicios como *Last.fm*. La aplicación se limita a recoger algunas de estas informaciones para crear el objeto de tipo *Artista* que posteriormente serán mostradas en el servicio web.

Se han elegido los datos más significativos y que tienen en común todo tipo de artistas ya que, por ejemplo, no todos los artistas que sean buscados serán una banda de música.

La ventaja de trabajar con este formato es que permite utilizar los datos como si fuese un diccionario, y por ello se envía el objeto completo al constructor de artista, ya que este simplemente deberá obtener la información que necesita, ignorando el resto.

Como queda evidente viendo la cantidad de información disponible se puede deducir que en un futuro se podría incluir alguna información complementaria además de la ya utilizada, como artistas similares o año de formación del artista.

5 - Planificación

La planificación para este proyecto no se pudo configurar correctamente desde el inicio y tuvo que ser corregida a medida que se corrigieron los objetivos.

Queda patente en el gráfico que durante las primeras tres semanas no se empezó a desarrollar nada de la aplicación ya que todos los esfuerzos fueron a analizar como plantear el proyecto, jugar con algunas APIs que parecían necesarias y empezar a definir como estaría estructurado el proyecto, en el lado de servidor.

A partir de octubre, donde ya se conocía el funcionamiento de algunas APIs y se tenia claro que servicios había que utilizar en esta primera fase de desarrollo, se empezó a programar la aplicación y se creó la base de datos.

Durante el desarrollo de la aplicación del servidor se siguió realizando pruebas con otras APIs que posteriormente serian añadidas o no en la aplicación. Un ejemplo es *MusicBrainz*, que por dificultades con la API de *Last.fm* tuvo que ser incluida a mitad del desarrollo.

Una vez se redefinieron los objetivos para incluir una página web, a finales de octubre, se empezó a definir su estructura, a medida que se seguía desarrollando el servidor.

La comunicación con el servidor fue un punto clave, que fue modificado por tal de mejorar la respuesta y devolver los datos de la mejor manera posible.

Durante las últimas semanas, mientras se acababa el desarrollo del servidor y del cliente (sobretodo *Javascript*), se empezó a testear el sistema para arreglar los problemas que debía tener en cuenta la aplicación una vez estuviese finalizada.

A continuación se muestra el gráfico de Gantt con la planificación definitiva.

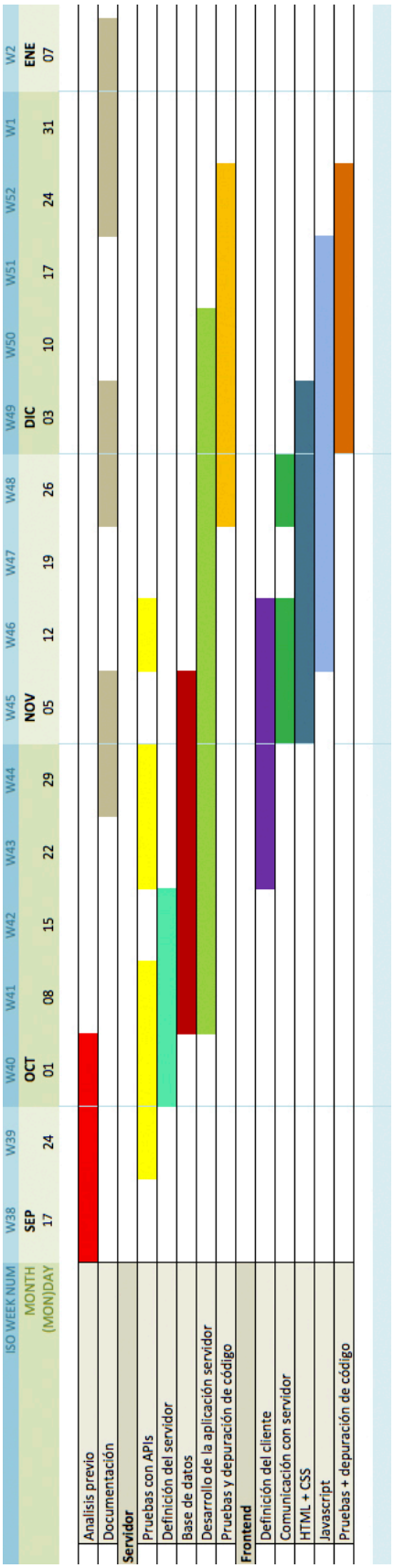


Diagrama 8: Diagrama de Gantt con la planificación del proyecto

6 - Aplicación

6.1 - Interfaz

La interfaz web está dividida en dos páginas, el buscador y la página de resultados. La página inicial no contiene más que un cajón de búsqueda con el que, aunque no tiene ningún botón para buscar, se muestran los resultados a medida que se escribe.

