Proyecto VLC

Desarrollado por José María Samos y Sebastià Adrover

Proyecto desarrollado en Centre Integrat de Formació Professional Francesc de Borja Moll

1er de Desenvolupament d' Aplicacions Web modalitat Dual

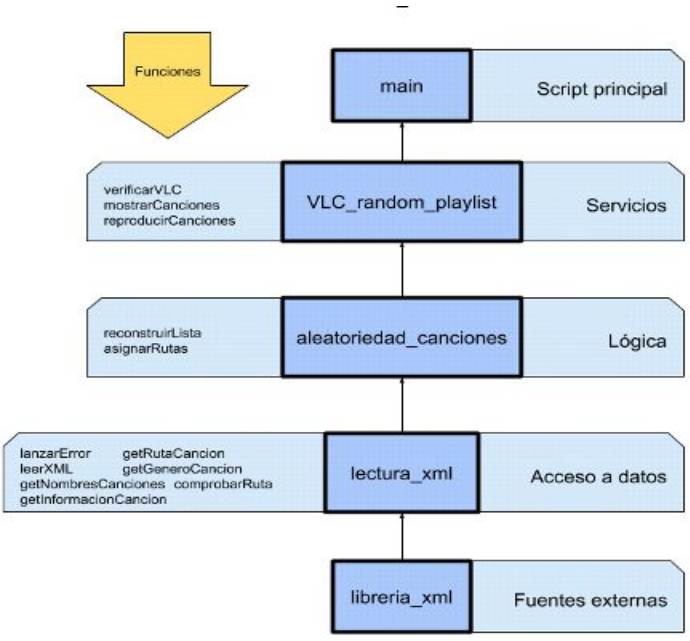
Tabla de contenidos

1. Descripción del proyecto

El programa reproduce una lista aleatoria de canciones que recoge de un archivo xml llamado libreria_canciones.xml y las ejecuta en el programa VLC y muestra por consola las canciones y su información en el orden correspondiente.
Por cada ejecución del programa, el orden de reproducción es diferente debido al algoritmo que reordena aleatoriamente las canciones haciendo uso de una librería externa ofrecida por python llamada random.
El programa consta de cuatro módulos más el fichero xml.
En caso de no tener VLC instalado saltará un mensaje de error y terminará la ejecución del programa, si no encuentra el archivo xml o no esta bien parseado también saltará el mensaje de error. Lo mismo pasará si una ruta de alguna canción en el xml está mal escrita o el programa no la encuentra.

2. Arquitectura de la aplicación

La aplicación se puede dividir en cinco partes como se muestra en la siguiente ilustración:



El fichero main.py es el script principal que se debe ejecutar para iniciar la aplicación. Actúa de nexo entre los distintos módulos de la aplicación. Consta de distintas llamadas a las funciones de los demás ficheros **VLC_random_playlist**, **aleatoriedad_canciones** y **lectura_xml**. También consta de un input para evitar que el programa cierre instantáneamente antes de poder ver las canciones que se muestran por pantalla. Por otra parte tenemos el fichero **libreria_canciones** que consta de toda la información respecto a las canciones.

En el fichero **VLC_random_playlist.py**, que representa la capa de servicios, contiene tres funciones:

- verificarVLC
- mostrarCanciones
- reproducirCanciones

El módulo **aleatoriedad_canciones** representa la capa lógica y está compuesto de las siguientes dos funciones:

- reconstruirLista
- asignarRutas

El módulo de **lectura_xml**, es el encargado de acceso a datos y está formado por siete funciones, haciéndolo así el módulo con más funcionalidades:

- lanzarError
- leerXML
- getNombresCanciones
- getInformaciónCanción
- getRutaCanción
- getGeneroCanción
- comprobarRuta

El fichero xml **librería_canciones** contiene la información de las canciones, está estructurado de la siguiente manera:

- raíz: librería
- albums
- album
- nombre
- fecha
- autor
- tracks
- track
- nombre
- duracion
- ruta
- género
- géneros
- género
- nombre

3. VLC_random_playlist

3.1 verificarVLC

Esta función se encarga de comprobar que existe el programa VLC en tu ordenador. En el caso de que no, lanza un mensaje de error y cierra el proyecto automáticamente. Ya que el paquete de software de VLC, dependiendo de la configuración de tu sistema operativo, puede instalar el ejecutable de VLC en la carpeta 'Program files' o en 'Program files (x86)'.

