

INTRODUCCIÓN

Este documento tiene como objetivo explicar el desarrollo completo del proyecto de Programación 3 del grupo Prog III - 01, formado por Sergio Gafo Sabín, Ander Íñiguez Bustillo y Adrián Isábal Martín, del doble grado de ingeniería informática + ciencia de datos e IA (INF + CDIA). Este documento está dividido según las distintas tareas que se fueron realizando a lo largo del desarrollo del proyecto, y según las cuales se organizó este mismo. El proyecto, con nombre TubeKeeper, es una aplicación desarrollada en Java Swing que tiene como propósito hacer las veces de gestor de descargas de datos de YouTube, plataforma a la que se conecta mediante la API [JavaTube](#). Debido a diversos motivos, la aplicación sólo trabaja con metadatos brindados por la API, y no con vídeos o audio de YouTube. A lo largo de este documento se explorarán en detalle las distintas fases del desarrollo que la aplicación ha tenido desde sus inicios hasta llegar a ser funcional.

T1 Implementación de la API

En primer lugar, decidimos probar a ver si la API era funcional para ver si podíamos seguir adelante con nuestra idea de proyecto o era necesario repensarlo. Para ello, empezamos a crear un proyecto de Gradle, y tras aprender sobre la estructura de este tipo de proyectos y su funcionalidad, implementamos el nuestro propio mediante un gradle init del que posteriormente modificamos el archivo settings.gradle para incluir el nombre final que decidimos para el proyecto, TubeKeeper:

```
Java
rootProject.name = 'TubeKeeper'
```

y el archivo build.gradle para incluir todas las dependencias necesarias basándonos en el build.gradle que se podía encontrar en el repositorio de la API y modificándolo para que se ajustara a nuestras necesidades:

```
Java
plugins {
    id 'java'
    id 'application'
}

group = 'app.javatube.tubekeeper'
version = '1.0.0'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(21)
    }
}
```

```
}

application {
    mainClass = 'main.Main'
}

sourceSets {
    main {
        java {
            srcDirs = ['src']
        }
        resources {
            srcDirs = ['resources']
        }
    }
    test {
        java {
            srcDirs = ['test']
        }
    }
}

repositories {
    mavenCentral()
    maven {
        url = 'https://jitpack.io'
    }
}

dependencies {
    implementation 'com.github.felipeucelli:javatube:06b8313744'
    implementation 'org.json:json:20231013'
    implementation 'com.github.felipeucelli:nodejs_wrapper:c604a04b2d'
    implementation 'org.xerial:sqlite-jdbc:3.45.2.0'
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.9.2'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.9.2'
    implementation 'org.slf4j:slf4j-simple:2.0.17'
}

test {
    useJUnitPlatform()
}
```

También tuvimos problemas con el archivo .gitignore, que hubo que modificar en varias ocasiones para asegurarse de incluir los archivos de gradle necesarios para la compilación y excluir aquellos que fueran resultado de esta.

Tras ello, verificamos la funcionalidad de la API con una prueba simple en el src/main/Main.java:

```
Java
package main;
import com.github.felipeucelli.javatube.Youtube;
public class Main {
    public static void main(String[] args) {
        try {
            Youtube yt = new
Youtube("https://youtu.be/dQw4w9WgXcQ?si=5RA0rSRyQVzT130u");
            System.out.println(yt.getTitle());
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Dado que se mostraba correctamente el título del vídeo por consola, la API funcionaba correctamente, por lo que seguimos a la siguiente fase del proyecto: la estructuración del mismo.

T2 Estructuración del Proyecto

En esta fase debíamos organizar la estructura de código que utilizaríamos a lo largo del desarrollo. La guía del proyecto era bastante detallada en este aspecto, pero aunque pensamos que el hecho de que fuera un proyecto de Gradle podría dar problemas para integrar la estructura de carpetas requerida, funcionó sin problema. Para asegurarnos de mantener la estructura completa a lo largo del proyecto, creamos un archivo dummy .gitkeep en cada subcarpeta del proyecto para que se subieran al repositorio remoto (dado que git ignora las carpetas vacías) y que no hubiera dudas con este aspecto.

