

**Proyecto VLC**

CFGS DUAL DESARROLLO DE APLICACIONES WEB – CIFP Francesc Borja Moll

Daniel Arqués Toro y Daniel Jonathan Zurita Mena

֎ Índice General

[ **Índice de Ilustraciones** 2](#_Toc26018162)

[**1.** **Introducción** 3](#_Toc26018163)

[**2.** **Entornos de Desarrollo** 4](#_Toc26018164)

[**2.1.** **Comandos Git** 4](#_Toc26018165)

[**2.2.** **Gestión del proyecto** 5](#_Toc26018166)

[**2.3.** **Clockify** 5](#_Toc26018167)

[**2.5.** **Conventional Commits** 6](#_Toc26018173)

[**3.** **Programación** 7](#_Toc26018174)

[**3.1.** **Descripción técnica** 7](#_Toc26018175)

[**3.1.1.** **Arquitectura de aplicación** 7](#_Toc26018176)

[**3.1.2.** **Tecnologías utilizadas** 7](#_Toc26018177)

[**3.1.2.1.** **TDD** 7](#_Toc26018178)

[**3.1.2.2.** **Sistema SOLID** 7](#_Toc26018179)

[**3.1.2.3.** **Barricadas** 7](#_Toc26018180)

[**3.1.2.4.** **Parte lógica** 7](#_Toc26018181)

[**3.1.3.** **Diagrama de Componentes** 8](#_Toc26018189)

[**3.2.** **Metodología de Desarrollo** 9](#_Toc26018190)

[**3.2.1.** **Prototipada en espiral** 9](#_Toc26018191)

[**4.** **Lenguaje de Marcas** 10](#_Toc26018192)

[**4.1.** **XML** 10](#_Toc26018193)

[**4.2.** **XSD** 10](#_Toc26018194)

[**4.3.** **Estructura en árbol del xml** 11](#_Toc26018195)

[**5.** **Base de Datos** 12](#_Toc26018196)

[**6.** **Sistemas Informáticos** 13](#_Toc26018197)

[**7.** **Conclusiones** 14](#_Toc26018198)

[**7.1.** **Cosas guays** 14](#_Toc26018199)

[**7.2.** **Posibles mejoras** 14](#_Toc26018200)

[**7.3.** **Dificultades** 14](#_Toc26018201)

[**8.** **Webgrafía** 15](#_Toc26018202)

# **Índice de Ilustraciones**

[Ilustración 1 - Introducción: Git Hub 3](#_Toc26018244)

[Ilustración 2 - Introducción: Repositorios y branch 3](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018245)

[Ilustración 3 - Introducción: Presentación Word 3](#_Toc26018246)

[Ilustración 4 - Crear un repositorio en Git Hub - Comandos Git Bash 4](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018247)

[Ilustración 5 - Conventional Commits 6](#_Toc26018248)

[Ilustración 6 - Estructura commit 6](#_Toc26018249)

[Ilustración 7 - Tipos de commits 6](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018250)

[Ilustración 8- TDD 7](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018251)

[Ilustración 9 - SOLID 7](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018252)

[Ilustración 10 - Barricadas 7](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018253)

[Ilustración 11 - Lógica 7](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018254)

[Ilustración 12 - XML validator 10](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018255)

[Ilustración 13 - XML 10](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018256)

[Ilustración 14 - XSD 10](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018257)

[Ilustración 15 - XML Schema Generator 10](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018258)

[Ilustración 16 - Árbol XML 11](file:///C:\Users\Thelpher\Desktop\DAW%201r\VLC%20Projecte\Documentación_ProyectoVLC.docx#_Toc26018259)

[Ilustración 17 - Exportar BBDD a XML 12](#_Toc26018260)

[Ilustración 18 - Playlist .bat 13](#_Toc26018261)

[Ilustración 19 - Lógica 14](#_Toc26018262)

# **Introducción**

El objetivo de este proyecto consiste en realizar un programa que permita reproducir canciones aleatorias con el programa *VLC.* Es un proyecto transversal de todos los módulos de 1º DAW Dual. Se realizará un control de versiones (basado en *Commitizen)* mediante línea de comandos *Git Bash* y en entorno gráfico *Git Hub*.

