UNIVERSIDAD REY JUAN CARLOS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Curso académico 2018/2019

Trabajo Fin de Grado

Aplicación para grabación y publicación de vídeos docentes

Autor: Carlos Ruiz Ballesteros
Tutor: Micael Gallego

Dedico este trabajo a mis padres por su apoyo y dedicación,
a mis tutor de proyecto Micael Gallego,
y a todas los amigos que sin darme cuenta
han estado ahí.

Índice

Resumen

Aqui ira un resumen (al final)

1. Introducción

En la actualidad en muchos centros educativos, universidades y escuelas, los profesores utilizan ordenadores para hacer sus presentaciones en clase, incluso centrándose unicamente en el contenido que aparece por la pantalla o el proyector como es el caso de los profesores dentro del área de la informática.

Una de las ventajas que presenta realizar las presentaciones y las clases con un ordenador, es poder grabar dichas clases y después dejarlas a disposición de los alumnos, para que cada uno pueda volver a ver las clases a su ritmo, pudiendo revisar todo el contenido de las mismas. Además, grabar las diapositivas y el contenido realizado en el ordenador, es muy útil para la creación de cursos online, permitiendo incluso la ausencia presencial de los alumnos, pudiendo llevar el conocimiento mucho más allá del lugar en el que se imparte.

Sin embargo, grabar las clases, editarlas y ponerlas a disposición de los alumnos no es una tarea trivial. Requiere un tiempo que los profesores no pueden utilizar para la preparación del contenido de sus clases, pudiendo incluso influenciar en la calidad de las mismas. Para solventar este tiempo dedicado a la grabación, edición y publicación de las clases surge *Class Recorder*.

A continuación, en los siguientes apartados de este capítulo, se abordarán los diferentes problemas con los que los docentes y profesores se encuentran a la hora de grabar sus clases, estudiaremos como mejorar las partes que mayor esfuerzo conllevan, a que usuarios esta enfocada está aplicación y se explicará a grandes rasgos el software presentado.

1.1. Contexto y estudio del problema

Para entrar un poco en el contexto y la problemática a la hora de grabar las clases con un ordenador, se va a exponer a continuación los problemas más comunes.

Lo más importante al principio es preparar el contenido del curso o de la asignatura, como diapositivas, ejemplos, proyectos, etc. Esto supone un tiempo y esfuerzo necesario para que las clases tengan cierta calidad. El tiempo que dedique un profesor a la preparación del contenido de sus clases es directamente proporcional a la calidad de las mismas. Complementar entonces este esfuerzo grabando cada una de las sesiones impartidas, le otorga al curso más flexibilidad y le da la posibilidad a todo el alumnado de ir a su propio ritmo.

El problema que surge a la hora de grabarlas, son en su mayor parte, debido a la multitud de herramientas de las que se disponen para grabar el contenido, y la distracción que provoca en el profesor el utilizarlas, reduciendo el tiempo dedicado a la enseñanza y la calidad de la lección. A esto hay que añadir el aprendizaje de dichas herramientas, las cuales suelen ser bastante complejas por lo general, con información excesiva acerca de los formatos de los vídeos, configuraciones de vídeo y audio, etc.

Otro aspecto que puede ser problemático, es la grabación del audio de las clases. En su mayor parte, las clases se imparten de pie, leyendo y explicando diapositivas, por lo que en algunas ocasiones el micrófono del ordenador se encuentra lejos de la fuente de voz, por lo que la calidad del audio tiende a empeorar a medida que nos alejamos de este. Esto se suele solventar grabando el audio con un dispositivo móvil y después uniendo vídeo y audio, pero esto solo añadiría aún

más carga de trabajo para producir el vídeo final.

A esto hay que sumarle la edición de vídeo, desechar de este las partes irrelevantes, los descansos o interrupciones, etc. El profesor debe entonces aprender y elegir probablemente otra herramienta con la que editar dicho vídeo, contando con multitud de software (en muchas ocasiones de pago), que se utilizan de forma diferente, desviando el tema principal, que es enseñar. Además los vídeos tras editarlos, hay que renderizarlos y subirlos.

Al final nos encontramos con un proceso bastante tedioso, que lleva un aprendizaje detrás para el profesor bastante costoso, que le separa de su objetivo final. En la Figura 1 se puede observar el proceso común y las opciones que tiene un profesor para grabar sus clases.

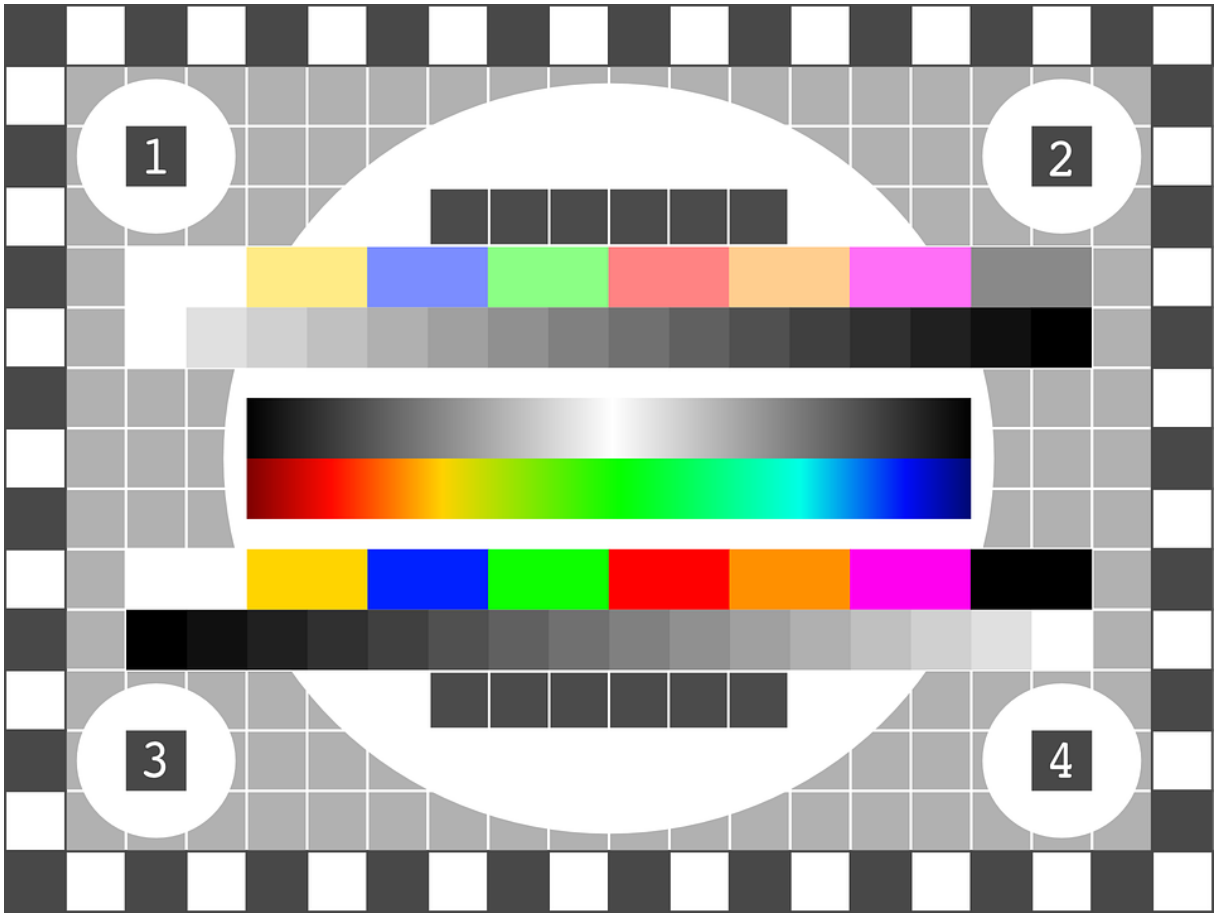

Figura 1: Proceso común de grabación de clases por ordenador

1.2. Motivaciones.

La principal motivación es reducir toda esa carga de trabajo adicional, y optimizar al máximo el tiempo necesario para grabar las clases, mejorando además los problemas mencionados en el apartado anterior.

Al final nos encontramos con que tenemos que reducir el consumo de tiempo y mejorar las siguientes tareas:

- Aprendizaje de herramientas para grabación.
- Grabación de audio por separado
- Identificar zonas en las que realizar cortes en el vídeo o juntar vídeos.
- Renderización y subida a una plataforma.

Hay que tener en cuenta que muchas de las herramientas que se utilizan comúnmente para la edición de vídeos y grabación de escritorio, son herramientas que abarcan muchos casos de uso, y que tienen funcionalidad extra, que el profesor no necesita conocer, y que pueden confundir a la hora de realizar los vídeos.

Una posible solución es la unificación del flujo de trabajo, utilizando un software unicamente para la realización de las grabaciones y simplificando la edición y grabación de vídeo únicamente a lo necesario por un profesor para grabar sus clases. Si observamos detenidamente la Figura 1, el proceso de grabación no consiste en la realización de un conjunto de pasos definidos ya que se debe elegir el software que quiere utilizar, aprenderlo, pero no hay un camino claro para alcanzar el objetivo concreto.

La motivación principal de *Class Recorder* es acabar con esta fragmentación y unificar todo el proceso en uno solo, para facilitar todo el trabajo y crear una sola plataforma con la que poder grabar, editar y subir vídeos, restando el tiempo innecesario de edición y grabación proporcionando herramientas fáciles de usar y enfocadas a la tarea en cuestión. Son muchas opciones, para llegar al mismo objetivo, que es grabar nuestras clases de la forma más cómoda posible al mismo tiempo que maximizamos el tiempo dedicado a las clases.

Queremos que el usuario pase de usar múltiples aplicaciones y herramientas, a que solo utilice una que esté enfocada exclusivamente en el objetivo principal del profesor y así poder distribuir sus clases de forma rápida y eficaz.

En la Figura 2 podemos ver como se simplifica el proceso, utilizando unicamente *Class Recorder* en comparación con el visto en la Figura 1. En las siguiente sección se explicará la aplicación final a la que se quiere llegar y el flujo de trabajo objetivo para optimizar este proceso.

