

**Documentación: Gestión de Proyectos con Git y GitHub**

CFGS DUAL DESARROLLO DE APLICACIONES WEB – CIFP Francesc Borja Moll

Daniel Arqués Toro y Daniel Jonathan Zurita Mena

֎ Índice General

[ **Índice de Ilustraciones** 2](#_Toc25943302)

[**1.** **Introducción** 3](#_Toc25943303)

[**2.** **Entornos de Desarrollo** 4](#_Toc25943304)

[**2.1.** **Registro en Git Hub** 4](#_Toc25943305)

[**2.2.** **Crear un repositorio en GitHub** 5](#_Toc25943306)

[**2.3.** **Gestión del proyecto** 7](#_Toc25943307)

[**2.4.** **Clockify** 8](#_Toc25943308)

[**3.** **Programación** 9](#_Toc25943309)

[**3.1.** **TDD** 9](#_Toc25943310)

[**3.2.** **Sistema SOLID** 9](#_Toc25943311)

[**3.3.** **Barricadas** 9](#_Toc25943312)

[**3.4.** **Parte lógica** 9](#_Toc25943313)

[**4.** **Lenguaje de Marcas** 10](#_Toc25943314)

[**4.1.** **XML** 10](#_Toc25943315)

[**4.2.** **XSD** 10](#_Toc25943316)

[**5.** **Base de Datos** 11](#_Toc25943317)

[**6.** **Sistemas Informáticos** 12](#_Toc25943318)

[**7.** **Webgrafía** 13](#_Toc25943319)

# **Índice de Ilustraciones**

[Ilustración 1 - Introducción: Git Hub 3](#_Toc25082742)

[Ilustración 2 - Introducción: Repositorios y branch 3](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082743)

[Ilustración 3 - Introducción: Presentación Word 3](#_Toc25082744)

[Ilustración 4 - $Git log --oneline --all --graph 4](#_Toc25082745)

[Ilustración 5 - Registro en Git Hub: Sing up 5](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082746)

[Ilustración 6 - Registro en Git Hub: Menú principal 5](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082747)

[Ilustración 7 - Crear un repositorio en Git Hub - New Repository 6](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082748)

[Ilustración 8 - Crear un repositorio en Git Hub - New 6](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082749)

[Ilustración 9 - Crear un repositorio en Git Hub - Comandos Git Bash 6](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082750)

[Ilustración 10 - Crear un repositorio en Git Hub - Parámetros 6](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082751)

[Ilustración 11- TDD 8](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082752)

[Ilustración 12 - SOLID 8](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082753)

[Ilustración 13 - Barricadas 8](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082754)

[Ilustración 14 - Lógica 8](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082755)

[Ilustración 15 - XML validator 9](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082756)

[Ilustración 16 - XML 9](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082757)

[Ilustración 17 - XSD 9](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082758)

[Ilustración 18 - XML Schema Generator 9](file:///C:\Users\dajoz\Documents\Entornos%20-%20Receta\recetas\Documentación_ProyectoVLC.docx#_Toc25082759)

[Ilustración 19 - Exportar BBDD a XML 10](#_Toc25082760)

# **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.

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 9 - Crear un repositorio en Git Hub - Comandos Git Bash

## **Gestión del proyecto**

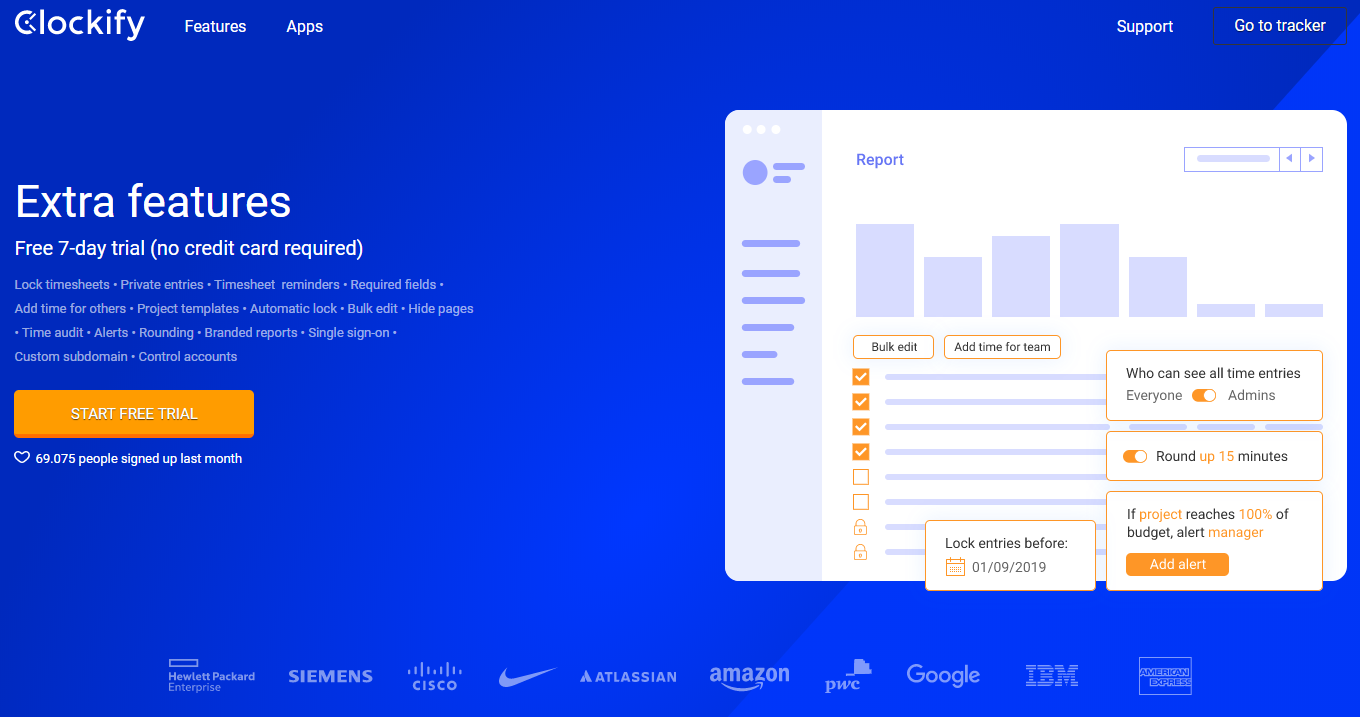
Una vez creado el repositorio, ya tenemos todo listo para empezar a trabajar, pero antes, elegimos una metodología de gestión del proyecto que más nos convenga, teniendo en cuenta los requisitos del usuario (***Historias de usuario***) y posteriormente los requisitos funcionales (***Requisitos de funcionalidad***)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