3.2 mostrarCanciones

Esta función se encarga de mostrar por pantalla el orden en el que se van a reproducir las canciones y la información sobre estas que podría interesar al usuario accediendo a la información que devuelve la función **getInfromaciónCanción** ejecutada en el módulo de acceso a datos.

3.3 reproducirCanciones

Esta es la función que usamos para reproducir las canciones. Para ello ejecutamos los comandos de nuestro sistema operativo Windows para que este llame al programa VLC. Para ello importamos el paquete de Python 'os' y usando sus respectivas funciones.

4. aleatoriedad_canciones

4.1 reconstruirLista

Esta función se encarga de reordenar la lista de los títulos de las canciones del fichero XML de una forma completamente aleatoria. Para ello usamos la función 'random' que devuelve valores numéricos aleatorios que usamos para posicionar las los títulos de nuevo en otra lista.

4.2 asignarRutas

Esta es la función que asigna las rutas a las canciones de la lista mencionada en la función **recontruirLista**. En caso de que la ruta no sea exista, es decir, que no se pueda acceder a ella, borra el título de la lista y ya no se muestra por pantalla.

5. lectura_xml

5.1 lanzarError

Esta función lanza por pantalla un mensaje de error en caso de que algo no esté correcto para poder ejecutar el programa. Para ello importamos el paquete ‘tkinter’ que lanza mensajes para que tu Sistema Operativo te los muestre.

5.2 leerXML

Este es la función que parsea el fichero XML y hace accesible toda su información para un interprete Python. Usamos el paquete ‘xml.etree.ElementTree’. En caso de que el fichero XML no esté bien formado salta un mensaje por pantalla y cierra la ejecución del programa automáticamente.

5.3 getNombresCanciones

Esta es la función recorre todo el fichero XML para obtener los títulos de las canciones y meterlos en una lista que usaremos en otras funciones para realizar las múltiples aplicaciones que ofrece nuestro proyecto.

5.4 getInformaciónCanción

Esta es la función recorre todo el fichero XML para agrupar toda la información que pueda interesar al usuario sobre las canción que va a escuchar y la mete mete en distintos diccionarios bien estructurados.

5.5 getRutaCanción

Al igual que todas las funciones de lectura, recorre todo el fichero para tomar la ruta de la canción, a través de la lista de títulos que ofrece la función **reconstruirLista**, dentro del ordenador para que el programa VLC pueda reproducirla.

5.6 getGeneroCanción

Esta función asigna un género a las canciones a partir de la estructura de Entidad Relación con el que tenemos organizado el fichero XML.

5.7 comprobarRuta

Esta función comprueba que la que las rutas de las canciones que va a reproducir el proyecto existan en tu ordenador. En caso contrario, se borran de la lista de reproducción y no se muestran por pantalla.