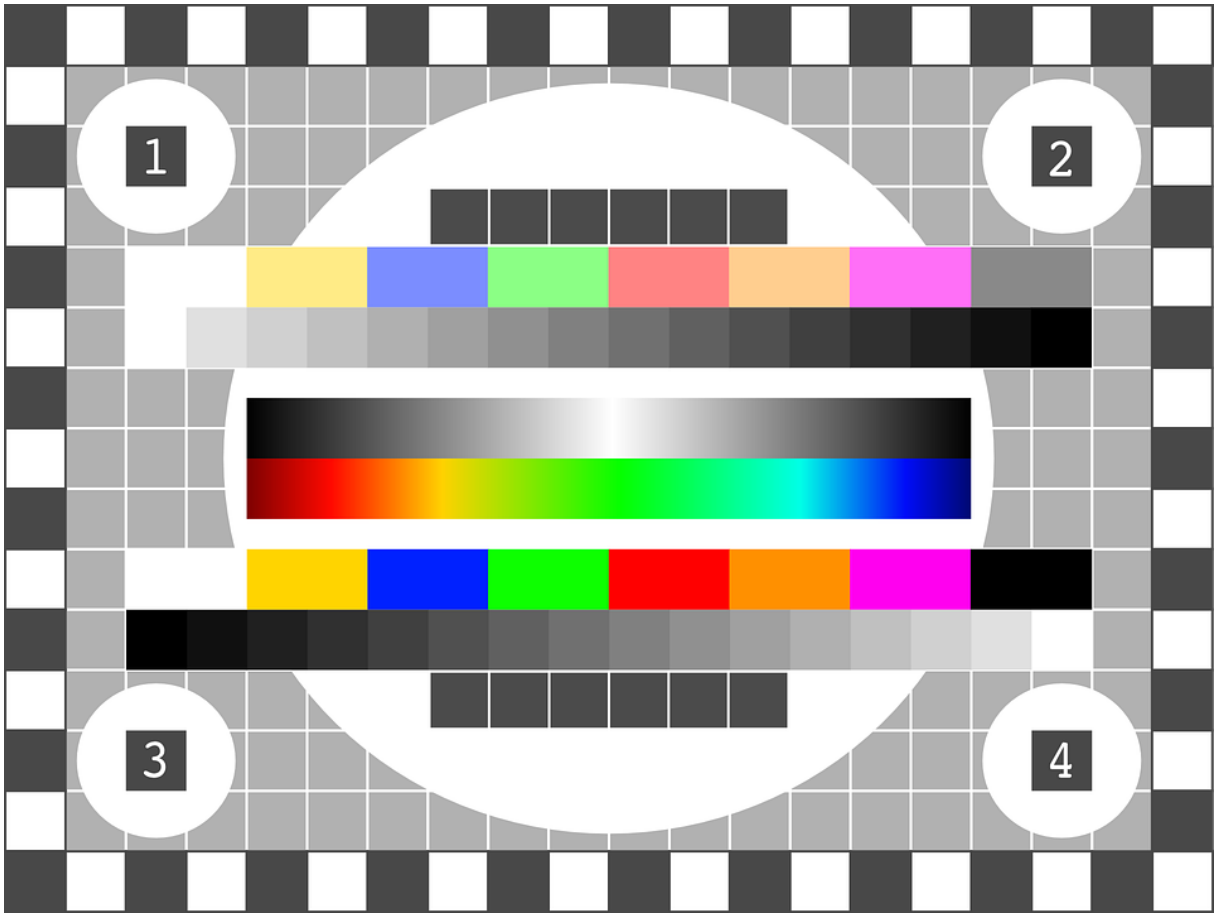

Figura 2: Proceso con Class Recorder

2. Objetivos y solución propuesta

En este proyecto proponemos una unica solución para la realización de todo este proceso de grabación y distribución de clases. Es importante recalcar que nuestro objetivo principal y el propuesto con este proyecto es crear un único flujo de trabajo, mediante las siguientes herramientas software que hemos creado:

- Aplicación de escritorio, con el que controlar (grabar, pausar), gestionar, cortar y subir los videos realizados.
- Aplicación móvil sincronizada con la aplicación de escritorio con la que controlar las grabaciones, grabar el audio y poder controlar las diapositivas.

El usuario ya no tiene que elegir entre decenas de herramientas software incompatibles muchas veces incompatibles entre sistemas operativos. En este proyecto ofrecemos una aplicación compatible, tanto en Windows como en GNU/Linux, y esta preparada para en un futuro tener soporte para Mac OS como podremos ver en el apartado 4.3 de diseño e implementación.

El usuario podrá elegir entre:

1. Realizar una sesión desde el propio ordenador y grabar unicamente la pantalla y la voz desde el micrófono del mismo.
2. Realizar una sesión controlada desde el móvil y grabar el contenido de la pantalla del ordenador y grabar el audio desde el movil.

El primer caso es ideal para clases en las que el profesor está solo enfrente del ordenador grabándose a si mismo realizando una tarea o impartiendo una lección. El segundo es ideal para impartir clases con diapositivas ya que permite al profesor levantarse, hacer algunas indicaciones y seguir grabando con el móvil su voz.

Además tanto en el ordenador como en el móvil se puede pausar y continuar la grabación, que más tarde puede ser editada o subida directamente a Youtube de forma privada.

Flujo de trabajo con ordenador:

Si el profesor decide impartir la clase unicamente con el ordenador, unicamente deberá hacer lo siguiente.

1. Abrir *Class Recorder* en Windows/Linux.
2. Crear un curso (elegirlo si ya esta creado).
3. Iniciar grabación, unicamente poniendo un nombre para el video, un framerate y un formato de contenedor (mkv, mp4).

No es necesario configurar nada, en Windows y Ubuntu se detecta la pantalla principal y el micrófono por defecto. El profesor entonces podrá comenzar su charla o clase y podrá pausar el video en los momentos que quiera. En la figura 3 se puede ver el flujo de trabajo más claramente.

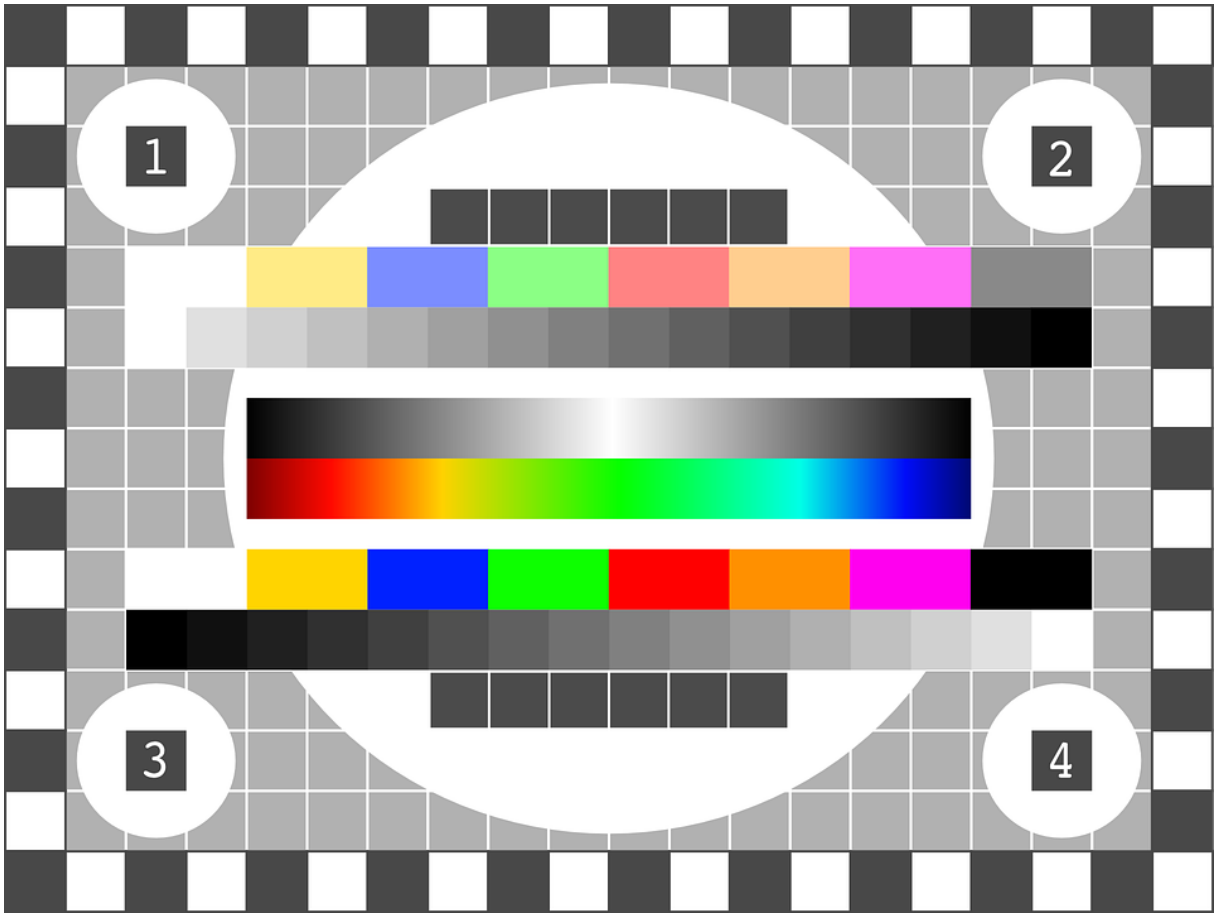

Figura 3: Flujo de trabajo con Class Recorder

Flujo de trabajo con ordenador y móvil:

En muchas ocasiones la clase no se puede impartir sentado, es por eso que *Class Recorder* ofrece la posibilidad de controlar la grabación de la pantalla de nuestro ordenador, grabar el audio desde el móvil y además poder controlar las diapositivas desde el. El flujo de trabajo sería el siguiente:

1. Abrir *Class Recorder* en Windows/Linux.
2. Consultar desde la aplicación nuestra IP local.
3. Iniciar la App de *Class Recorder* y introducir nuestra IP local.
4. Comenzar a grabar.

Así de sencillo, y la app disponde de los mismos controles que la aplicación de escritorio. En la siguiente Figura 4 se puede ver más en detalle.

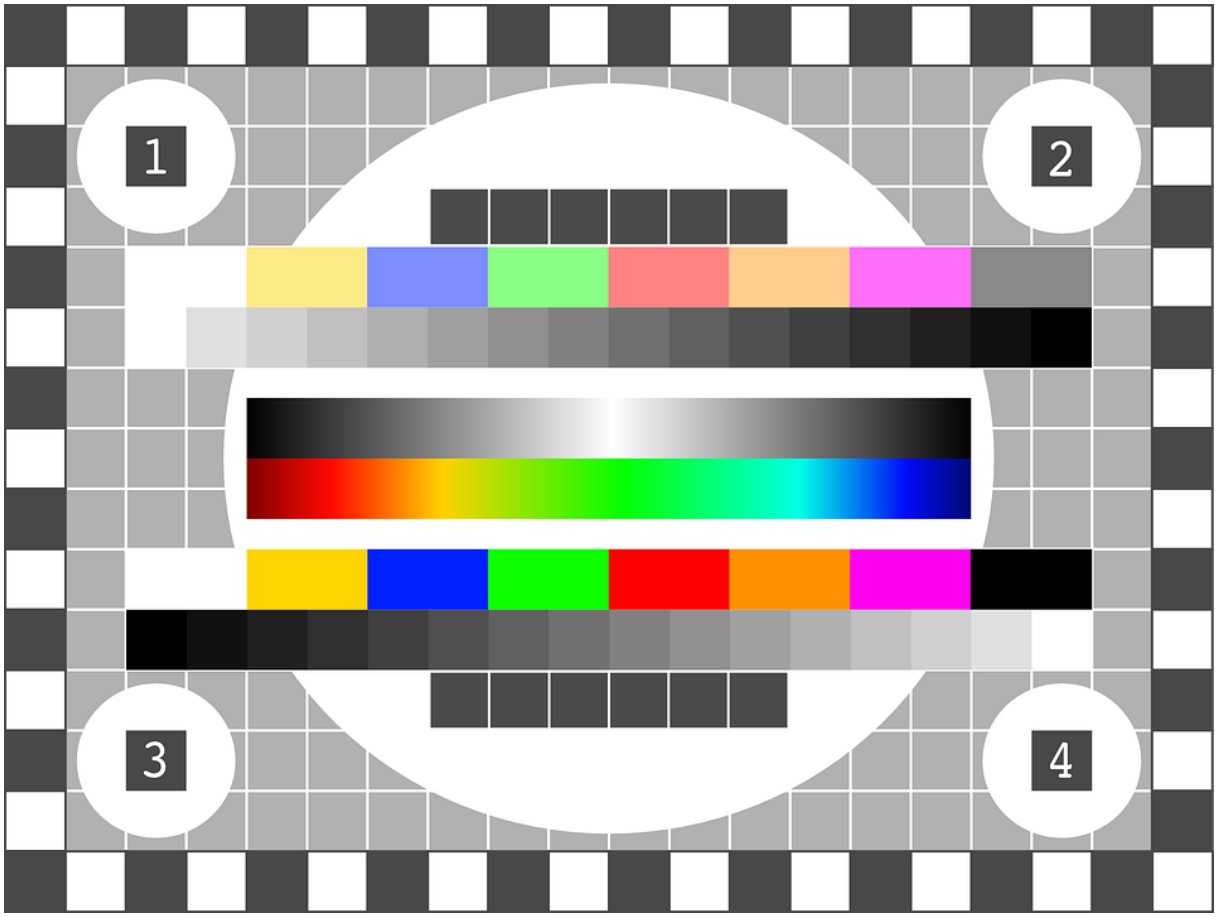

Figura 4: Flujo de trabajo con Class Recorder

3. Tecnologías, herramientas y metodologías

Antes de introducir las tecnologías que se han utilizado, es necesario comprender a grandes rasgos cómo esta estructurada la aplicación y cual es su arquitectura en general. No se ahondará en el diseño de la aplicación si no que se va a introducir a lo largo de este capítulo a grandes rasgos la arquitectura de la aplicación y las herramientas que hemos utilizado para gestionar el desarrollo, las pruebas y el por qué de cada decisión tecnológica. Podríamos dividir este capítulo en 6 subsecciones:

1. Metodología de desarrollo.
2. Lenguajes y frameworks utilizados para el desarrollo de la aplicación.
3. Organización de Repositorio y tecnologías para automatización de flujo de trabajo y CI/CD.
4. Entorno de desarrollo y de integración continua completamente dockerizado.
5. Tecnologías utilizadas para escribir esta memoria (Pandoc + Markdown)
6. Electron como plataforma final de producción.