En la imagen de ejemplo se puede comprobar como al escribir la palabra “the”, ya aparecen todos los artistas que pueden ser buscados que contengan dicha palabra. Además aparecen ordenados de más popular a menos.

A medida que se escribe el nombre del artista se va actualizando esta cuadrícula de resultados apareciendo cada vez menos, dado que se restringe la búsqueda.

Una vez se selecciona un artista, se manda la petición al servidor, que depende de si tiene la información del artista o no, tardará más o menos. En el caso de que tenga que hacer la búsqueda en directo, se eliminará la cuadrícula de resultados y se mostrará un mensaje conforme se esta cargando la información del artista.

Se ha intentado que la página fuese lo más funcional posible, aunque no se ha podido poner mucho énfasis en el diseño, debido a que la página web surgió como un añadido al proyecto.

Imagen 1: Ejemplo de búsqueda en la página de inicio.

La página de resultados se ha diseñado para que parezca un *microsite* del artista. Se ha querido juntar toda la información que se dispone del artista en la misma página para incentivar al lector a recorrerla y fijarse en cada uno de los apartados antes de llegar al último, que se asume que es el motivo por el cual el usuario entra en la página.

A pesar de todo, hay un menú al inicio de la página para dirigirse directamente al apartado deseado.

Cada uno de los apartados puede ser escondido o desplegado clicando encima de su título.

El diseño de la página intenta mantener una coherencia con la página del buscador, dando un aspecto minimalista que contrasta con la cantidad de información que se muestra en ella.

A continuación se comentará la experiencia del usuario para cada apartado de la página.

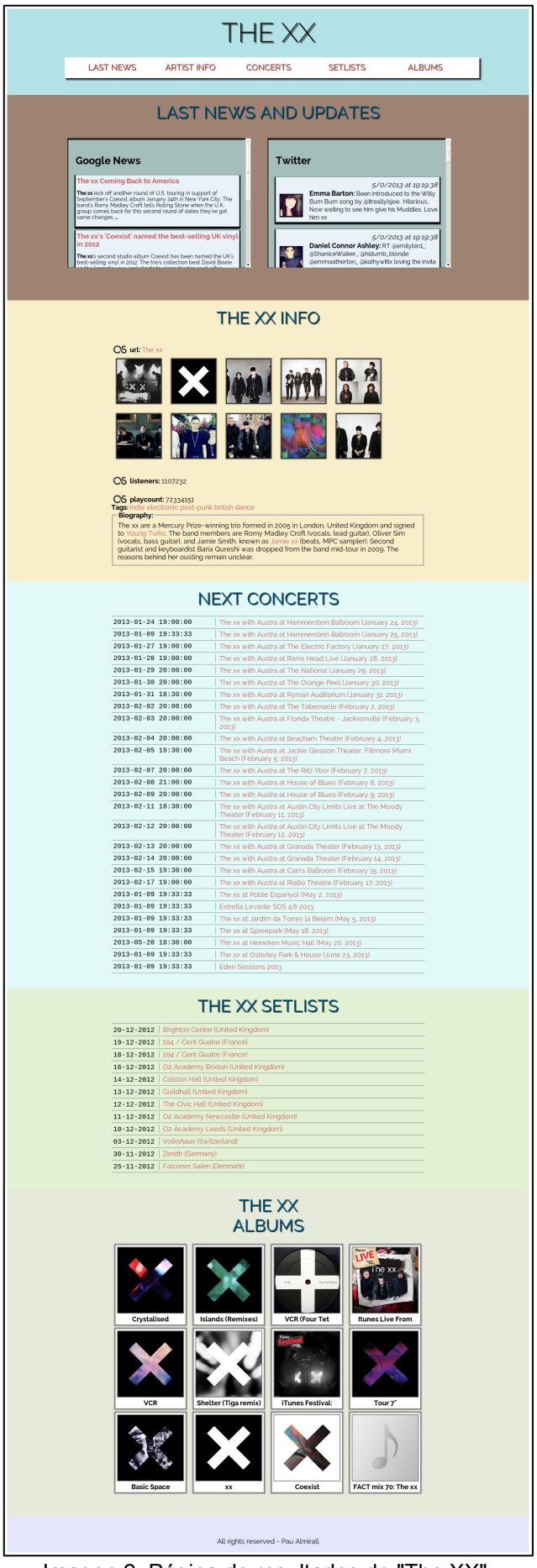


Imagen 2: Página de resultados de "The XX".