```
Java
touch src/gui/.gitkeep
touch src/domain/.gitkeep
touch src/db/.gitkeep
touch src/io/.gitkeep
touch resources/db/.gitkeep
touch resources/images/.gitkeep
touch resources/data/.gitkeep
touch doc/.gitkeep
touch lib/.gitkeep
touch log/.gitkeep
touch test/.gitkeep
```

T3 Diseño de las Views

Tras completar la configuración de Gradle, nos pusimos manos a la obra con el desarrollo del proyecto, aunque pronto surgió un problema principal: no contábamos con un diseño previo de la interfaz de usuario. Para solucionarlo, utilizamos una herramienta externa, una tablet (link abajo), con la que diseñamos la experiencia visual que tendría el usuario al iniciar la aplicación. Esta fase nos llevó más tiempo del esperado, ya que en un principio habíamos planteado un número menor de vistas del que finalmente acabamos implementando. Como consecuencia, aunque realizamos el diseño general al inicio del proyecto, durante la fase de programación tuvimos que volver atrás en varias ocasiones para modificarlo y pulir ciertos aspectos, como por ejemplo la barra lateral extensible.

Nuestro objetivo principal fue diseñar una interfaz simple, con pocos elementos y un aspecto actual. Por ello, optamos por un enfoque minimalista, que puede apreciarse en la página inicial de la aplicación, donde se muestra el logotipo junto con una barra de búsqueda.

Por otro lado, la creación de las distintas vistas en Swing resultó una tarea bastante laboriosa, no tanto por su complejidad técnica, sino porque al comienzo no teníamos del todo claro el rumbo del proyecto. En nuestra idea inicial habíamos diseñado varias vistas, siendo la primera una pantalla de inicio con el logotipo y la barra de búsqueda. Sin embargo, el planteamiento de las vistas posteriores generó cierta confusión, ya que cada integrante del grupo tenía una idea distinta sobre cómo debía evolucionar la aplicación. Finalmente, tras debatir y ponernos de acuerdo, conseguimos un diseño coherente, limpio y visualmente atractivo.

[HAZ CLICK AQUÍ PARA VER EL DISEÑO DE LA GUI EN LA TABLET](#)

T4 Implementación de los componentes de las Views

Implementar las vistas no fue lo más complicado de todo ya que hicimos una muy buena labor diseñando y decidiendo la estructura de las vistas. Para crear las vistas usamos Java Swing y las siguientes vistas fueron creadas:

- SearchView: Esta es la vista principal, la vista de descargar. En esta vista podemos ver la simplicidad que queríamos darle a la app y de la que hablábamos previamente. Trae en ella el logo que escogimos (y diseñamos) y una barra de búsqueda funcional en la que al insertar una url de un video empieza la descarga del mismo. Además, al iniciar la descarga, aparece en la parte derecha de la pantalla una barra indicando el progreso de ella.
- PlaylistMenuView: A esta vista se accede de dos maneras: la primera (y probablemente la más usual) es desde la barra lateral desplegable que hay a la izquierda de la pantalla (la cual se despliega al clickar en el ícono de las tres rayas que hay en la parte superior). La segunda manera de acceder a la vista la veremos

más tarde en la siguiente vista. En esta vista se ven todas las playlists que hay creadas en la base de datos de la aplicación. De cada una de ellas se ve el nombre y la miniatura del primer vídeo. Cuando haces click en una de ellas, se entra a la vista PlaylistView.

- PlaylistView: En esta vista se ven los datos de todos los videos que componen la playlist seleccionada además de los metadatos de la propia playlist. Además, hay un botón de retroceso en la parte superior con el que se puede acceder de vuelta a la vista PlaylistMenu.