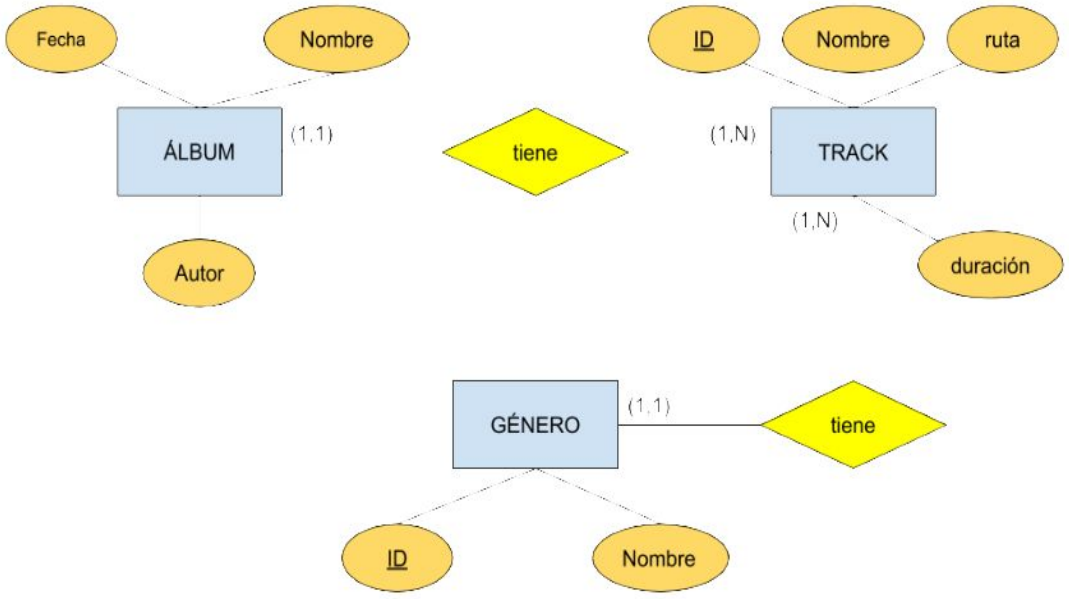
6. Estructura librería_canciones

Es el fichero que contiene toda la información sobre las canciones que se van a reproducir.

6.1 Modelo Relacional

- álbum(nombre, fecha, autor, id_track*)
- track(id, nombre, duración, ruta, género_id*)
- género(id, nombre)

6.2 Modelo Entidad Relación



7. Casos test

Hemos realizado minuciosos casos test durante el desarrollo del proyecto para evitar funcionalidades erróneas. Hemos utilizado insertado barricadas que impiden la entrada de datos corruptos a nuestro proyecto. Y además hemos asegurado los datos cada vez que estos se manipulaban durante la lógica.

7.1 test_aleatoriedad_canciones

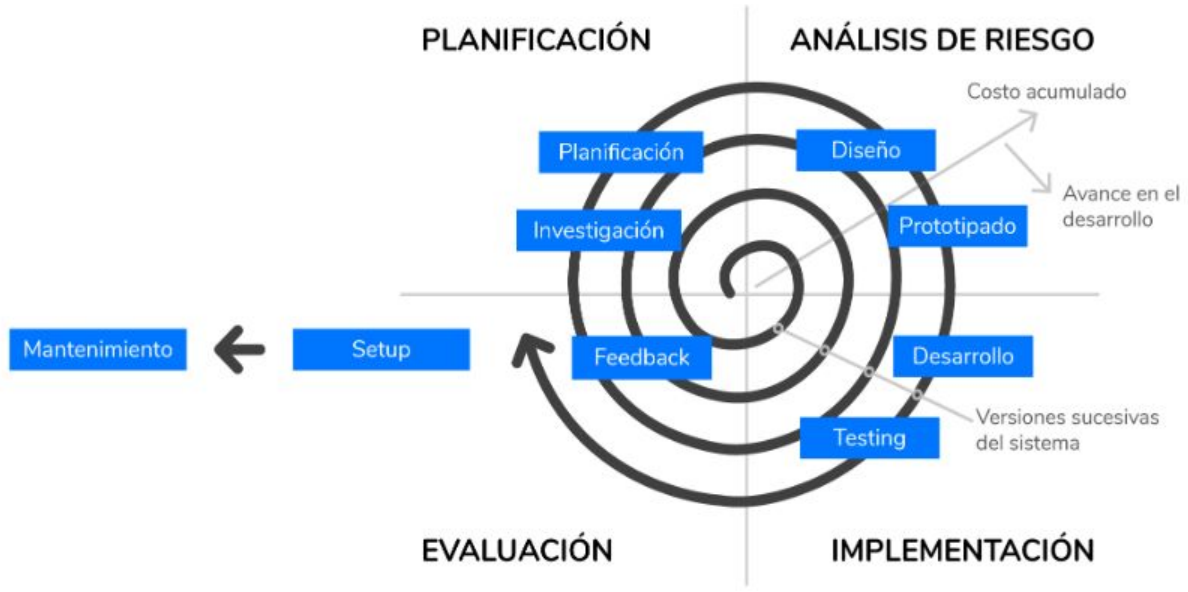
Comprueba que una vez reorganizada la lista de canciones de forma aleatoria, siga habiendo el mismo número de canciones sin estar reorganizadas. Además comprueba también que en la nueva lista no hay ningún título repetido.