Ilustración 1 - Introducción: Git Hub

Se realiza en parejas, por lo tanto, ambos estaremos trabajando en los mismos archivos, y se encargará de ir documentando las partes del proyecto que realice.

La manera de la que trabajaremos simultáneamente en los mismos archivos será mediante *Git Hub*. El repositorio constará de dos *branches*, la principal, llamada *master* donde se encuentran los archivos ya finalizados, y otra llamada *preprod* donde se hacen los *commit* a medida que se va avanzando en el proyecto. Una vez acabadas todas las partes del proyecto, se hace un *merge* de las dos *branch* para que quede el proyecto listo en la rama master.

La documentación del mismo tiene que estar presente con los puntos que hay en el índice de este documento.

Ilustración 2 - Introducción: Repositorios y branch

No es necesario, pero el documento tiene que estar bien presentado, con una tabla de contenido, uso de estilos, uso de cabeceras y pies de página, el texto justificado y otras opciones que dan los editores de texto.

Ilustración 3 - Introducción: Presentación Word

La fecha de entrega del trabajo será día **2 de Diciembre,** donde también tendremos que defender el proyecto.

El final del documento consta de una Webgrafía hypervinculada a las direcciones que se han tomado como referencia y/ o consulta para este trabajo.

# **Entornos de Desarrollo**

En este proyecto ponemos en práctica lo anteriormente aprendido sobre control de versiones, compartir repositorios, crear *branches* y hacer un *merge* entre las mismas.

## **Comandos Git**

Estos comandos son los mas utilizados en git:

* ***Git init*** : Iniciar un repositorio en local
* ***Git add .*** : Añade todos los archivos a estar *tracked* o añadidos los cambios para un próximo commit.
* ***Git commit -m “ “:*** Hace una captura de las instancias y archivos que previamente se ha hecho un  ***add*** y que estan trackeadas. La opción m nos permite añadir un mensaje al commit en el que describimos el commit. Estos últimos dias hemos aprendido que exite un estandar de cómo se crear los comentarios.
* ***Git push -u origin master:*** Hacemos una subida del último commit al repositorio remoto donde master es la rama en la que hacemos el push. También hemos aprendido a manejar diferente ramas.