3.1. Metodología de desarrollo

El proyecto ha sido desarrollado por mi, unido a la continua retroalimentación de mi tutor. Se podría considerar que se ha utilizado la metodología de Scrum[1], pero para los más puristas en cuanto a metodologías software no sería considerado como tal, ya que no se ha contado con el numero de personas suficientes para poder aplicar Scrum de la manera más eficiente posible, teniendo como Product Owner a mi tutor y yo mismo como Scrum Master y equipo de desarrollo. La propia guía de Scrum especifica que hay 3 roles diferentes: Product Owner, Equipo de desarrollo, y un Scrum Master. Además se dicta que el mínimo de personas necesarias para aplicar Scrum de la manera más eficaz es de 3 personas. Equipos de menos de 3 personas reduce la interacción y resulta en ganancias de productividad pequeñas. Sin embargo, sí que se han tenido reuniones pasadas ciertas semanas en el equipo para ver cómo ha ido avanzando el proyecto, retrospectivas, prototipos, integración, pruebas, etc. No obstante, como no aplicamos todas las reglas de Scrum, consideraremos que el desarrollo se está realizando con una metodología iterativa e incremental ágil tal y como se muestra en la Figura 5.

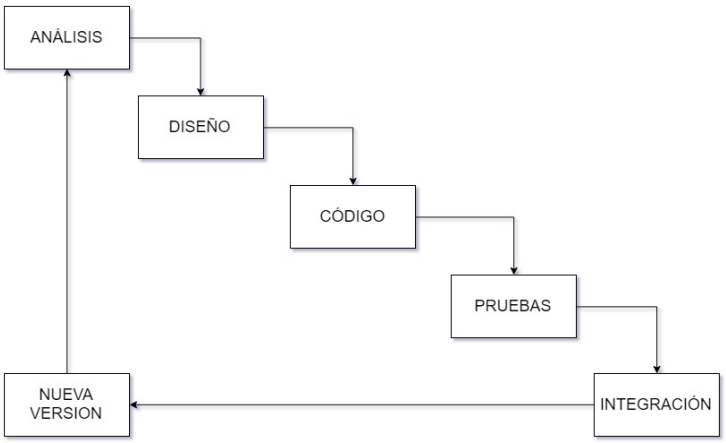

Figura 5: Ciclo iterativo e incremental

En este modelo, primero se realiza un análisis de los requisitos que se van a necesitar para cada iteración. Después del desarrollo de estos, se hacen pruebas y para finalizar se integra con el resto del sistema.

Cada 2 semanas se ha realizado una iteración donde se realizaron todos los pasos comentados anteriormente, donde todo se decidió qué historias de usuario eran más críticos e importantes y los bugs que se detectaron para resolverlos. En base a estas decisiones y utilizando herramientas online como Trello[1], se gestionó que tareas debían realizarse en cada iteración.

Un desarrollo iterativo e incremental ofrece varias ventajas con respecto a otras metodologías como puede ser el desarrollo en cascada. Una de las ventajas que ofrece es la entrega de software que se puede usar a mitad de desarrollo, mientras que en el modelo en cascada cada fase del proceso debe ser finalizada (firmada) para pasar a la siguiente fase. El desarrollo de software no es lineal y esto crea dificultades si se utiliza una metodología en cascada[2].

Cada cierto tiempo se han realizo releases. Utilizamos un formato de versiones semántico[3] del tipo X.Y.Z donde, X, Y y Z son números enteros mayores que 0.

X se corresponde a la versión mayor (cambios grandes que modifican parte o gran parte de la funcionalidad). Y se corresponde a la versión menor(pequeños cambios, corrección de bugs) y Z, que son micro versiones (parches, pequeños bugs críticos...). Se han incorporado además de las prácticas ágiles más comunes un sistema de integración y despliegue continuo.

[1]Es un tablero online donde se pueden crear, asignar y clasificar tareas, de tal modo que todo el equipo tiene una visión global del estado actual de desarrollo que se está creando.

3.2. Lenguajes y frameworks utilizados para el desarrollo de la aplicación.

La aplicación está formada por tres partes principales:

1. Backend (Java + Spring + Ffmpeg)
2. Frontend PC (Angular)
3. Android App (Ionic)

Su organización a nivel de arquitectura y diseño se comentan en más profundidad en la sección 4.2

3.2.1. Backend (Java + Spring Boot + Ffmpeg)

Como lenguaje de servidor hemos decidido utilizar Java junto con Spring Boot. Las razones de por qué hemos utilizado Java como lenguaje de programación en el backend son:

- Es el lenguaje con el que más experiencia he acumulado y más proyectos he realizado.
- Es un lenguaje fuertemente tipado y robusto, lo cual lo hace facilmente escalable y entendible por terceros programadores que deseen continuar el proyecto.
- Multiplataforma.

Para esta aplicación necesitabamos ofrecer una API RESTful para realizar una aplicación web frontend con Angular y una aplicación con Ionic. Se decició utilizar Spring Boot por las siguientes razones:

- Inyección de dependencias: Spring cuenta con un poderoso inyector de dependencias con el que podemos realizar nuestra aplicación utilizando las mejores prácticas para que sea escalable y testeable, separando la lógica en servicios independientes.
- Spring Data: Abstrae el acceso a bases de datos, de tal forma que no tenemos tanto que preocuparnos por el acceso y manejo de los mismos.
- Controladores Rest: Spring Boot ofrece la posibilidad de crear controladores Rest de una forma muy sencilla.

Figura 6: Java and Spring boot logo

Pero al servidor Java hay que añadirle la parte central y más importante de nuestra aplicación: ffmpeg.

Ffmpeg es una herramienta multiplataforma (posee binarios para cada plataforma) que nos permite convertir videos y audio de un formato a otro. Ffmpeg ofrece un CLI el cual utilizaremos en nuestra aplicación Java para realizar las grabaciones del escritorio, cortar los videos, juntar el audio del movil con el video grabado en el pc, etc.

Figura 7: FFmpeg logo

Haremos llamadas al sistema desde Java para la ejecución de Ffmpeg y controlaremos todo el proceso desde el mismo servidor. Se hablará de esto con mas detalle en la seccion 4.3

3.2.2. Frontend PC y Android App (Angular + Ionic)

Como estamos desarrollando un servidor RESTFul podemos crear dos proyectos SPA que serán los siguientes:

- Frontend PC: Angular + TypeScript
- Android App: Ionic 3 + Angular + TypeScript

En ambos proyectos utilizamos Angular como framework para el frontend (el cual utiliza TypeScript). ¿Por qué utilizar el framework Angular frente a otras opciones como React y Vue? En primer lugar, al ser un framework y no una librería, un framework tiene mucho más clara la construcción y organización de un proyecto. Además con TypeScript el código es mucho más mantenible debido a su tipado estático. Eso unido a la estructura de componentes y servicios que ofrece Angular, dan una base muy sólida para comenzar un proyecto que pueda crecer a largo plazo.

Por otro lado, Ionic es un framework que nos permite crear aplicaciones híbridas nativas para móbiles utilizando tecnologías web. Utilizando el framework de Angular y librerias que ofrece Ionic para utilizar elementos hardware del dispositivo móvil, podemos crear una aplicación móvil completa para la tarea que se quiere realizar. Ionic hace llamadas a las API's de cordova, que son un conjunto de librerías que permiten interactuar con el hardware móvil. Cordova ofrece una API javascript e Ionic encapsula este comportamiento mediante servicios que son facilmente utilizables dentro de Ionic.

Figura 8: De izquierda a derecha, logotipos de Cordova, Angular e Ionic

3.3. Organización de Repositorio y Task Managing

Este proyecto podría considerarse que está utilizando una organización monorepositiorio con tres partes principales que ya comentamos en la sección anterior.

Para organizar nuestro proyecto he utilizado Git como control de versiones y GitHub como repositorio remoto.

Para gestionar las builds y los scripts de test, hemos utilizado gulp, una librería de NodeJS, que nos permite automatizar tareas repetitivas tales como copiar ficheros al hacer builds, compilar en diferentes entornos, minificación de código, logs, etc. Gulp nos permite crear y organizar dichas tareas repetitivas para poder ejecutarlas desde una terminal o poder automatizar los builds y la ejecución de test de una forma organizada y relativamente sencilla.

Figura 9: Logotipo de gulp

Gulp permite crear tareas indivuales (que pueden ser builds, test, minificaciones) de tal modo que podemos encadenar tareas de una forma bastante sencilla y cómoda.

En la raíz del repositorio se encuentran los scripts más generales para compilar toda la arquitectura, pero gulp no es el único que hace todo el trabajo de automatización de builds y test. Cada proyecto individual utiliza sus propias herramientas de gestión y organiza también sus propios scripts. Por ejemplo:

- Backend: Maven
- Frontend: Angular-cli + npm + Node
- Ionic: Ionic-cli + npm + Node

Al final el proposito de gulp, es llamar a los scripts correspondientes para hacer las builds y los test de cada proyecto independiente de nuestro repositorio, y organizar todos los ficheros compilados en una unica carpeta `build/` en la que se encontrará nuestra aplicación final.

En definitiva, con gulp creamos un conjunto de scripts que nos abstrae en cierta medida de tener que utilizar angular-cli, ionic-cli, maven, etc. De esta forma se puede ejecutar, compilar o hacer test de nuestra aplicación con un solo comando desde la carpeta raíz. Para ejecutar gulp, necesitamos node y npm, por lo que en la carpeta raíz de nuestro proyecto, tenemos en un fichero `package.json` definidos todos los scripts necesarios para nuestro proyecto.