- DownloadsView: En esta vista se ven todos los vídeos descargados con la aplicación junto a sus metadatos. Al hacer click en un vídeo, se abre una pestaña con aún más información del vídeo además de un hipervínculo para visitarlo. Además, en caso de que el título del vídeo no quepa en la pantalla, hay una posibilidad de verlo entero gracias a las tooltips implementadas. Si se sitúa el cursor encima del nombre del vídeo, se verá el título entero como tooltip. Además, se puede buscar un vídeo en específico si se recurre a la barra de búsqueda en la parte superior. También se puede ordenar por cada una de las 4 columnas haciendo click sobre la cabecera de ellas (Ej: hacer click en “Autor” ordenará los vídeos alfabéticamente por su autor).
- SettingsView: A esta vista se accede mediante el botón de rueda dentada que se halla en la parte inferior izquierda de la pantalla (en la parte inferior de la barra lateral). Al hacer click sobre él, se abre una nueva ventana de configuración con la que se puede cambiar los valores predeterminados de la aplicación así como borrar la base de datos actual.

T5 Implementación de la base de datos

Implementar la base de datos fue un vaivén de problemas y dolores de cabeza ya que no sabíamos qué queríamos guardar realmente en ella. Porque cuando descargas un vídeo, ¿qué guardas de él? ¿Su imagen? ¿Su audio? Tuvimos muchas dudas a lo largo de la creación y el desarrollo de la base de datos, pero finalmente hemos conseguido solventar todo y dejar los datos limpios y bien estructurados. Al final, cosas como la miniatura o el audio se han tenido que gestionar de otras maneras. Por ejemplo, la miniatura se guarda como una url y se accede a ella cada vez que se quiera mostrar.

En la base de datos que se encuentra en resources/db/[tubekeeper.db](#) podemos encontrar 2 tablas que se crean en src/db:

- Tabla de Vídeos: En esta tabla se guardan y parsean los datos de cada vídeo. Estos se establecen en DatabaseManager. Los datos guardados son:
 - id
 - youtube_id (url cortada)
 - título

- autor
 - descripción
 - fecha de publicación
 - duración (segundos)
 - thumbnail_url (url de la miniatura)
 - visitas
 - url
 - file_size (cuanto ocupa)
 - playlist_id (si lo tuviese: es clave foránea)
- Tabla de Playlists: En esta tabla se guardan y parsean los datos de cada playlist. Los vídeos apuntan a las playlist, por lo que en las playlist no se guarda una lista de vídeos ni nada por el estilo. Estos datos se establecen en el archivo DatabaseManager. Los datos guardados son:
- id
 - título
 - autor
 - descripción
 - visitas
 - última modificación
 - owner_id
 - owner_url (url del creador)
 - url
 - thumbnail_path (directorio de la miniatura)

T6 Implementación de los threads

En TubeKeeper, hemos querido hacer muchas cosas a lo grande y es obvio que íbamos a tener muchos problemas para que ocurriese. Es por ello que la implementación de hilos en el programa es algo vital y de suma importancia. En el programa se pueden ver un total de 5 hilos (3 categorías).

- Hilo de Descargas de la UI: Este hilo ejecuta todo el flujo de descarga de un enlace (vídeo o playlist) en segundo plano, para no bloquear el hilo de eventos de Swing (la interfaz). Esta implementación fue bastante positiva para el funcionamiento de la app ya que sin ella, habría que esperar hasta 1 minuto para poder ver la primera vista completamente.

Cada descarga se ejecuta en su propio hilo dedicado. Esto permite que la interfaz siga respondiendo mientras se realizan operaciones de red, disco y base de datos.

El hilo se encarga de:

- Resolver el tipo de enlace (vídeo o playlist).
- Descargar el contenido con JavaTube/FFmpeg.
- Guardar los metadatos en SQLite.
- Actualizar la barra de progreso de la descarga.

- Hilo del gestor FFmpeg: En el arranque, TubeKeeper lanza un hilo de gestión de FFmpeg. Este hilo comprueba si FFmpeg está disponible; si no lo está, solicita permiso al usuario, descarga los binarios desde GitHub y los instala en segundo plano. Todos los diálogos de confirmación se realizan sobre el Event Dispatch Thread mediante SwingUtilities.invokeAndWait, para mantener la UI coherente.