Ilustración 4 - Crear un repositorio en Git Hub - Comandos Git Bash

## **Gestión del proyecto**

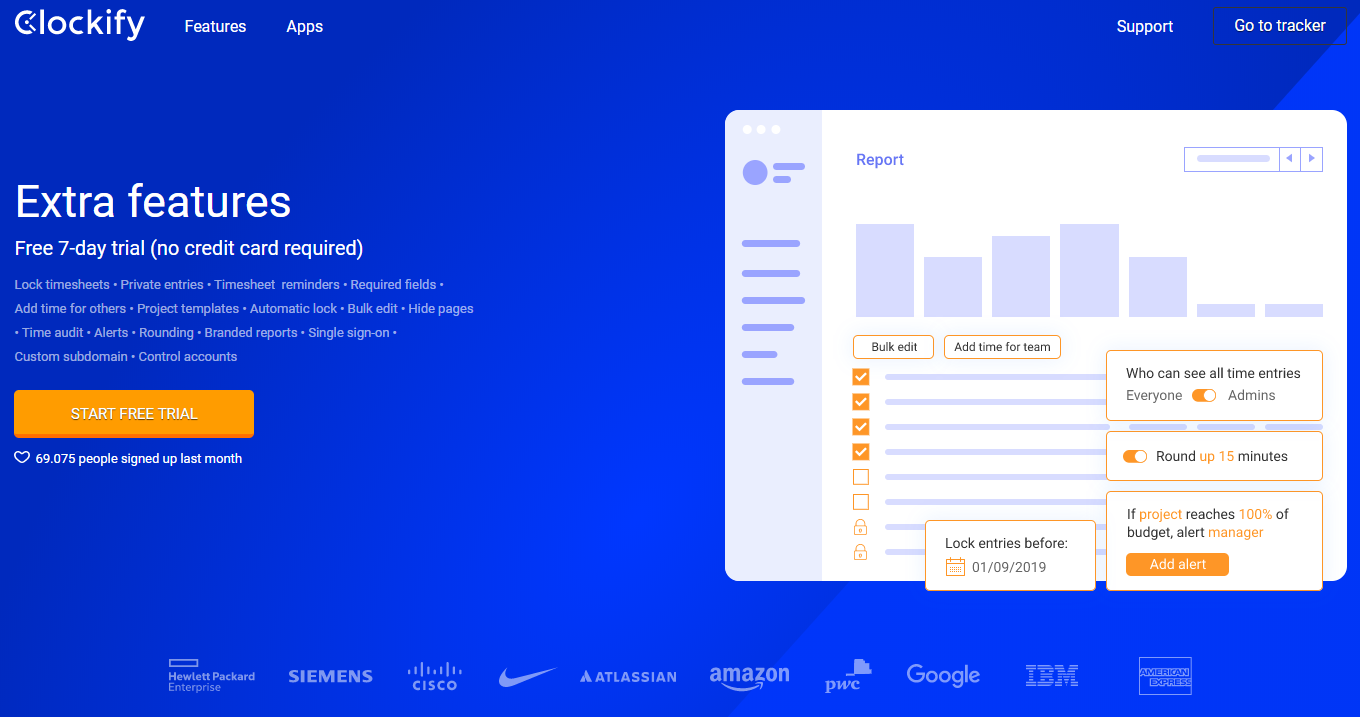
Una vez creado el repositorio, ya tenemos todo listo para empezar a trabajar, pero antes, elegimos una metodología de gestión del proyecto que más nos convenga, teniendo en cuenta los requisitos del usuario (***Historias de usuario***) y posteriormente los requisitos funcionales (***Requisitos de funcionalidad***) que el equipo de desarrollo debe tener en cuenta.

También, se tiene que considerar cual será el alcance del proyecto, delimitando hasta dónde llegan los límites mínimos.

Para ello, se tendrán que crear los diferentes bloques en los que se divide el proyecto, y especificar los requisitos concretos de cada uno.

De manera que nosotros, en este caso, escogemos trabajar, primero, en un prototipo funcional que permita al *product owner* y al equipo de desarrollo tener una idea general de cuál será el resultado del proyecto. Una vez hecho esto, seguiremos trabajando bloque por bloque (sin tocar un bloque sin acabar el anterior) hasta que el proyecto quede hecho al 100% .