Por ejemplo para instalar las dependencias y ejecutar el servidor, solo es necesario ejecutar:

```
npm run install-dependencies
npm run dev:start-pc-server
```

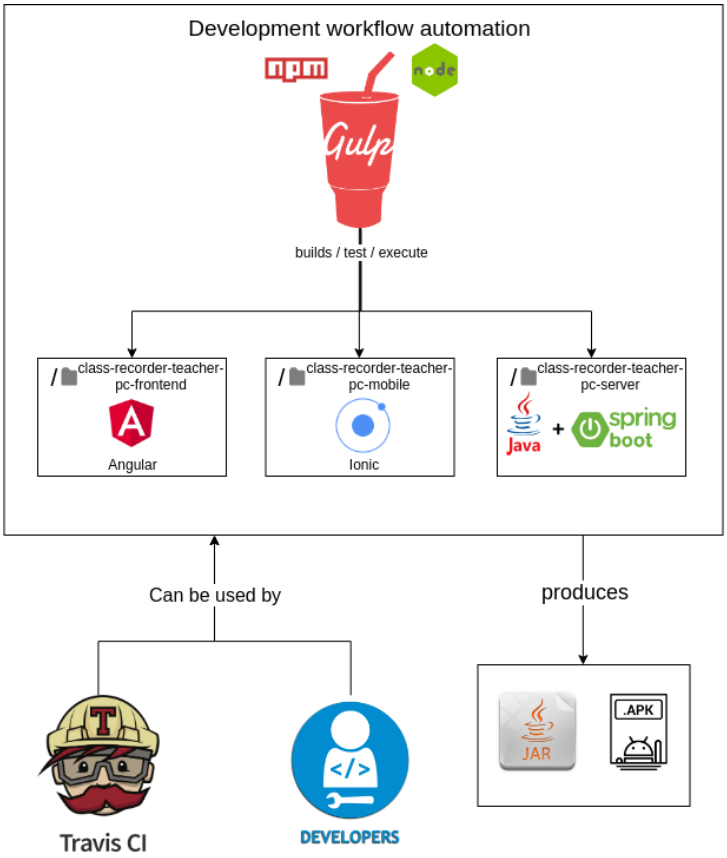

Figura 10: Automatización con gulp

En la Figura 10 se puede ver a grandes rasgos como gulp (ejecutado y gestionado con node y npm) actúa con los proyectos de nuestro repositorio. Todos los spripts definidos en el `package.json` pueden ser utilizados tanto por los desarrolladores como el entorno de CI/CD, agilizando el proceso de desarrollo.

En el Anexo 1 se puede ver además de una guía de configuración para el entorno de desarrollo, una lista con los scripts para compilar y ejecutar el proyecto.

Como sistema de CI/CD hemos utilizado travis. El sistema de CI/CD podría perfectamente ejecutarse en travis sin Docker, pero para tener un mayor control de las versiones que utilizamos, las configuraciones del sistema y tener un sistema en el que desarrollar homogéneo, decidimos utilizar Docker. En la siguiente sección explicaremos en más detalle cómo está montado el sistema de CI/CD junto con el uso de Docker.

3.4. Entorno de desarrollo y de integración continua completamente dockerizado.

Al desarrollar este proyecto me encontré con ciertas problemáticas. Una de ellas es la tediosidad de configurar todo el entorno para ponerse a desarrollar, y más teniendo en cuenta la utilización de ffmpeg y la instalación de Android SDK para poder crear la aplicación de Ionic. En los momentos en los que desarrollaba esta aplicación, tenía a mi disposición varios ordenadores, y tenía que configurarlos todos para poder desarrollar o corregir un simple bug. Si tenía un ordenador en la oficina, ¿por qué tenía que llevarme el portátil?. Si tenía un problema con el sistema, tenía que reconfigurar todo el entorno, y quería poder tener un sistema universal para poder desarrollar en cualquier ordenador que tuviera a mano.

Para esto se me presentaron varias posibles soluciones:

1. Crear una maquina virtual y configurar el entorno dentro para desarrollar

2. Configurar un servidor y desarrollar a través de él.

3. Dockerizar el entorno de desarrollo y levantar una infraestructura docker para desarrollar dentro de los contenedores.

Si no hubera conocido Docker, la primera opción hubiera sido la más factible, pero podía aprovechar un poco mejor los recursos con Docker. La segunda opción no era tan viable ya que por aquel entonces no existian herramientas lo suficientemente sólidas como para desarrollar desde un servidor. Eclipse Che era una opción pero no podía desarrollar Angular con él. Ahora existen servidores web como Code Server[2] u opciones del propio Visual Studio Code que te permiten desarrollar en remoto por ssh.[3] Las otras opciones consistian en instalar en un servidor un Linux con un sistema de ventanas, ya que necesitaba grabar el escritorio de Linux con ffmpeg, y desarrollar a través de una conexión por VNC no era una opción para mí.

Por eso me decante por la opción número 3. Decidí dockerizar todo lo necesario para desarrollar, incluidos programas de desarrollo como VSCode e IntelliJ, Android SDK, OpenJDK, ffmpeg y todo lo necesario en una misma imagen de docker. Esto además me permitió compartir dicha imagen con el entorno de integración, teniendo un sistema inmutable para desarrollar, testear y hacer las builds.

[2]Code Server: https://coder.com/

[3]Realizar desarrollo remoto con VSCode por ssh https://code.visualstudio.com/docs/remote/ssh

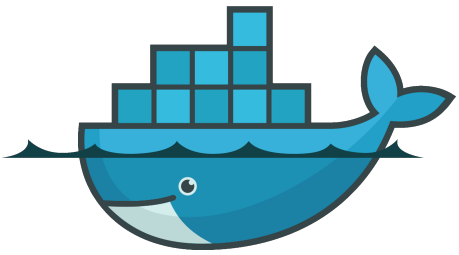

Figura 11: Logotipo de docker

La imagen docker comparte ciertos sockets de X11[4] y de pulseaudio[5] con el sistema operativo host e incorpora todas las IDE's y herramientas necesarias para desarrollar. En la siguiente imagen se puede ver un esquema visual de lo anteriormente explicado.

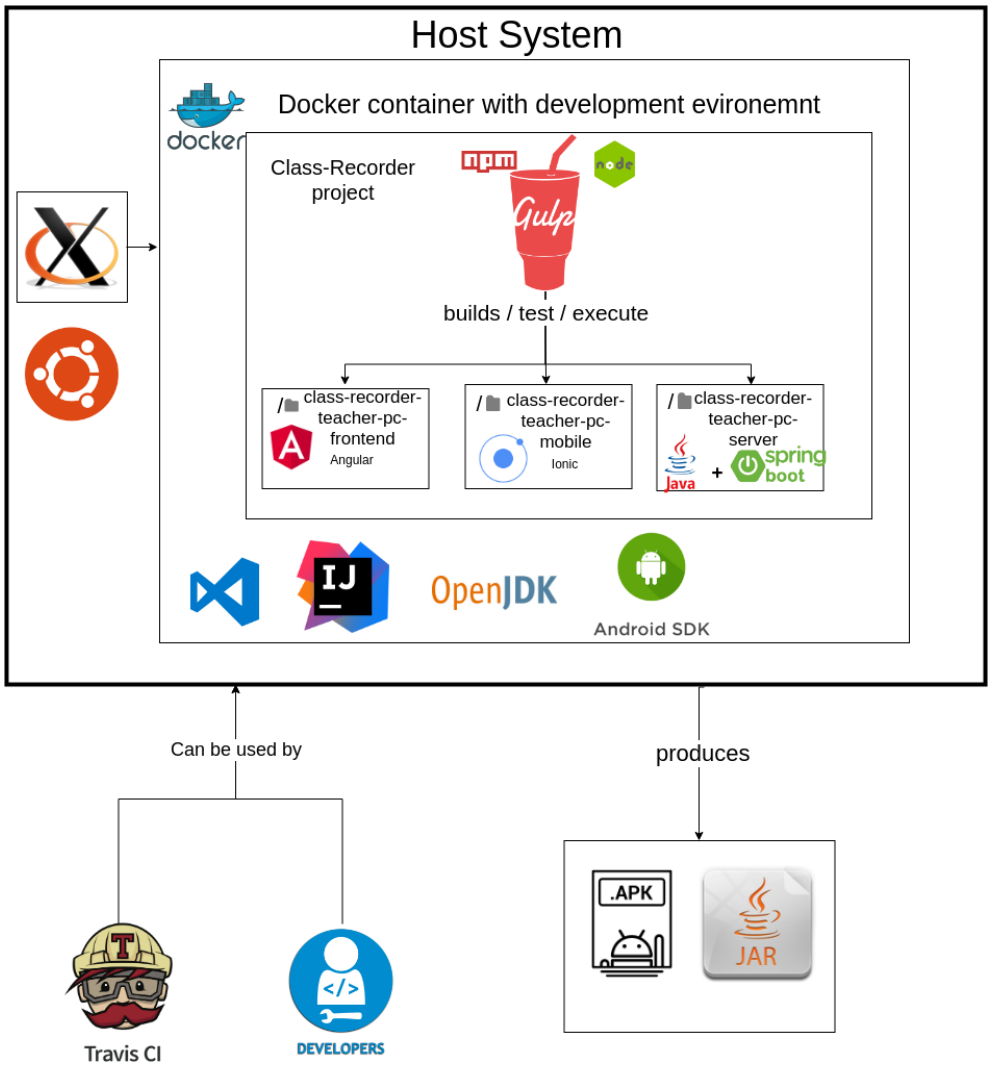

Figura 12: Entorno de desarrollo y entorno de CI/CD

De esta forma, podemos ponernos a desarrollar simplemente teniendo una máquina con ubuntu y docker instalado. En el Anexo 1 se puede ver como utilizar este entorno para desarrollar más

detenidamente. Pero para demostrar la comodidad que puede suponer está configuración, estos son los comandos necesarios para ponerse a desarrollar en cualquier máquina con Ubuntu.

```
git clone https://github.com/Class-Recorder/docker-class-recorder
cd docker-class-recorder/docker-runnables/crecorder-dev
./docker_run.sh
```

Con estás instrucciones levantamos la infraestructura necesaria para desarrollar. Una vez estan ejecutados los contenedores, el desarrolador debería ejecutar:

```
docker exec -it crecorder-dev_teacher-pc-server_1 /bin/bash
code class-recorder
```

Esto le abrirá un VSCode con todo listo para programar, hacer builds, testing, etc. No es necesario instalar absolutamente nada, el script `docker_run.sh` ejecuta un `docker compose` descargando las imagenes necesarias para desarrollar. El único inconveniente que puede tener esta solución es el tamaño final de la imagen (7.44 GB). Incluso se puede conectar un dispositivo Android y este sería detectado perfectamente ya que la imagen comparte los dispositivos usb y lleva incorporada una version de Android SDK y ADB.

He podido comprobar la utilidad de dockerizar el entorno de desarrollo. Cambiar entre ordenadores con Ubuntu no me requería de ningún tipo de instalación ni configuración previa, tan solo tener docker instalado, lo cual también me permitió cambiar entre entornos de desarrollo facilmente.

3.5. Tecnologías utilizadas para escribir esta memoria (Pandoc + Markdown)

La presente memoria, como se puede observar, es similar a un proyecto de Latex. Sin embargo la totalidad de su contenido está escrita en Markdown que es un lenguaje de marcado el cual permite dar formato a un texto. ¿Por qué Markdown?

- Posee una sintaxis sencilla y ágil.
- Hay multitud de herramientas web para posteriormente reutilizar el contenido escrito en markdown para realizar blogs.
- Realizar tablas, insertar imagenes, es realmente sencillo, y se puede utilizar con otras herramientas.
- No dependes de un formato especifico. No se utiliza ningún formato especial como XML para guardar los ficheros y el documento es editable desde cualquier editor de texto, por lo que se facilita la tarea de escribir en cualquier sitio y distribuir el conocimiento sin depender de un software.

Pero markdown de por sí, como lenguaje de marcado se quedaba un poco corto. Necesitabamos añadir figuras, numerarlas, crear una tabla de contenido, gestionar una bibliografía, etc. Entonces decidi utilizar Pandoc, que permite:

- Convertir documentos de texto.

- Podemos utilizar la sintaxis específica de Markdown para añadir figuras, utilizar una bibliografía, dar formato, y cambiar incluso entre formatos de citación, todo ello desde un único fichero de texto plano.

- Si nuestro documento final es un documento Latex, podemos utilizar sintaxis especifica de este para realizar ciertas características que el propio formato de Markdown no permite.

Así, con Pandoc y Markdown podemos hacer de forma muy sencilla cosas como:

- Añadir una citación poniendo su id junto con un `@`:

  `@id_citacion`

  Todas las referencias pueden ir organizadas en un fichero .bib y tan solo es necesario poner la referencia para que se autogenere en la bibliografía.