- **Last News:**

Imagen 3: Sección de últimas noticias de la página de resultados

Este es el primer apartado que aparece en la página, aunque es el último en cargar, ya que su información se carga desde *Javascript* con una petición *AJAX*. Hay 2 cajones, uno de ellos contiene noticias relacionadas, extraídas de *Google News*, y el otro tiene los últimos *tweets* publicados sobre el artista relacionado.

El apartado de las noticias tiene un titular clicable que lleva a la página originaria de la noticia. También aparece parte del texto de la noticia junto al titular.

El apartado dedicado a *Twitter* muestra todos los *tweets* recientes, con su fecha y hora de publicación y el autor junto a su foto.

Cada uno de estos apartados tiene una barra de desplazamiento para poder contener toda esta información en un espacio reducido.

- **Artist info:**

Imagen 4: Sección con la información detallada del artista de la página de resultados.

Este apartado contiene la información más general del artista, como su biografía, las etiquetas que se le han asignado en *Last.fm*, los usuarios que los siguen, etcétera.

También cuenta con una cuadrícula con imágenes del artista. Estas imágenes, que se aumentan levemente al pasar por encima, llevan a la dirección donde está hospedada la imagen original. Excepto la primera que se extrae de *Last.fm*, todas las imágenes son extraídas de *Google Images* a través de una búsqueda, es por ello que dependiendo del grupo, hay imágenes que pueden no corresponder.

- **Next Concerts:**

Imagen 5: Sección con la información de conciertos de la página de resultados.

En la sección de conciertos se muestran los próximos conciertos del artista seleccionado. En el caso de que el artista ya no esté en activo o no tenga conciertos pendientes, este apartado no aparece en la página.

Los resultados están ordenados del más próximo al más lejano, aunque algunas fechas resultan confusas, debido a problemas con los datos obtenidos a través de la API de *Songkick*.

Cada uno de estos elementos es clicable y expande información adicional sobre el concierto, como el lugar donde se va a celebrar, su popularidad o un enlace a la página de dicho concierto en *songkick.com*.

- **Setlists:**

Imagen 6: Sección con las *setlists* del artista de la página de resultados.

Este apartado muestra una lista ordenada de más reciente a más antiguo con todas las *setlists* que se han recopilado del artista seleccionado.

Cada elemento de la lista despliega información detallada de la *setlist*, como el lugar donde se dio el concierto, el enlace de la *setlist* en *setlist.fm* o la lista de las canciones que fueron interpretadas.

Este apartado también puede no aparecer en el caso de que no se haya recopilado ninguna *setlist* del artista seleccionado.

- **Albums:**

Imagen 7: Apartado de álbumes de la página de resultados

Este es el último apartado y el que muestra la información que puede ser más interesante para el usuario, la discografía del artista.

Esta se presenta, al igual que los resultados de la página anterior, como una cuadrícula donde aparecen las portadas de las publicaciones del artista junto con el nombre de la publicación. Si por algún motivo no se dispone de la caratula de la publicación aparecerá una por defecto como se aprecia en la imagen.

Al clicar en alguno de estos elementos se despliega la información del álbum y se reorganiza la colocación del resto de álbumes.

Dentro de la información detallada del álbum se puede apreciar la portada más al detalle, así como los seguidores que escuchan regularmente dicho álbum a través de *Last.fm*, el enlace a su página en *Last.fm*, la fecha de publicación y la lista de canciones.

Cada canción también dispone de información adicional, como su duración, un enlace a su página en *Last.fm*, los usuarios que escuchan la canción en la mencionada comunidad y vídeos relacionados.

Dependiendo de la canción pueden aparecer más o menos videos según los resultados obtenidos a través de la API de *Youtube*. Si el usuario hace clic sobre el enlace del vídeo, en vez de ir a su página de *Youtube*, aparece un diálogo emergente con un reproductor incrustado.

Imagen 8: Sección de albumes con un álbum seleccionado.

7 - Resultados obtenidos

"The Music Collectorship" , al ser una aplicación que recoge información de manera automática, se ha tenido especial cuidado de los resultados obtenidos y el tiempo que tardan en ser obtenidos.

Los tiempos ha sido un factor determinante durante todo el proceso de creación. Si el artista seleccionado no está almacenado en la base de datos, se debe recoger su información a través de las diferentes APIs. Para todas ellas se hace una sola petición que devuelve toda la información que se necesita para ese tipo de dato.

Para todas excepto para los álbumes y canciones. Para estos dos tipos de datos se debe hacer una petición a *Last.fm* para cada elemento, ya sea álbum o canción. Es por ello que cuantos más álbumes tenga un artista más peticiones se deberán hacer a *Last.fm*. Este problema, sumado al hecho de que la clave utilizada para usar la API es de uso "no comercial" y tiene una limitación de 1 petición por segundo, hacen que la búsqueda en tiempo real no sea tan rápida como se desearía.