## **Clockify**

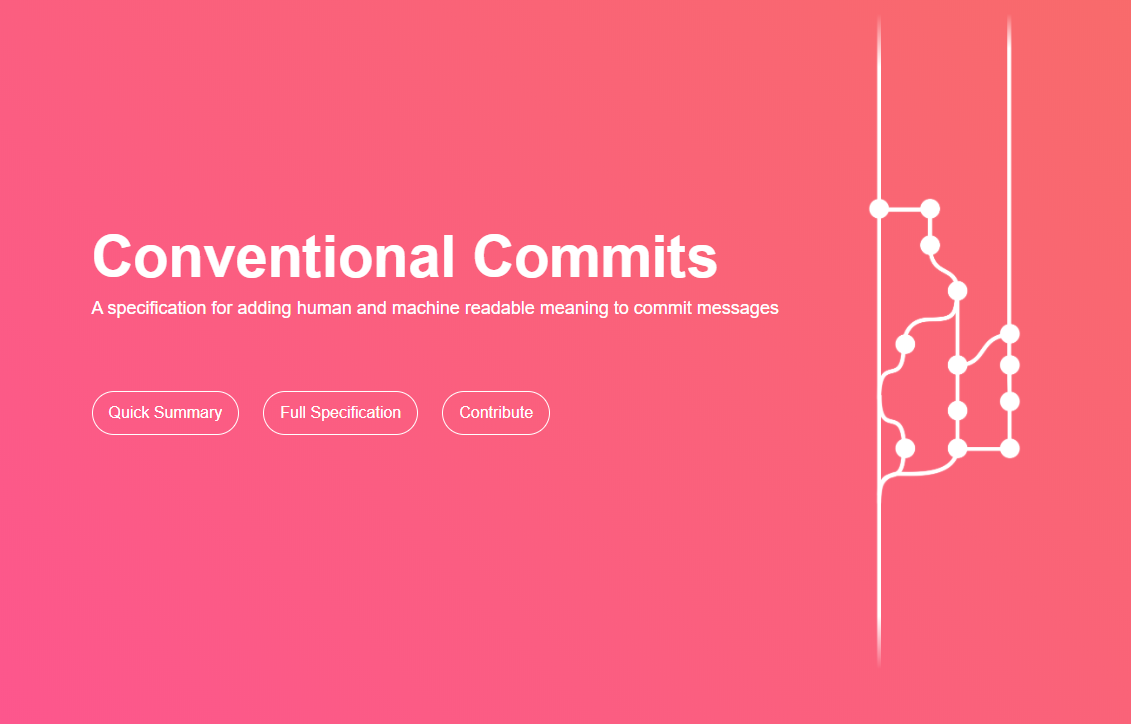
*Clockify* es una aplicación web que nos permite tener un registro de las horas que cada uno le dedica al proyecto. 

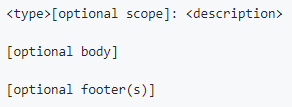
En ella podemos diferenciar los tiempos por minutos y horas e incluso compartir un ***Workspace*** donde ir añadiendo los tiempos que cada uno de los integrantes de la pareja invierte en cada parte del proyecto.



## **Conventional Commits**

Para poder hacer los commits de una manera que podamos entender qué tipo de modificaciones hemos hecho en cada uno utilizamos el estándar de ***Conventional Commits.***



Ilustración 5 - Conventional Commits

Este se basa en seguir una estructura donde asignamos el tipo de commit, a donde apuntan los cambios realizados, una breve descripción. Las partes que se encuentran entre paréntesis son opcionales y solo se añaden si el commit lo requiere; las que van entre “**<>**” son obligatorias.

Ilustración 6 - Estructura commit

VSCode tiene una extensión para que nos sea más fácil realizar estos commits.

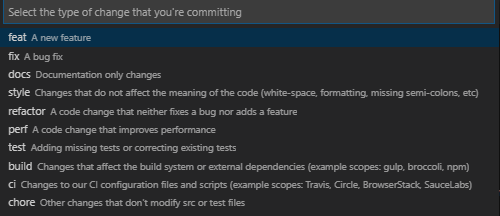


Ilustración 7 - Tipos de commits

# **Programación**

La programación de este proyecto es la base, ya que con lo que hemos aprendido durante el curso, debemos programar el acceso a la base de datos, la parte lógica y el invocar al programa VLC con Python.

## **Descripción técnica**

## **Arquitectura de aplicación**

La arquitectura se divide en tres: **Acceso a datos, Lógica e Invocar VLC**. Estas se importan al ***Main*** desde donde se invocan tocas las funciones necesarias para ejecutar el programa entero.

## **Tecnologías utilizadas**

## Resultado de imagen de tdd png**TDD**

Ilustración 8- TDD

Para realizar los programas nos basamos en usar ***Test Drive Developement*** que consiste en utilizar casos test, hacer *debugging* al código y refactorización; repetir este bucle hasta acabar e trabajo.