- Numerar títulos automáticamente y generar tablas de contenido con sintaxis especifica de markdown.

- Referencias con ids.

Esta memoria se encuentra en un repositorio de GitHub[4] y se puede ejecutar de la siguiente manera mediante docker:

```
git clone https://github.com/cruizba/TFG-Class-Recorder

cd TFG-Class-Recorder

docker run -it -v $(pwd):/home/userdocker/tfg --entrypoint "/usr/bin/node" \
-p 3000:3000 cruizba/markdown-to-latex-book server.js
```

Esto creara un servidor web en el puerto 3000 el cual contiene un index.html con el pdf de la salida en latex de todo el documento. Cada vez que se guarda el documento en markdown, se autogenera de nuevo y se recarga la página.

3.6. Electron como plataforma final de producción.

En un principio, este proyecto se iba a distribuir a través de una imagen docker. Pero surgieron varios problemas:

- La imagen docker dependía mucho del sistema, ya que necesita el servidor de x11 (el servidor de ventanas) debía ser el del sistema host, además del servidor de audio pulseaudio.

- Surgió un problema al compartir el servidor de x11 con docker al grabar videos con un parámetro específico.

- Las personas que no conocieran docker o que estuvieran fuera del ambito del desarrollo,

[4]TFG Class Recorder: https://github.com/cruizba/TFG-Class-Recorder

como profesores de otro tipo de asignaturas, no podrían utilizar este programa.

Entonces decidí usar electron para la aplicación Electron[5].

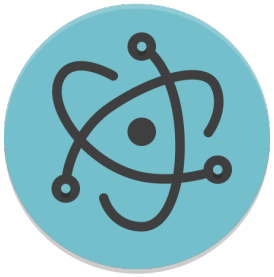

Figura 13: Logotipo de Electron

Electron es una tecnología que nos permite empaquetar aplicaciones web como si fueran aplicaciones de escritorio y ofrece una modelo que nos permite interactuar con el sistema operativo, los ficheros, las notificaciones... para poder crear aplicaciones nativas con tecnologías web.

Hablaremos más en detalle sobre como se empaqueta la aplicación en la sección 4.3

[5]Electron: https://electronjs.org/

4. Descripción informática

En la secciones anteriores hemos explicado las tecnologías que hemos utilizado y como hemos configurado el entorno de desarrollo, de CI y la metodología que estamos utilizando. En las siguientes secciones nos centraremos más en lo que es el desarrollo de la apliación en si misma.

Primero se hará una descripción de los requisitos funcionales y no funcionales, posteriormente se describirá de forma detallada la arquitectura y como se comunicarán los diferentes componentes de nuestra aplicación.

Posteriormente se ahondará a nivel de código y de diseño como se han implementado las partes más complejas de la aplicación.

4.1. Requisitos

Antes de comenzar las iteraciones y primeros prototipos del proyecto, es necesario crear una especificación de requisitos clara y concisa. Vamos a seguir algunas de las recomendaciones del estándar IEEE830[6] para ello. En un desarrollo iterativo e incremental ágil debemos tener muy en cuenta que los requisitos puedan ser modificables con frecuencia.

4.1.1. Requisitos funcionales

Los requisitos consisten en una serie de comportamientos o módulos que deben ser integrados en la aplicación y son los siguientes:

Requisito funcional 1

Grabar escritorio: La aplicación deberá ser capaz de grabar el escritorio junto con la voz de una forma fácil y sencilla con las características, con una configuración previa .

Requisito funcional 2

Controlar grabación del escritorio: La aplicación deberá ofrecer una interfaz de usuario para poder comenzar, pausar o parar la grabación.

Requisito funcional 3

Editar los cortes de los videos: La aplicación deberá generar durante la grabación unos metadatos con los que poder editar posteriormente los cortes.7

La grabación se realizará de tal forma que cada pausa no parará la grabación del video, si no que creara un metadato con la información del momento exacto en el que se realizó dicha pausa, para poder posteriormente editar el video a traves de una interfaz de usuario.

Requisito funcional 4

Subir videos a una plataforma en la nube: La aplicación deberá ofrecer un mecanismo facilmente

configurable con el cual poder realizar subidas automatizadas por ejemplo a Youtube, para poder ofrecer las clases a los alumnos rápidamente. Estos videos subidos deberán ser los videos previamente editados o grabados.

Requisito funcional 5

App móvil: La aplicación deberá ofrecer un software adicional móvil con el que poder controlar las grabaciones y grabar el audio desde el mismo. Así pues se distinguirían dos posibles casos de grabación:

- Si el profesor inicia la grabación desde la aplicación móvil, se grabará el audio procedente del micrófono del smartphone, y se grabará el contenido del escritorio de su ordenador principal.
- Si el profesor inicia la grabación desde el ordenador, se grabará el audio procedente del PC y se grabará también el escritorio del mismo.

Así mismo la aplicación móvil ofrecerá también la posibilidad de cambiar de diapositivas para que el profesor no tenga que recurrir a mandos o controladores externos.

Requisito funcional 6

Gestión de usuarios: La aplicación por el momento será de escritorio. Pero en un futuro deberá ofrecer la posibilidad de gestionar diferentes cuentas de usuario.

Requisito funcional 7

Gestion de cursos: La aplicación deberá ofrecer también un mecanismo para organizar los cursos que imparte.

Requisito funcional 8

Modificación de videos subidos: Se deberá poder modificar ciertos campos del video subido:

- Título.
- Descripción
- Tags

4.1.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que no representan partes de la lógica del problema que se quiere resolver, si no sobre su calidad de ejecución, entornos donde se debe poder ejecutar, y su posible extensibilidad escalabilidad. Los requisitos no funcionales son los siguientes:

Requisito no funcional 1

La aplicación debe ser ejecutable en los siguientes sistemas operativos:

- Windows 10
- Ubuntu

Requisito no funcional 2

La aplicación debe ser fácilmente configurable, sin necesidad de complejas configuraciones para su inicialización.

Requisito no funcional 3

La aplicación móvil debe poder ejecutarse en Android, pero ofrecer la posibilidad de extenderse a otros sistemas operativos.

4.2. Análisis y arquitectura

En la siguiente sección haremos un previo análisis de los requisitos anteriormente mencionados, para poder definir una arquitectura y definir los distintos componentes de nuestra aplicación final.

Previamente a la implementación es necesario hacer un análisis de algunos de los requisitos y abstraer dichas necesidades para poder plantear la arquitectura de nuestra aplicación. A continuación se describen algunas características que nos llevan a pensar en la arquitectura que se va a plantear:

1. Debe ser una aplicación de escritorio que permita editar los videos. (Requisito funcional 1, 2 y 3; Requisito no funcional 1, 2)

2. Aplicación móvil externa para controlar la aplicación y grabar el audio. (Requisito funcional 2 y 5; Requisito no funcional 3)

3. Debe poder grabar y controlar la grabación del el escritorio y el audio desde el móvil y desde la aplicación. (Requisito funcional 1, 2 y 5)

4. Control de usuarios y gestion de cursos. (Requisito funcional 6, 7 y 8).

5. Deben poder subirse los videos a una plataforma web. (Requisito funcional 4).

Con los 5 puntos podemos definir la arquitectura y además se puede ver como satisfacemos todos los requisitos.

A continuación, vamos a ir creando nuestra arquitectura a partir de los puntos mecionados anteriormente.

1. Para cumplir el punto 1 de la lista anterior, crearemos una aplicación web con su parte backend en Spring boot y Java; y su parte Frontend en Angular. Todo ello finalmente será empaquetado en un electron con ffmpeg y la JVM. Esta parte de la arquitectura se corresponde con el componente de color rojo de la Figura 14

2. Con respecto al punto 2, otra parte de la arquitectura se correspondería con una aplicación móvil desarrollada con Ionic. Esto se corresponde con el componente azul de la Figura 14

3. Para poder cumplir con los puntos 2 y 3 en lo respectivo a la grabación, el servidor, el frontend del ordenador personal en Angular y el móvil se comunicarán a través de websockets en red local, de tal modo que tanto el móvil como el servidor se comunicarán para controlar la grabación, llevando el servidor la gestion de la misma, y haciendo los frontend peticiones vía websocket para controlar la grabación. Se puede ver en la Figura 14 como tanto servidor como aplicación móvil están ambos están en la misma red local.

4. En cuanto al punto 4, utilizaremos por el momento una base de datos con persistencia de ficheros H2, pero en el futuro podría perfectamente sustituirse por otra base de datos gracias a la abstracción que nos aporta Spring Data y Hibernate.

5. Para subir los videos y cumplir con el punto 5, utilizaremos la API de Youtube.

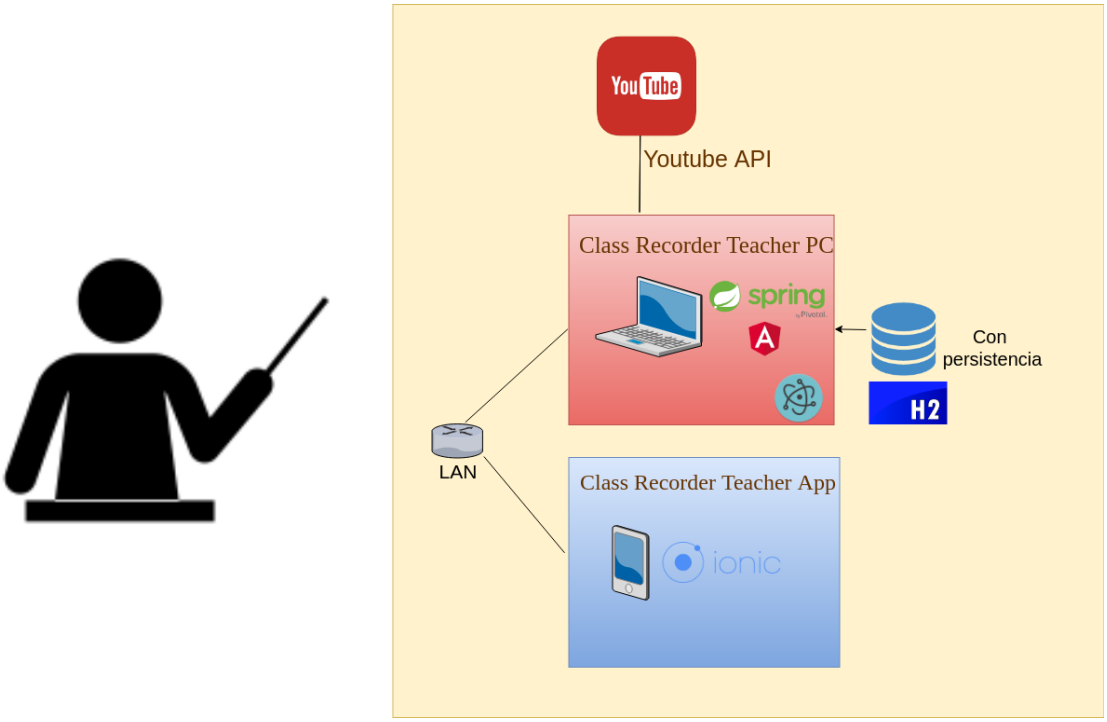

Figura 14: Arquitectura Class Recorder

Puede ser que el lector se haya dado cuenta, de que la arquitectura de desarrollo mostrada en la Figura 12 es bastante más compleja que la de la Figura 14. Esto se debe a que verdaderamente es mucho más sencillo el entorno de producción que el de desarrollo, porque podemos prescindir de muchas herramientas y porque compartir servicios del sistema operativo Host con docker complicaba mucho las cosas. Decidimos utilizar Electron en vez de Docker para distribuir la aplicación, por el hecho de que también era preferible que la aplicación fuera lo más independiente posible y porque evitaba posibles errores de configuración y facilitaba las cosas al usuario no experto.