Los tiempos que tarda la aplicación si la información esta disponible en la base de datos son de uno o dos segundos.

Por otro lado, la única información que realmente se guarda en la base de datos es texto. Se decidió no guardar ninguna imagen por tal de no llenar de datos el servidor, al menos en una aplicación con este nivel de madurez.

Uno de los temas con los que se ha tenido también mucho cuidado es que la información encontrada coincidiera con la información demandada. Lo más habitual es que toda la información se encuentre correctamente y sin ningún tipo de problema.

En cambio, hay casos donde alguna información no es la correcta. Un ejemplo son los artistas con nombres comunes, donde las noticias, *tweets* e imágenes pueden no corresponder con las del artista.

Otro caso es con los grupos con muchos álbumes. *MusicBrainz* devuelve un número máximo de álbumes y en el caso de los artistas que tienen más álbumes publicados, no se pueden mostrar a través de la aplicación.

8 - Conclusiones y posibles ampliaciones.

8.1 - Conclusiones

Bajo mi punto de vista este proyecto ha sido bastante ambicioso, ya que se ha tratado de conseguir en un periodo limitado de tiempo desarrollar un servicio web desde cero y que dependiera de otros servicios externos, cosa que añade complejidad a la tarea.

Todo lo que ha tenido de ambicioso también lo ha tenido de satisfactorio, ya que he aprendido a utilizar nuevas herramientas como *SQLAlchemy* o *Flask* y también he ampliado mis conocimientos en *Python* y *Javascript*. El hecho de poder visualizar el resultado de los avances que se van cometiendo a medida que se desarrolla la aplicación también ha ayudado y motivado para seguir adelante.

Los problemas derivados de las APIs externas han supuesto numerosos dolores de cabeza y han hecho replantear parte del diseño de la aplicación, incluso en algún punto ha habido problemas que no se han podido solucionar. Esto se debe a la mala documentación de algunas de las APIs o de su pobre surtido de métodos.

Una vez visto el producto final me siento orgulloso del resultado y creo que es un servicio útil y que aporta un valor añadido a los existentes en el mercado, aunque seguramente le falte madurar un poco para poder ser lanzado al mercado. En el siguiente punto se explica en que puntos se podría mejorar o ampliar la aplicación.

8.2 - Posibles ampliaciones

Hay puntos en la aplicación donde se podrían aplicar mejoras o ampliar el servicio. A continuación se pondrán algunos ejemplo de mejoras que me hubiera gustado añadir pero por motivos de tiempo o de otro tipo no fue posible.

- **Uso de más API's:**
  Utilizar más servicios como *Amazon*, *Spotify*, *Grooveshark* o *TuneWiki* ampliaría el valor añadido que ya ofrece la página al usuario, la harían mucho más completa y supondría una menor dependencia de servicios como *Last.fm*.

- **Licencias menos restrictivas:**
  Para realizar el proyecto se han utilizado en todo momento licencias gratuitas y de uso no comercial. Esto ha hecho que la aplicación se viese condicionada a unas limitaciones bastante restrictivas que dificultan su uso.

- **Web más atractiva/funcional:**
  La página web se ha hecho básicamente para poder mostrar los datos de la aplicación y permitir hacer una simulación para comprobar que el servicio funciona, pero tiene algunas carencias que podrían ser resueltas en una futura ampliación.

- **Script recolector:**
  Una de las ampliaciones más necesarias e inmediatas sería un *script* que se dedicara a realizar búsquedas automatizadas en el servidor por tal de rellenar la base de datos. Empezando con los artistas más populares y con más discos y repitiendo las búsquedas cada cierto tiempo para actualizar la información.

- **Permitir añadir información:**

  Dado que hay grupos que no están almacenados en *Last.fm*, no pueden ser buscados tampoco en esta aplicación. Este inconveniente puede ser solventado habilitando la posibilidad de añadir de manera manual la información de dichos artistas a través de los usuarios, mejorando el servicio e incrementando su valor añadido respecto a los otros servicios.

9 - Bibliografía

- http://docs.python.org/
- http://flask.pocoo.org/
- http://docs.sqlalchemy.org/
- http://stackoverflow.com/
- http://caniuse.com/
- http://www.wikipedia.com/
- http://mashable.com/2009/05/19/social-music-history/
- http://css-tricks.com/specifics-on-css-specificity/
- http://coding.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/
- http://www.genbetadev.com/desarrollo-web/7-consejos-para-hacer-buen-responsive-design
- Javascript: The Good Parts, Douglas Crockford. Editorial O'Reilly.