## **Sistema SOLID**

En el trabajo nos basamos principalmente en estos dos puntos:

***Single Responsability Principle* (SRP)**, consiste en fraccionar el código en funciones aisladas para que cada método se encargue de una única labor.

***Open/Closed Principle*** **(OCP)**, se define como el código abierto a la extensión, cerrado a la modificación.

## Resultado de imagen de SOLID programming png**Barricadas**

Ilustración 9 - SOLID

Utilizamos código para comprobar que los *inputs* del programa son válidos y casos test para testear su funcionalidad. Cada función principal de cada módulo tiene un ***assert*** como precondición y una postcondición.

## Resultado de imagen de barricada png**Parte lógica**

Ilustración 10 - Barricadas

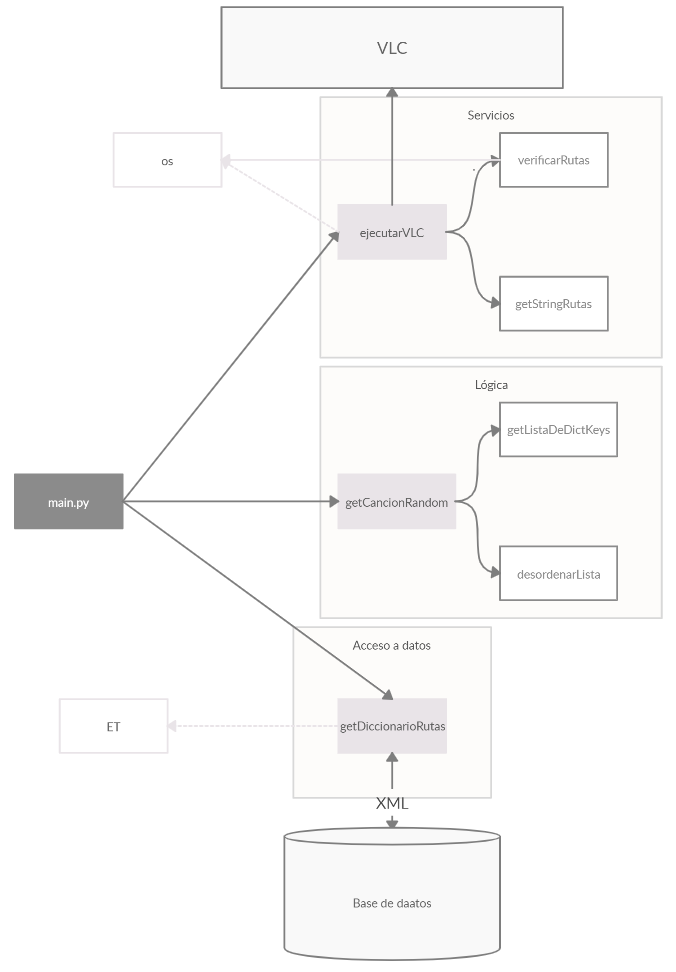
Creamos una lista aleatoria de canciones a partir de un diccionario que contiene como *input* la información de las canciones. En el proceso, comprobamos la existencia de las rutas de cada una de las canciones con la función ***access***. No se permite usar el orden aleatorio que tienen los diccionarios en *Python*. Hay que usar un método que utilice el procedimiento *random* y que genere números aleatorios.

Una vez hecha la lógica random hay que invocar el programa vlc. Corroboramos la existencia del programa en la ruta proporcionada por el usuario, sino salta un mensaje de error personalizado y se finaliza el programa. Una función transforma la lista en un *string* que forma parte de otro mayor para crear el comando que ejecutable por **popen**.

Ilustración 11 - Lógica



## **Diagrama de Componentes**

b

## **Metodología de Desarrollo**

## **Prototipada en espiral**

En nuestro proyecto, hemos escogido esta porque nos parecía interesante la idea de crear un prototipo que funciones y posteriormente ir avanzando desde ahí ya que serviría también para ir probando que vamos haciendo mal y que bien.