# **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 11- 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 - 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 - 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 - Lógica

## **Diagrama de Componentes**

## **Diagrama Entidad/Relación**

## **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 - 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 - 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 - XSD



Ilustración - XML Schema Generator

# **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 19 - 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**

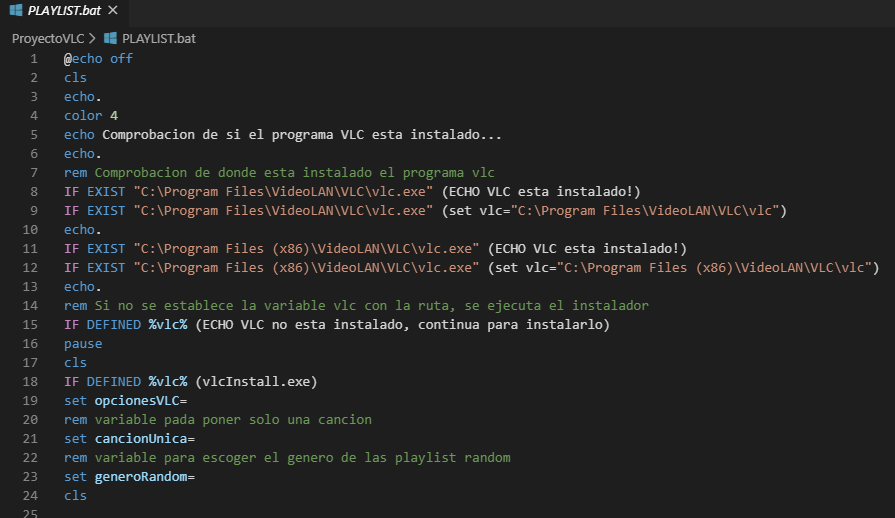
Mediante comandos MS-DOS creamos las carpetas correspondientes del repositorio local, la librería de música a la que llamaremos desde el programa para acceder a las canciones e instalamos el programa VLC, si no está instalado. 

Ilustración - 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]

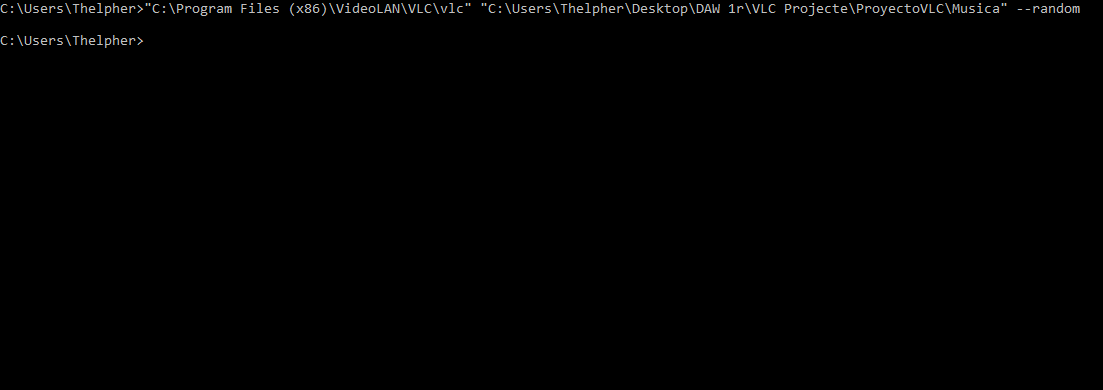


Ilustración 20 – Ejemplo comando vlc

# **Conclusiones**

# **Cosas guays**

# **Posibles mejoras**

# **Dificultades**

# **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/>