Es necesario mencionar, que no utilizamos para nada la API de Electron para hacer operaciones con el sistema. Electron no es necesario en ningun momento para el desarrollo, todo el trabajo de ficheros y llamadas al sistema las realiza el servidor de Spring, y utilizamos Electron sólo para empaquetar la aplicación.

4.2.1. Diseño módulos del servidor. Ffmpeg Wrapper y Youtube.

En la siguiente sección explicaremos el diseño de dos módulos independientes que hemos creado para la aplicación en cuestión. Uno de ellos es un wrapper de Ffmpeg para poder realizar las grabaciones y el otro es una clase que utiliza la API de Youtube.

Ffmpeg Wrapper

En la parte de tecnologías (Sección 3.2.1) comentabamos el uso de Ffmpeg. Esta herramienta es multiplataforma pero si se quiere utilizar para la grabación del escritorio del PC y del audio del micrófono, difieren los comandos correspondientes en cada sistema operativo.

Para ello crearemos un módulo desacoplado de la lógica de nuestro servidor que nos permita lanzar comandos a ffmpeg de forma agnóstica al sistema operativo en el que se utilice, para poder posteriormente acloparlo a nuestro servidor. Vamos a crear un wrapper limitado a esta funcionalidad de Ffmpeg.

Ya existian previamente wrappers de Ffmpeg para Java, pero estaban algo desactualizados y además no ofrecían la funcionalidad suficiente como para poder ejecutar en distintos sistemas operativos diferentes.

Es por esto que decidí crear uno desde 0, sólo con la capacidad de poder realizar grabaciones de escritorio.

En la Figura 15 se pueden ver 2 clases que implementan las llamadas a los comandos concretos para los sistemas operativos Linux y Windows. Ambas clases implementan una interfaz `ICommand` la cual es instanciada en la clase `FfmpegWrapper`que se encargará de instanciar la clase concreta dependiendo del sistema operativo en el que se encuentre. Las clases que implementan ICommand deben tener los siguientes métodos y todos ellos lanzan un comando de Ffmpeg:

- `executeFfmpegVideoAndSound(): Process`: Ejecuta un comando para grabar el video y el audio del escritorio.
- `executeFfmpegVideo(): Process`: Ejecuta un comando para grabar sólo video.
- `executeFfmpegMergeVideoAndAudio: Process`: Ejecuta un comando el cual permite juntar un audio y un video. Esta parte es principalmente para juntar el audio del móvil y el video grabado del escritorio.
- `executeFfmpegCutVideo: Process:` Ejecuta un comando el cual permite cortar un video en diferentes secciones, a partir de un fichero con metadatos de cortes del mismo.
- `executeMergeVideos():Process`: Junta varios videos en uno único.

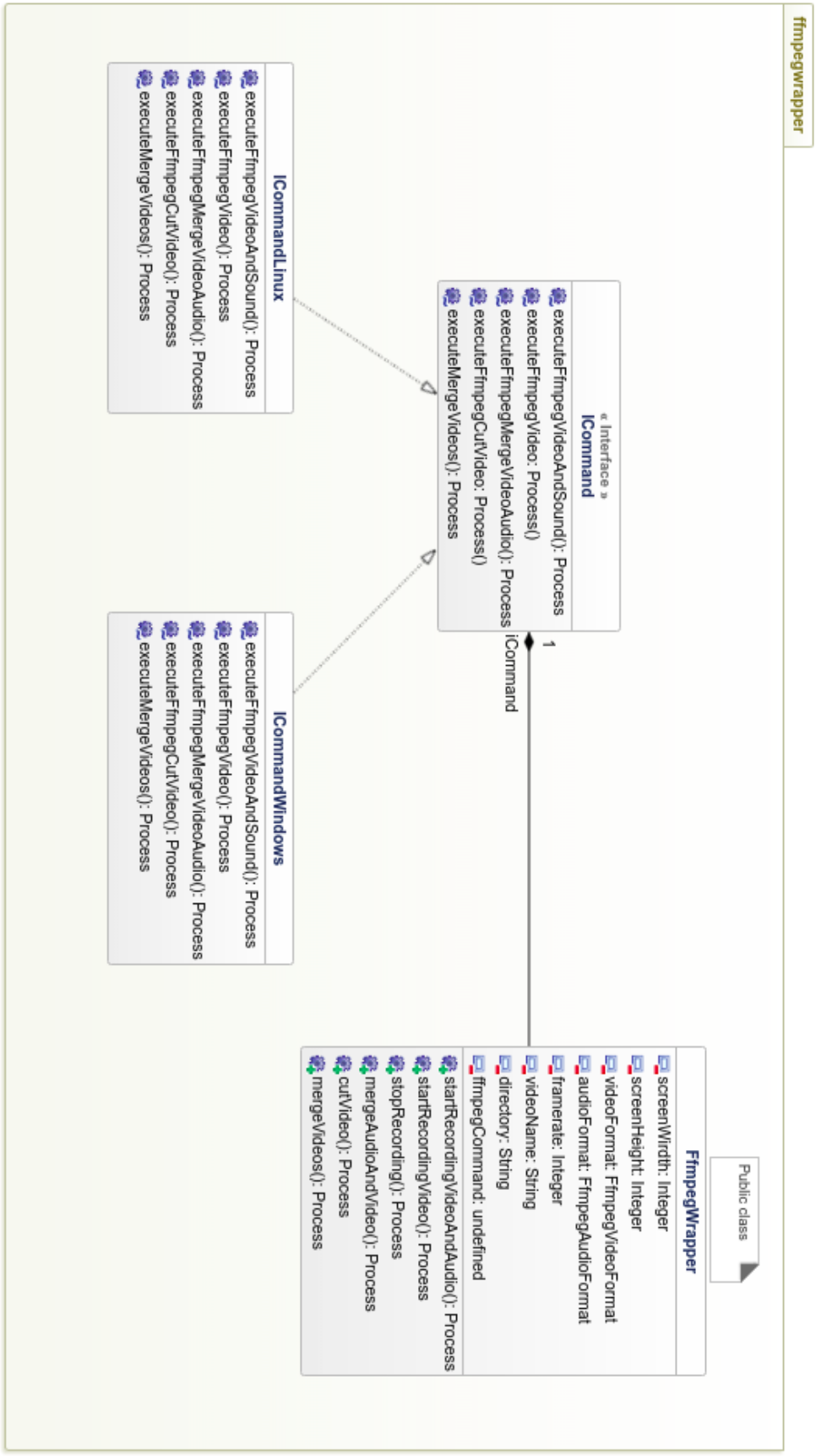

Figura 15: Diagrama de clases del wrapper de ffmpeg para Class Recorder

FfmpegWrapper por otro lado se encarga de gestionar la grabación y los procesos lanzados por ICommand. FfmpegWrapper se encarga de matar el proceso en caso de ser necesario, de saber si está pausado, grabando o parado, de recibir los parametros para ejecutar una grabación, y de gestionar los cortes y de utilizar una implementación de `ICommand`dependiendo del sistema operativo en el que se encuentre.

FfmpegWrapper también contiene una lista de observadores a los cuales envia la salida estandar de los comandos lanzados por ICommand, para poder enviar los resultados al frontend y gestionar los logs. Instanciar este módulo es realmente sencillo desde nuestra aplicación de Spring.

Youtube API

También necesitamos crear una clase que gestione las llamadas a la API de youtube para poder enviar los videos grabados con la aplicación. Esta clase contiene los siguientes métodos públicos:

- `getOAuthUrl(): string`: Devuelve la url para hacer login en Youtube y poder subir el video.
- `uploadVideo(YoutubeVideoInfo ytVideoInfo): Video`: Sube el video que se le pasa como argumento a youtube.
- `updateVideo(String youtubeId, YoutubeVideoInfo ytVideoInfo): Video`:Actualiza la información de un video de youtube en función de su Id.
- `deleteVideo(String youtubeId): Video`: Borra un video de youtube en función de su id.
- `getState():YoutubeUploaderState`: Devuelve el estado de la subida.
- `getProgress(): double`: Devuelve, si está subiendo un video, el porcentaje subido.

4.2.2. Diagrama Entidad-Relacion de los datos.

Las siguientes entidades son utilizadas para realizar persistencia en la base de datos. Es necesario que nuestro programa sea multiusuario (para el caso de que en un futuro se pueda escalar), y que ademas gestione los cursos de cada profesor. No solo eso sino que deberá guardar también los videos de Youtube que se han guardado en la plataforma.

El diagrama entidad relación es el de la Figura 16 y son las siguientes entidades.

- YoutubeVideo: Representa la información de un video subido a youtube. Contiene tags. Un video pertence a un curso.
- Tag: Palabras clave de un video.
- User: Representa los usuarios de la plataforma. Pueden ser de dos tipos, cuyo tipo es diferenciado segun su `user_type`. Los profesores tienen en el campo user_type, el valor `teachery` los estudiantes el valor `student`. Veremos en la sección 4.3 como este valor discriminatorio nos permite implementar herencias con JPA. Los profesores podrán tener

un conjunto de cursos creados y los estudiantes un conjunto de cursos a los que están suscritos.

- Course: Representa la información de un curso. Un curso tiene varios videos.

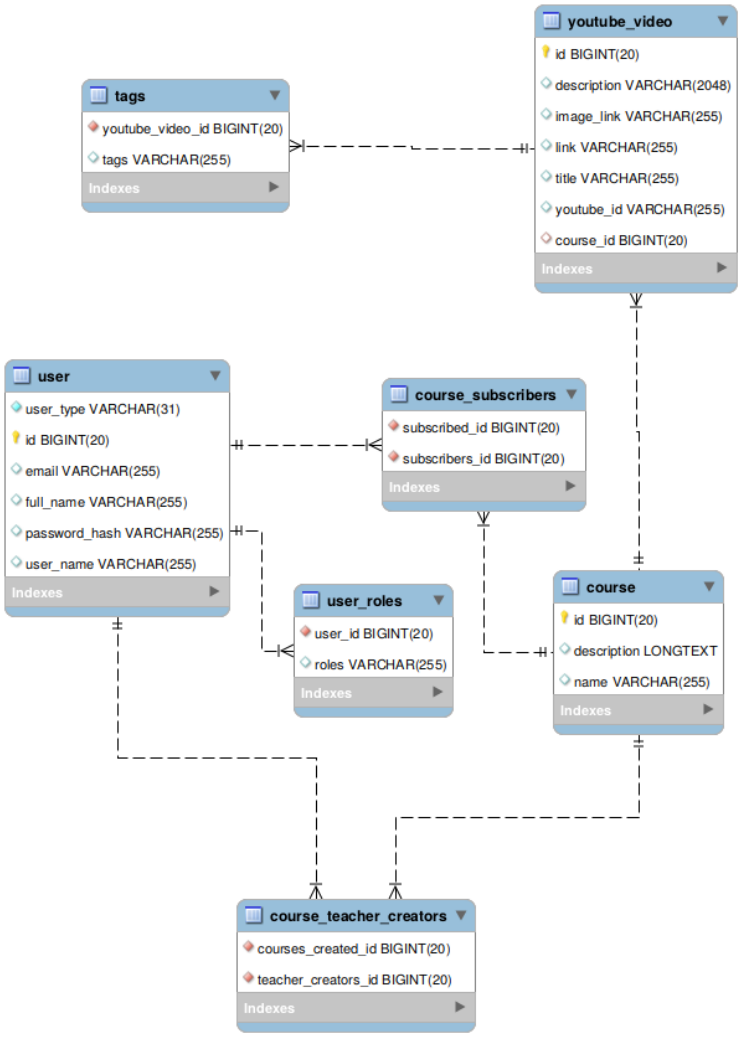

Figura 16: Diagrama entidad relacion de la base de datos

Los videos grabados sin subir a la plataforma de youtube, son representados por ficheros en una carpeta a elegir por el usuario.