# **Lenguaje de Marcas**

Ilustración 12 - XML validator

La introducción a los lenguajes de marcas consistía en explicarnos que el lenguaje XML sirve para crear lenguajes de marcas a partir del mismo y que tiene un esquema que lo soporta.

## Resultado de imagen de xml png**XML**

Ilustración 13 - XML

Realizamos un primer XML a mano que contiene una librería de canciones junto con la información de estos.

Una vez hecho esto, comprobamos el XML con una herramienta web.

## **XSD**

Creamos un esquema de definición mediante una herramienta web. Nos daba el XSD ya creado y tuvimos que añadir comentarios de porqué se utilizaban algunos atributos y el porqué de sus valores.

Ilustración 14 - XSD



Ilustración 15 - XML Schema Generator

## **Estructura en árbol del xml**

La ilustración 16 contiene la estructura del XML que hemos utilizado en el proyecto para almacenar la información. Contiene marcados, como elementos más importantes, las etiquetas y los atributos que almacenan la información que va a analizar la lógica desde *Python.* La etiqueta *library* tiene como hijo la etiqueta *tracks,* que a su vez tiene como hijo la etiqueta *track.* Por último, *track* contiene dos atributos: *id* y *ruta,* que es la única información que nos servirá de utilidad para poder lanzar las canciones a través del programa VLC.