7.2 test_lectura_xml

Comprueba que el número de títulos leídos en el fichero XML sea el mismo que aguarda el mismo fichero, es decir, que no se pierda ningún título durante la lectura. Comprueba que las rutas de los títulos sean accesibles. También comprueba que los títulos leídos en fichero XML no sean repetidos. Además comprueba que la lectura del fichero XML se realiza correctamente. También hace una comprobación sobre la validez de las rutas, sobre la validez de la información dada por pantalla y el género asignado a las canciones.

8. Metodología de desarrollo

El modelo en espiral es una combinación entre el modelo lineal o de cascada y el modelo iterativo o basado en prototipos que habíamos mencionado anteriormente. Se utiliza con éxito en proyectos donde el coste de un fallo es un gran riesgo, de ahí que su principal aportación sea considerar la gestión de esos riesgos, algo que en los modelos anteriores ni siquiera se menciona.
En cuanto a su ejecución, el modelo en espiral consiste en seguir ciclos crecientes de cuatro fases cada uno, que se van realizando siguiendo una forma de espiral. En cada ciclo se pasa por dichas fases bien definidas, como en el modelo de cascada, pero con capacidad de evolucionar su complejidad con cada ciclo. Por tanto, se trata de un modelo evolutivo que, conforme avancen los ciclos, aumentará el tiempo de ejecución, así como el volumen de código fuente desarrollado y la complejidad de la gestión de riesgos y de la planificación.



El desarrollo en espiral consiste en cuatro fases:

1. Planificación: Se determinan los objetivos y hasta donde vamos a desarrollar durante la fase.
2. Análisis de peligro: Se analiza todo aquello que pueda afectar a todo lo plantead
3. Implementación: Se desarrolla y valida el software según el alcance acordado.
4. Evaluación: Se realiza una meticulosa evaluación de todo lo desarrolado anteriormente es correcto.

Este ha sido el planteamiento que hemos llevado a cabo para cada una de las funciones desarrolladas en nuestro proyecto. Primero la planteábamos junto a todos sus posibles riesgos, la desarrollábamos evitando todos los riesgos captados, y luego hacíamos un meticuloso estudio para comprobar que todo estuviese correcto. En caso de que no, realizábamos otra vuelta de espiral con esa misma función; si todo estaba correcto, empezábamos de nuevo el ciclo con otra función.

9. Análisis y justificación del tiempo invertido

El desarrollo del código se ha realizado en dos semanas. Al usar una metodología en espiral hemos ido desarrollado distintos aspectos de los distintos módulos de forma alterna. A medida que acabábamos una funcionalidad planteábamos otra y así iba tomando forma el proyecto.

9.1 Primera semana de desarrollo

Esta gráfica nos indica el tiempo invertido durante los días de trabajo desde un aspecto general. Como se puede apreciar la actividad laboral es prácticamente exponencial, a

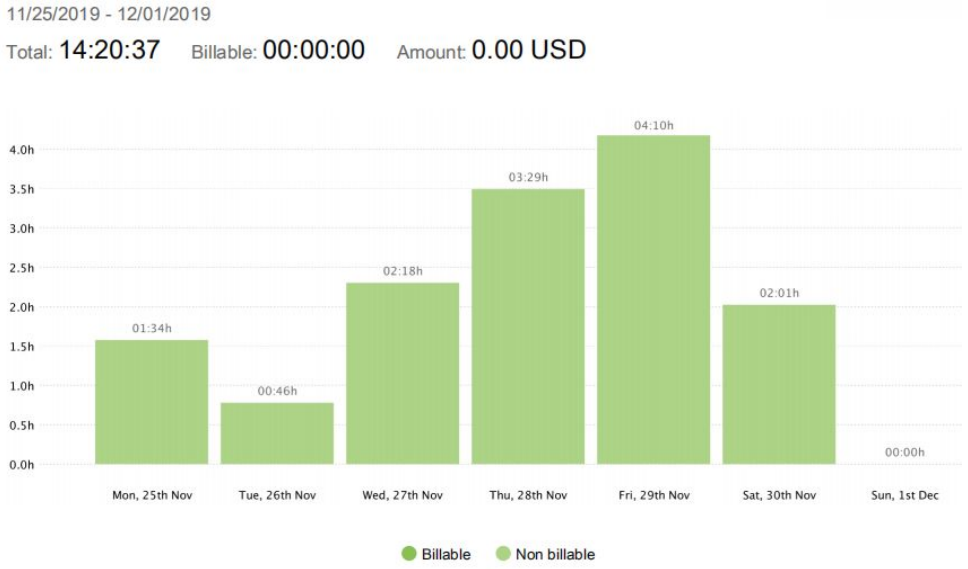
medida que desarrollábamos plateábamos más aspectos para desarrollar y más incrementaba el trabajo por hacer.