- Hilos del gestor EDT de Swing:
 - CheckIfAllFinished: Su función es asegurar que la actualización del botón "Close" de la ventana de descargas se hace siempre en el hilo de eventos de Swing. Además intenta evitar problemas de concurrencia actualizando componentes Swing desde el hilo de descarga.
 - TubeUtils.downloadVideo: Su función es mostrar un JOptionPane de error desde el EDT, aunque la descarga se esté ejecutando en un hilo de fondo.
 - FFmpegManager: Hemos hecho uso de varios métodos SwingUtilities.invokeAndWait / invokeLater para mostrar diálogos de confirmación y error desde el hilo de FFmpeg.

T7 Conexión real de los objetos con la web vía API

Conectar los objetos con la web ha sido un total caos para todos los que hemos participado en la creación del proyecto ya que hemos tenido que consultar la documentación de la API Javatube incontables veces. Esta API lo que hace es brindarnos datos de vídeos de diversas plataformas. Gracias a ello, es posible la descarga de éstos y de sus metadatos en nuestra aplicación. La conexión se realiza siempre a través de objetos de la librería com.github.felipeucelli.javatube:

- Para vídeos individuales utilizamos la clase Youtube:
 - new Youtube(url) inicializa el objeto a partir de la URL pegada por el usuario.
 - Métodos como streams().getHighestResolution(), streams().getOnlyAudio() o streams().getDefaultResolution() devuelven los distintos streams disponibles para descarga.
 - A partir de ese objeto rellenamos nuestro dominio Video con metadatos como título, descripción, duración, views, etc.

- Para playlists utilizamos com.github.felipeucelli.javatube.Playlist:
 - new Playlist(url) construye la lista remota.

- `getTitle()`, `getDescription()`, `getOwner()`, `getVideos()` nos permiten poblar nuestro objeto `domain.Playlist` y obtener las URLs de cada vídeo que contiene.

En nuestro código, esa conexión con la API se hace principalmente en dos sitios:

- En `DownloadManager`:
 - En `downloadVideo(String url, Playlist playlist)` se crea un `Youtube` a partir de la URL, se obtienen los streams para descargar el fichero físico (a través de `TubeUtils.downloadVideo`) y se construye un `Video` de dominio con todos los metadatos devueltos por la API.
 - En `downloadPlaylist(String url)` se instancia `com.github.felipeucelli.javatube.Playlist` a partir del enlace de la playlist y, con esos datos, se crea un `domain.Playlist` que luego se guarda en la base de datos.
- En `TubeUtils.downloadVideo`:
 - Se vuelve a utilizar el objeto `Youtube` para pedir los distintos streams y realizar la descarga del vídeo o del audio, combinándolos con FFmpeg según la política de calidad configurada por el usuario.

Toda esa lógica funciona en hilos de fondo para no bloquear la interfaz. Cada vez que el usuario pega un enlace, detrás se crea un objeto de la API Javatube, se consulta la web, se descargan los streams y se sincronizan los resultados con nuestros objetos de dominio (`Video`, `Playlist`) y con la base de datos SQLite. El trabajo más complicado ha sido entender en qué momento llamar a cada método de la API, interpretar los errores que devuelve (especialmente cuando YouTube cambia el formato de sus respuestas) y adaptar la aplicación para que falle de forma controlada cuando Javatube no puede parsear ciertos JSON.

T8 Refactorización recursiva de funciones

Durante el desarrollo del proyecto hemos tenido que refactorizar el código varias veces de forma casi “recursiva”: cada cambio en una parte del flujo arrastraba a otras capas (dominio, GUI, persistencia), obligándonos a volver una y otra vez sobre las mismas funciones hasta encontrar un diseño más limpio.

Algunos ejemplos claros:

- `DownloadManager` y descarga de playlists
Inicialmente la lógica de descarga de una playlist estaba mezclada en un solo método grande que:
 - Detectaba si la URL era vídeo o playlist.
 - Descargaba el contenido.
 - Actualizaba la UI.

- Insertaba datos en la base de datos.
- Tras varias iteraciones se separó en funciones más pequeñas:
 - download(String url) → crea el hilo y delega en downloadRec.