4.2.3. Diagrama de secuencia de una grabación.

En esta sección vamos a explicar dos posibles casos de uso en los que se puede utilizar esta aplicación y como se realiza esto secuencialmente para cada uno de los dos casos. Si el profesor quiere grabar el audio del micrófono del PC deberá comenzar la grabación desde el mismo PC, y del mismo modo si quiere grabar el escritorio del ordenador y grabar el audio desde el móvil deberá iniciar la grabación desde la aplicación móvil.

El diagrama de secuencia del primer caso es el siguiente.

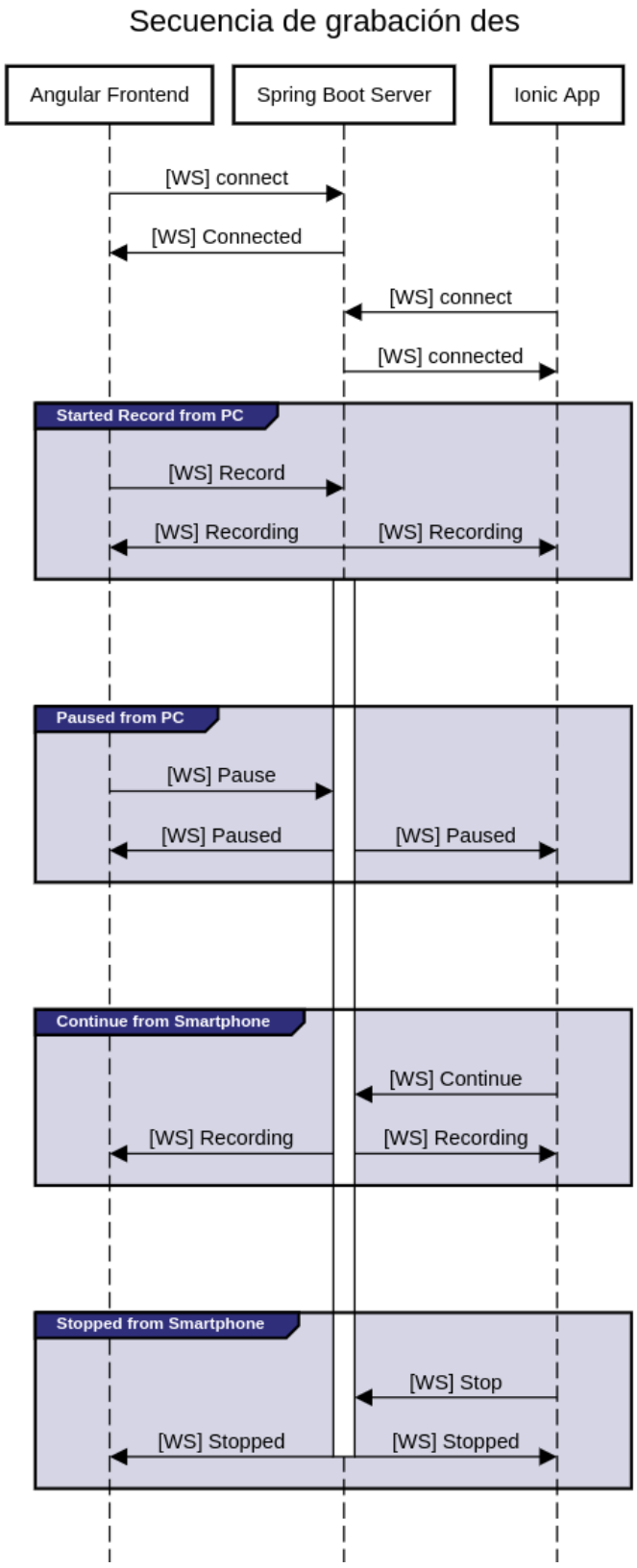

Figura 17: Diagrama de secuencia de grabación grabando desde PC

Todas las peticiones se realizan a través de WebSocket. Tanto la aplicación Frontend de Angular como la aplicación Ionic se conectan al servidor de Spring boot y reciben el estado de la grabación, pudiendose controlar desde ambos dispositivos al mismo tiempo.

En la Figura 17 se realizan los siguientes pasos:

1. Se conecta primero el PC y despues el móvil aunque el orden es irrelevante.

2. Se comienza la grabación desde el ordenador, para grabar el micrófono del PC y ambos dispositivos se les notifica del estado de la grabación.

3. Se pausa la grabación desde la aplicación móvil e igualmente se actualiza el estado en ambos dispositivos.

4. Finalmente se para la grabación desde el móvil.

Por otro lado en el segundo caso, si el usuario quiere grabar el audio del móvil, el diagrama de secuencia es el de la Figura 18 y las siguientes:

1. Se conecta, al igual que en el caso anterior, el PC y el móvil.

2. Se comienza la grabación desde la aplicación móvil, y el servidor notifica del estado de la grabación, indicando que se ha realizado desde la aplicación.

3. Se pausa la grabación desde el móvil

4. Se continua desde el PC.

5. Se para la grabación desde el móvil (aunque se podría hacer desde el ordenador).

6. La aplicación móvil, al saber que la grabación la inició él, envía el audio grabado a traves de una petición http Multipart.

7. Al terminar el móvil envia una petición http al servidor pidiendo mergear el video y el audio previamente enviado.

8. Notifica a los dispositivos que el proceso a terminado.

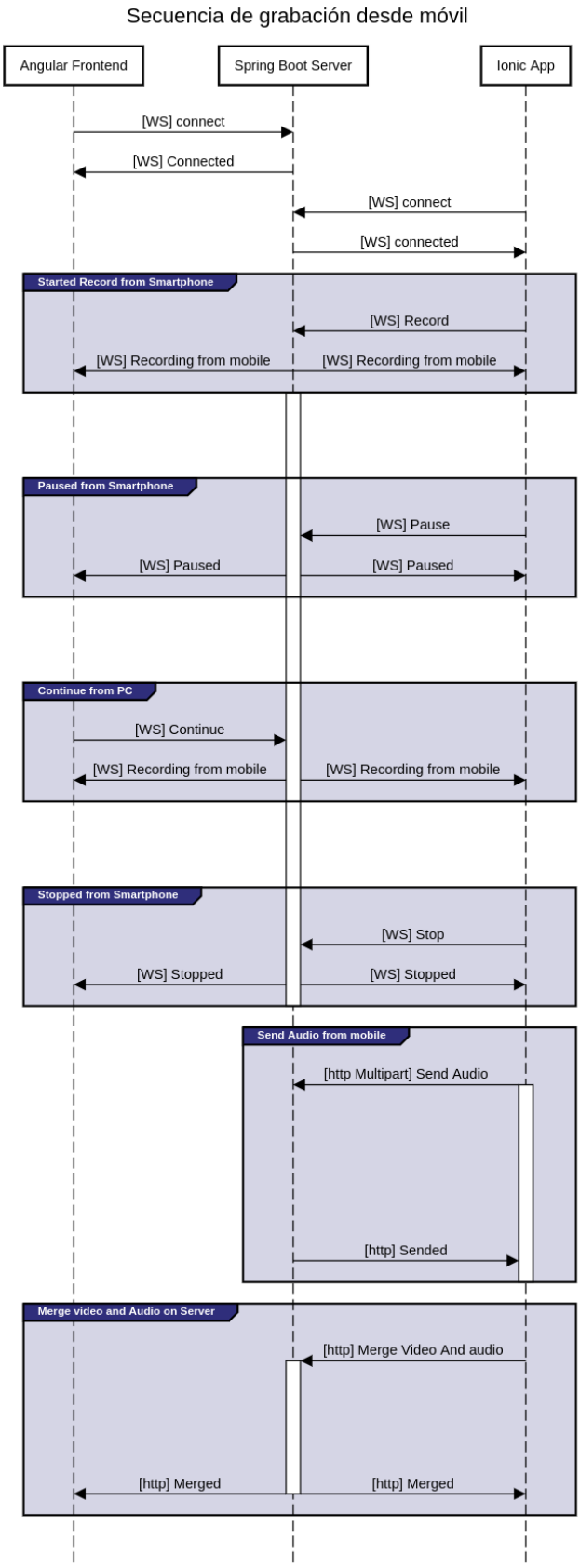

Figura 18: Diagrama de secuencia de grabación desde móvil

4.3. Diseño e implementación

En esta seccion se explicará con más detalle como se han implementado los modulos introducidos en la sección 4.2.1, como instanciarlos sin Spring Boot, cómo se han instanciado a modo de Servicios en Spring Boot, como es el fichero de metadatos de los cortes, los comandos de ffmpeg utilizados, además de algunas muestras de la interfaz de usuario, las pruebas automáticas que se han realizado y los pasos que sigue el sistema de CI/CD.

4.3.1. Implementación Wrapper Ffmpeg

Para simplificar el proceso de grabación, se ha aplicado el principio de responsabilidad única, creando para cada sistema operativo, una clase muy simple que simplemente ejecuta comandos Ffmpeg a partir de unos parámetros y la clase `FfmpegWrapper.java` controla el proceso de grabación de forma agnóstica al sistema. Para entender como funciona el Wrapper de Ffmpeg tenemos que echar un vistazo al constructor de la clase `FfmpegWrapper.java`.

```
1  public FfmpegWrapper(Path ffmpegOutput,
2                       String x11device,
3                       String ffmpegDirectory)
4                       throws OperationNotSupportedException, IOException {
5      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
6      this.os = System.getProperty("os.name");
7      log.info("Operaiting system: " + this.os);
8
9      this.screenWidth = new Double(screenSize.getWidth()).intValue();
10     this.screenHeight = new Double(screenSize.getHeight()).intValue();
11     this.x11device = x11device;
12     this.videoContainerFormat = null;
13     this.videoName = null;
14     this.directory = null;
15     this.recording = false;
16     this.observers = new ArrayList<>();
17     this.observers.add(new FfmpegWrapperLogger());
18
19     if (this.os.equals("Linux")) {
20         this.ffmpegCommand = new ICommandLinux(ffmpegOutput,
21                                                x11device,
22                                               ffmpegDirectory);
23     }
24     else if(this.os.contains("Windows")) {
25         this.ffmpegCommand = new ICommandWindows(ffmpegDirectory);
26     }
27 }
```

Como se puede observar en las líneas 9 y 10 cogemos las dimensiones de la pantalla (independientemente del sistema operativo, Java ofrece un método multiplataforma para ello), y en las lineas 19-26 instanciamos el objeto `ICommand` correspondiente para el sistema operativo en ejecución.

Instanciar esta clase es realmente sencillo. Solo es necesario introducir las siguientes instrucciones:

```
FfmpegWrapper ffmpeg = new FfmpegWrapper();

ffmpeg.setAudioFormat(FfmpegAudioFormat.libvorbis)
  .setVideoFormat(FfmpegVideoFormat.libx264)
  .setContainerVideoFormat(FfmpegContainerFormat.mkv)
  .setFrameRate(60)
  .setDirectory("videos")
  .setVideoName("test");

ffmpeg.startRecordingVideoAndAudio();
```

Como vemos, solo necesitamos configurar una serie de parámetros al principio, especificar el formato de audio y de video que se va a utilizar, el formato de cotenedor, el framerate, el directorio donde guardar los videos y el nombre del video. Posteriormente podríamos ejecutar la última instrucción y ffmpeg comenzaría a grabar.