A continuación tenemos las actividades realizadas durante la primera semana y el tiempo concreto invertido en dichas actividades.

Project/Time Entry	Duration	Amount
Proyecto VLC	**04:19:29**	**0.00 USD**
Plantear función para usar VLC	00:21:42	0.00 USD
Crear función: getNombresCanciones	00:22:25	0.00 USD
getRutaCancion	00:37:50	0.00 USD
Crear XML	00:28:19	0.00 USD
Depuración	00:40:01	0.00 USD
Crear método reconstruirLista()	00:23:01	0.00 USD
Modificar función: getNombresCanciones	00:21:40	0.00 USD
Crear verificador de VLC i empezar método para reproducir canción	01:04:31	0.00 USD

9.2 Segunda semana de desarrollo

Al igual que la semana anterior, esta gráfica es exponencial y mantenemos la misma metodología de trabajo.



Al tener el proyecto prácticamente planteado en su totalidad el número de actividades a realizar aumenta notablemente como vemos en la siguiente tabla.

Proyecto VLC	**14:20:37**	**0.00 USD**
Documentación	02:01:22	0.00 USD
Programación por parejas	00:54:59	0.00 USD
añadir todas las llamadas al main	03:24:11	0.00 USD
(No description)	00:31:41	0.00 USD
getInformacionCancion	01:41:12	0.00 USD
solucionar bug getNombresCanciones	00:10:00	0.00 USD
arreglar fallos	00:36:40	0.00 USD
Conseguir que VLC reproduzca una lista	01:20:01	0.00 USD
Añadir readme	00:46:18	0.00 USD
Conseguir que VLC reproduzca canción	01:05:58	0.00 USD
Añadir casos test	00:53:22	0.00 USD
mensajes de error	00:05:01	0.00 USD
Rellenar XML	00:28:29	0.00 USD
testing	00:21:23	0.00 USD

10. Conclusiones

10.1 Dificultades encontradas

A la hora de desarrollar el proyecto no hemos encontrado ninguna dificultad que nos haya impedido el desarrollo del proyecto en ningún momento.
En cuanto al módulo de 'acceso a datos' lo más difícil fue aprender a recorrer el fichero XML con los comandos de Python. Para desarrollar las funciones de **lectura_xml**, estas debían acceder a la etiqueta correcta.
En cuanto a la lógica, la dificultad encontrada fue desarrollar la función que reconstruía la lista de canciones aleatoriamente sin que se repitieran los títulos.
Y en cuanto a la capa de servicios, la dificultad encontrada fue que el Sistema Operativo interpretara correctamente los comandos que le lanzaba nuestro proyecto.

10.2 Mejoras aplicables

Consideramos que hemos alcanzado todos los aspectos realizables para este proyecto. Hemos realizado distintos estudios para solucionar cualquier bug que pudiéramos apreciar. El código podría estar más depurado y legible para otros posibles programadores que desarrollaran la aplicación. También se podrían añadir más comentarios que facilitaran su comprensión.
Los nombres asignados a las funciones y variables son bastantes entendibles para cualquier región hispano parlante aunque podríamos haber establecido los nombres en inglés y así abarcar un contexto más internacional.