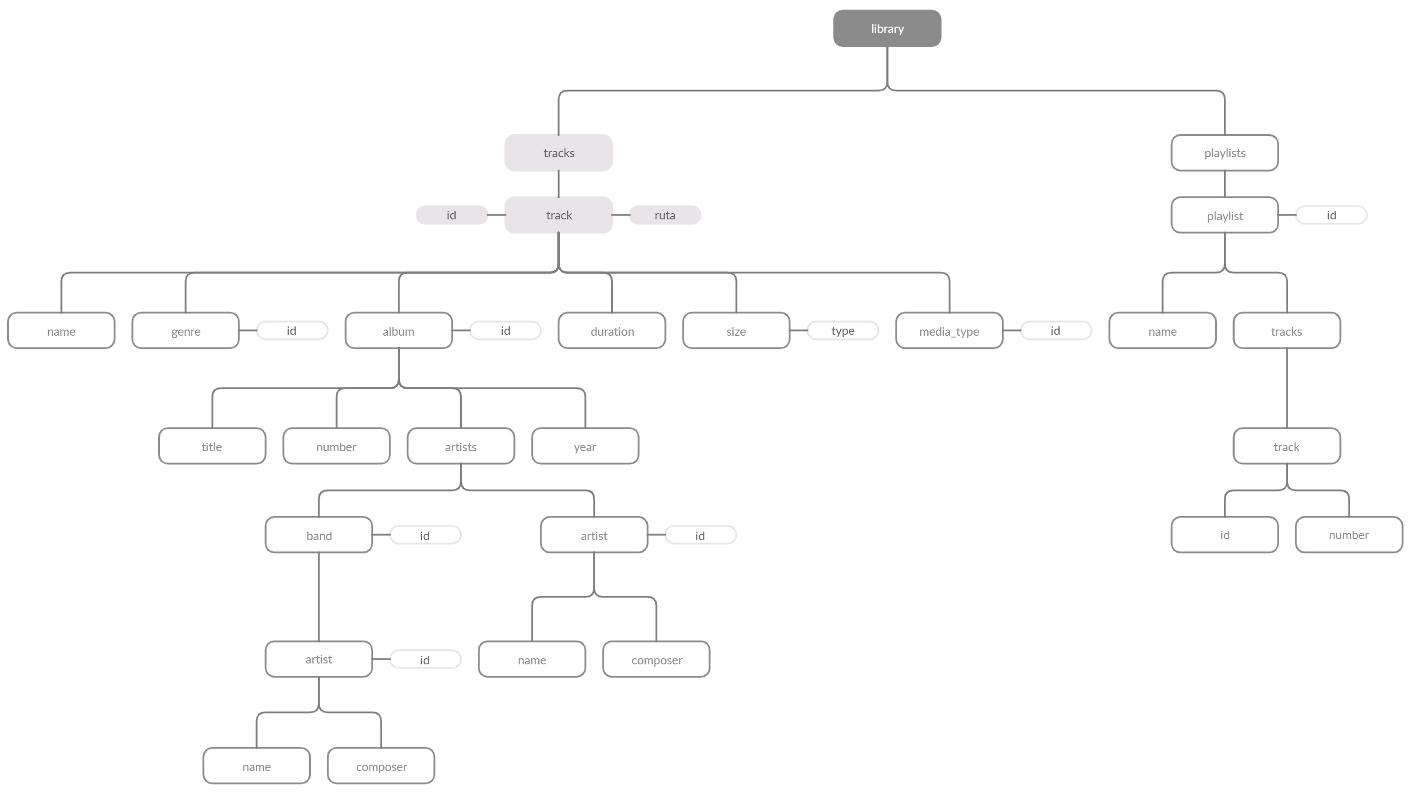
****

Ilustración 16 - Árbol XML

# **Base de Datos**

La asignatura de BBDD no aporta mucho en este proyecto. Aún así, aprendimos como exportar una base de datos en MySQL a un documento XML con comandos.

Ilustración 17 - Exportar BBDD a XML

La página nos muestra dos maneras de exportar la base de datos, una con los datos introducidos y otra sin.

En nuestro, caso utilizamos la que añade los datos ya que son necesarios para posteriormente *parsearlos* y utilizarlos en la parte lógica del proyecto

# **Sistemas Informáticos**

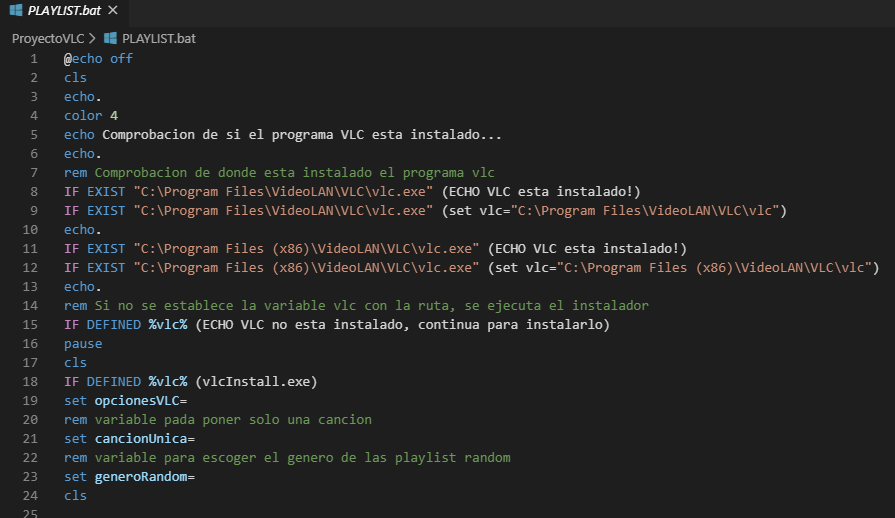
Mediante comandos MS-DOS creamos las carpetas correspondientes del repositorio local, la librería de música a la que llamaremos desde el programa para acceder a las canciones e instalamos el programa VLC, si no está instalado. 

Ilustración 18 - Playlist .bat

Llegamos a la conclusión de que el programa VLC, lanza un error si la sintaxis del comando para ejecutarlo es incorrecto. Tampoco funciona si tratamos de lanzar cualquier comando vlc si no accedemos primero al directorio en el cual se encuentra mediante una ruta absoluta (otra opción es añadir esa ruta al *path*).