 - downloadRec(...) → controla el flujo según el LinkType (VIDEO, PLAYLIST, PLAYLIST_VIDEO).
 - downloadVideo(String url, Playlist playlist) → responsabilidad clara: descargar un único vídeo y persistirlo.
 - downloadPlaylist(String url) → crear la playlist de dominio y guardarla en la base de datos.
- Este refactor no se hizo de una sola vez, sino en varias rondas: primero se extrajeron pequeñas funciones, luego se ordenó la lógica de estados y finalmente se ajustó la gestión de errores para que el mensaje que ve el usuario sea coherente.
- Clases de dominio y DAOs

Video y Playlist empezaron siendo clases muy pegadas a la API externa. Con el tiempo se añadieron:

 - Constructores específicos para instancias que vienen de la base de datos (con dbID).
 - Getters y setters adicionales para poder reconstruir el estado completo a partir de un ResultSet.
 - Métodos utilitarios como setThumbnail() en Playlist para recalcular la miniatura a partir de los vídeos.
- Cada vez que se tocaba la estructura de estas clases, era necesario refactorizar los DAO (VideoDAO, PlaylistDAO) para mantener la coherencia entre el modelo de dominio, la API y la persistencia.

En resumen, la “refactorización recursiva” ha consistido en ir descomponiendo funciones monolíticas en métodos más pequeños y reutilizables, corrigiendo dependencias circulares, y ajustando la firma de los métodos tantas veces como ha hecho falta hasta conseguir un flujo más claro: API → dominio → base de datos → GUI.

T9 Optimización

La optimización en este proyecto no ha sido tanto a nivel de algoritmos complejos como de experiencia de usuario y rendimiento percibido:

- Uso de hilos en descargas

El mayor salto de rendimiento percibido viene del hecho de ejecutar las descargas en hilos separados mediante DownloadManager. Esto permite:

 - Mantener la interfaz siempre fluida.
 - Lanzar varias descargas en paralelo sin bloquear el Event Dispatch Thread de Swing.
 - Actualizar las barras de progreso de cada descarga de forma independiente.

- Gestión de FFmpeg en segundo plano:
La comprobación e instalación de FFmpeg (FFmpegManager.FFmpegThread) también se realiza en un hilo dedicado. Descargar ~200 MB y descomprimirlos en el hilo principal habría bloqueado completamente la aplicación durante varios segundos/minutos. Al hacer esto en segundo plano, el usuario puede seguir configurando o explorando la aplicación mientras la instalación avanza.
- Persistencia ligera con SQLite:
El uso de SQLite con conexiones cortas y operaciones sencillas (consultas sin joins complejos, inserciones directas) mantiene el acceso a disco relativamente barato. No se hace caching agresivo, pero el volumen de datos típico (historial de vídeos/playlist y configuración básica) no lo requiere.
- Uso moderado de recursos gráficos:
Las miniaturas (ImageIcon) se redimensionan en la vista (PlaylistView, PlaylistMenuView) en lugar de mantener múltiples copias en memoria a distintos tamaños. Esto reduce el consumo de memoria gráfica y mantiene la UI razonablemente rápida incluso con muchas filas.

T10 Completar la documentación

Finalmente, llegamos a la completación de la documentación. Ahora es cuando podemos decir que el proyecto tiene una estructura bastante parecida a la que se había pensado en un inicio. Obviamente, no es ni de cerca igual a la grandeza que le habíamos planteado, pero hemos hecho un gran trabajo y esfuerzo sabiendo lo que nos ha costado implementar las cosas mientras las íbamos dando en la asignatura. Estamos muy orgullosos de TubeKeeper y de cómo ha quedado, al igual que probablemente continuemos trabajando en ello a lo largo del tiempo en perfeccionarla para apoyar a la comunidad y proporcionarles herramientas útiles.

La tarea de completar la documentación ha sido bastante dura ya que recorrer todo el progreso de la aplicación es bastante complicado. En mi opinión, lo mejor que se puede hacer es leer los commits para ver el desarrollo de la app ya que así se puede ver cómo íbamos implementando cosas y arreglando otras.

Esperamos que disfrutéis y le deis uso a la app.