La sintaxis correcta para ejecutar el programa VLC remotamente mediante línea de comandos es la siguiente:

>>> <ruta comando *vlc*> <archivos a ejecutar> [opciones]

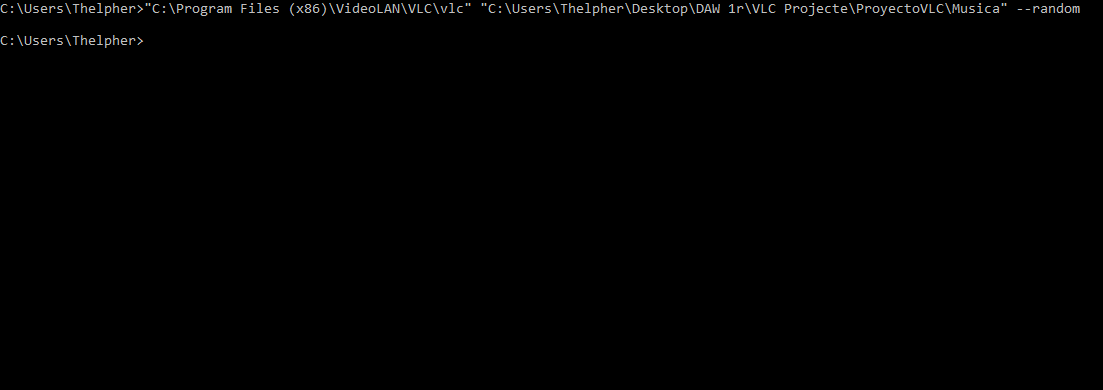


Ilustración 20 – Ejemplo comando vlc

# **Conclusiones**

# **Cosas guays**

Una de las cosas a destacar de nuestro proyecto es la implementación de una función que nos crea una copia de la lista y la utiliza en un while hasta que este vacía y que vaya añadiéndolas en otra lista con un valor aleatorio.

De esta manera, nos quitamos el problema de que aparezca por casualidad dos veces el mismo número random ya que, aunque salga de nuevo el mismo número ya no simboliza la misma canción.

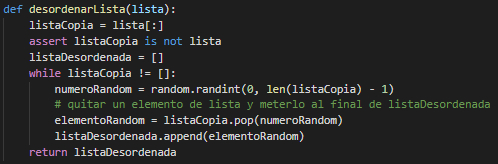


Ilustración 19 - Lógica

# **Posibles mejoras**

* En comparación con otros proyectos, el nuestro no tiene un sistema que compare la canción actual con la siguiente para que no repita el artista.
* La implementación de la ruta absoluta solo permite que el equipo de Dani Arqués ejecute el programa, Sería posible crear un módulo aparte que cambie la ruta proveniente del XML y de la BBDD que cambie la ruta absoluta con una variable PATH.
* La posibilidad de que nuestro parseador funcionase con otras estructuras de XML.

# **Dificultades**

* En la parte de la lógica teníamos muchos problemas y perdimos mucho tiempo por culpa de las “//” de las rutas.
* En los test intentamos hacer un assert que nos capture una Expection en específico de una barricada try.

# **Webgrafía**

* Pull Request
  + <https://help.github.com/en/articles/about-pull-requests>
* Como volver a un commit anterior
  + <https://es.stackoverflow.com/questions/41489/volver-a-commit-anterior>
* Slides Git Classroom
  + <https://classroom.google.com/u/1/c/NDMyNjA1NjcwNDda/m/NDMzMjIwNzg0Nzda/details>
* Repositorio compartido
  + <https://classroom.google.com/u/1/c/NDMyNjA1NjcwNDda/m/NDMzMjIwNzg0Nzda/details>
* Conventional commit
  + <https://www.conventionalcommits.org/es/v1.0.0-beta.3/>
* XML Validator
  + <https://www.xmlvalidation.com/index.php?id=1&L=0>
* XML Schema Generator
  + <https://xml.mherman.org/>
* Exportar BBDD a XML
  + <http://www.tic2.org/WebTecnica/Programas/Aplicaciones/MySQL/exportar_XML.html>
* Comandos VLC
  + <https://wiki.videolan.org/Documentation:Command_line/>