¿Cuales son los comandos que lanza cada método? Las implementaciones de los comandos concretos se encuentran en las clases: `ICommandWindows.java` y `ICommandLinux.java` Vamos a explicar los tres comandos más importantes implementados de la clase `ICommandLinux.java`

Grabar video con audio: Se utiliza el método `executeFfmpegVideoAndSound` que implementa el siguiente comando para los videos `mkv`. Este comando concreto es un ejemplo de un video grabado desde la aplicación en Linux:

```
ffmpeg -f x11grab -framerate 60 -s 1366x768 -i :0 \
-vcodec h264 -thread_queue_size 20480 -f alsa -i default \
-pix_fmt yuv420p -acodec mp3 -preset ultrafast -crf 30 \
/home/user/test.mkv
```

Los parámetros del siguiente comando son los siguientes:

- `-f`: Formato de entrada. En este caso especificamos que el formato es `x11grab` ya que proviene del servidor de ventanas x11 en Linux. Para el audio en Linux especificamos alsa.
- `-framerate`: Frames por segundos capturados. Esta opción es parametrizable desde el método.
- `-s`: Resolución de la pantalla. Este parámetro es inicializado por el sistema y no es necesario que el usuario lo introduzca.
- `-i`: Input de video. Especificamos `:0`para que obtenga la salida por defecto de video del servidor de ventanas de x11. En el caso del audio ponemos `default` para que coja el micrófono configurado por defecto en el sistema.
- `-vcodec`: Codec de video con el que se guardará el video. En este caso h264.
- `-acodec`: Formato de audio. En este caso mp3.

- `-thread_queue_size`: Especifica el tamaño de una cola de procesamiento para la conversión. Especificando un tamaño grande para la cola estamos incrementando la cantidad de paquetes que pueden esperar para ser codificados al formato especificado. En este caso hemos puesto un valor por defecto alto de 20480 bytes.

- `pix_fmt`: Especifica el formato de pixeles. En este caso hemos utilizado `yuv420p` que es un formato de píxeles que permite reducir significativamente el tamaño en favor de píxeles más iluminados, pero reduciendo la calidad.

- `-preset`: Esta opcion es específica del codec h264. Cada opción posible de este parámetro representa un conjunto de opciones para codificar. En nuestro caso hemos utilizado `ultrafast`, el cual produce un video de mayor tamaño pero la compresión del video requiere de menos procesamiento.

- `-crf` Esta opción también es especifica de h264. Los rangos posibles van desde el `0` (que representa una codificación sin perdidas pero que requiere un mayor procesamiento) al `51` (que representa la peor calidad posible pero con una velocidad de procesamiento necesaria más baja). Hemos optado por un valor más o menos equilibrado, que en nuestro caso es `30`.

- Nombre del fichero final. Este parámetro también es parametrizable desde el método.

Tras muchas pruebas, este fue el comando que mejores resultados se obtuvieron midiendo calidad y procesamiento, aunque en un futuro dichos parámetros pueden ser configurables.

En el caso de Windows el comando es similar, solo cambian los parametros de entrada y de formato.

Cortar video: Se utiliza el método `executeFfmpegCutVideo()`. Al igual que el comando anterior, este comando es un ejemplo concreto ejecutado por la aplicación:

```
ffmpeg -i <directory-videos>/video_to_cut.mkv \
-vcodec copy -acodec copy \
-ss 00:00:00 -to 00:00:15 \
<directory-temp-class-recorder>/temp/out0.mkv -acodec copy \
-ss 00:00:29 -to 00:00:34 \
<directory-temp-class-recorder>/temp/out1.mkv -acodec copy \
-ss 00:00:36 -to 00:00:37 \
<directory-temp-class-recorder>/temp/out2.mkv
```

Este comando a grandes rasgos es similar al anterior, lo unico distinto es la entrada que es el video a cortar, y que por cada segmento de video que se quiere cortar, se introducen los siguientes parámetros:

- `-acodec copy`: Utiliza como codec de audio, el mismo codec que el video de entrada.

- `-vcodec copy`: Utiliza como codec de video el mismo codec que el video de entrada

- `-ss`: segmento de video que se quiere cortar

Los segmentos de video son proporcionados por un parametro en el método que contiene una

lista de cortes, y que es representado por un fichero a modo de metadatos en la aplicación. El formato de los cortes es explicado en la sección 4.3.4. Los videos cortados seran creados en una carpeta especificada.

Juntar videos: Se utiliza el método `executeMergeVideos()`. Ejemplo del comando:

```
ffmpeg -f concat -i <directory-temp-class-recorder>/temp/files.txt \
-c copy <directory-videos>/test_cutted.mkv
```

A partir de un fichero de texto que se habrá creado con anterioridad en una carpeta temporal especificada se generará un video con los videos unidos juntados en uno único.

4.3.2. Implementación módulo Youtube

Para implementar la subida a videos de youtube se ha decidido utilizar una librería que haga las llamadas http pertintentes, ya que para esta tarea especifica, si existían librerías apropiadas para subir vídeos a Youtube. La librería que hemos utilizado ha sido las librerías de youtube de Google.

Así pues hemos seguido un ejemplo de la api[6] y lo hemos adaptado a nuestras necesidades. Esta api necesita de una clave secreta de aplicación que hemos generado a traves de las herramientas de desarrollo de Google. Al subir un video generamos una url que el usuario visita para dar permisos a nuestra a nuestra api de Youtube para poder subir videos a la cuenta del usuario de Google.

Lo unico necesario para instanciar esta clase y subir un vídeo son las siguientes instrucciones:

```
1 YoutubeUploader youtube = new YoutubeUploader();
2
3 YoutubeVideoInfo videoInfo = new YoutubeVideoInfo();
4   videoInfo.setVideoTitle("Video de prueba");
5   videoInfo.setDescription("Descripcion del video");
6   videoInfo.setPrivacyStatus("unlisted");
7   videoInfo.setVideoPath("videos/video_de_prueba1.mkv");
8
9 youtube.uploadVideo(videoInfo);
```

Como vemos la forma de inicializar el módulo es bastante similar al del wrapper de ffmpeg. Simplemente le pasamos los parametros necesarios para subir el video y ejecutamos la última instrucción.

4.3.3. Inyección de los módulos como servicios en Spring Boot.

Los dos módulos descritos con anterioridad, de por sí, son independientes de cualquier framework y podrían ser utilizados en cualquier proyecto de Java 8. Pero en nuestro caso, tenemos que proporcionar la funcionalidad de estos módulos como servicios en una aplicación de Spring Boot.

[6] https://developers.google.com/youtube/v3/code_samples/java?hl=es-419

Para ello tenemos que hacer uso de la inyección de dependencias de Spring y la configuración de los Beans.

Lo que hemos hecho ha sido crear para cada uno de los dos módulos un `@Service` que contiene una instancia ya creada del propio módulo en el mismo servicio. En la Figura 19 se puede ver a grandes rasgos la estructura de la aplicación en Spring Boot.

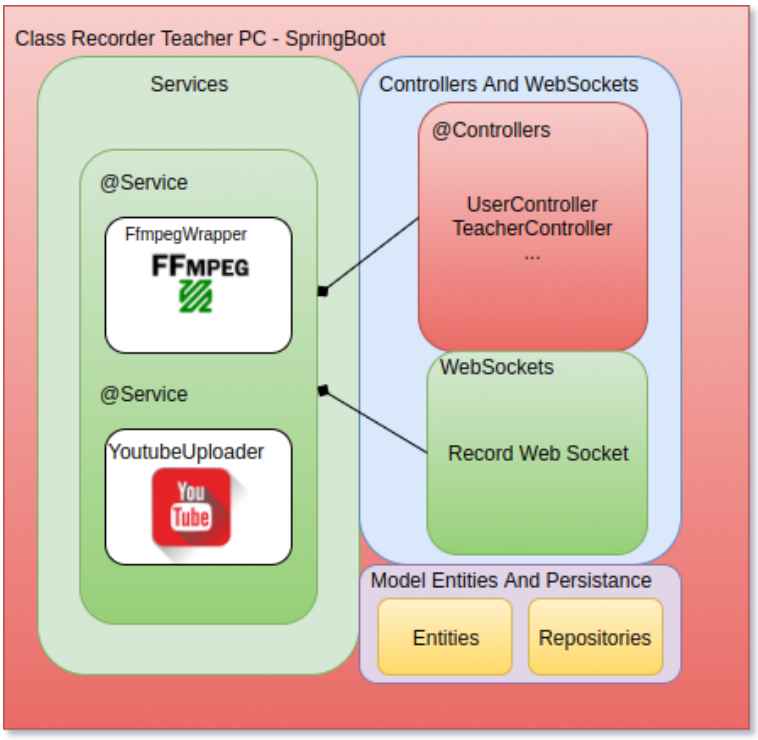

Figura 19: Estructura servidor Spring boot

Como podemos ver tanto los controladores como los websockets harán uso de los Servicios que a su vez integrarán los dos módulos mencionados anteriormente.

Para que Spring se encargue de inicializar nuestros módulos creamos ambos servicios de esta forma:

```
1  @Service
2  public class FfmpegService {
3
4      private FfmpegWrapper ffmpegWrapper;
5
6      public FfmpegService(Path ffmpegOutput, String x11device, String ffmpegDirectory) throws
7          this.ffmpegWrapper = new FfmpegWrapper(ffmpegOutput, x11device, ffmpegDirectory);
8      }
9
10     public FfmpegService setContainerVideoFormat(FfmpegContainerFormat videoFormat) {
11         ffmpegWrapper.setVideoContainerFormat(videoFormat);
12         return this;
13     }
```

```
14
15      public FfmpegService setFrameRate(int frameRate) {
16          ffmpegWrapper.setFrameRate(frameRate);
17          return this;
18      }
19      ...
```

Lo que hacemos es inicializar el módulo en el constructor y crear un método para cada uno de los métodos existentes en ese módulo. De esta forma podemos utilizar los módulos a modo de servicios en los controladores y websockets. La implementación del servicio de Youtube es similar en concepto. Inicializamos ambos servicios con la notación `@Bean`. Los Beans son objetos que maneja el contenedor de Spring y los declaramos ambos de la siguiente manera:

```
1  @Bean
2  @Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
3  public YoutubeService youtubeService() throws YoutubeApiException {
4      return new YoutubeService(properties);
5  }
6
7  @Bean
8  @Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
9  public FfmpegService ffmpegService() throws OperationNotSupportedException,
10                                             IOException {
11     return new FfmpegService(classRecProperties.getOutputFfmpeg(),
12     System.getenv("DISPLAY"), classRecProperties.getFfmpegDirectory());
13 }
```

Ambos los declaramos como Singleton ya que el servidor es una única instancia que es ejecutada en el ordenador del profesor. Además el servicio de grabación está preparado para realizar las grabaciones en una sola máquina ya que es su cometido, y la mejor opción es tener una única instancia del servicio de grabación para controlar en todo momento el proceso de manera adecuada.

4.3.4. Websocket controlando el wrapper de Ffmpeg y metadatos para los cortes.

Como sabemos de la sección 4.2.3 donde explicabamos la comunciación de los distintos componentes de la arquitectura de nuestra aplicación, necesitamos que tanto la aplicación web ejecutada desde el ordenador del profesor, como la aplicación móvil, sean capaces de iniciar, parar y pausar las grabaciones. Para ello debemos echar mano de los websockets. Los websockets permiten una comunicación bidireccional continua full-duplex, entre uno o varios clientes y un servidor. De esta forma, podemos notificar de eventos a todos los dispositivos que se conecten a nuestro servidor a través de websockets, pudiendo notificar a todas las aplicaciones el estado de la grabación.

Una pregunta que puede tener el lector ahora es, ¿Por qué Websockets? Con los WebSockets evitamos por ejemplo hacer *polling* de peticiones al servidor para saber cierto dato, el servidor puede enviar información a los clientes sin necesidad de que el dispositivo cliente realice una petición http.

Para esta aplicación necesitaremos implementar tres websockets, que serán los siguientes:

- Grabación: Todos los controles de grabación serán controlados a traves de un websocket.
- Procesamiento de la información. Crearemos un websocket para enviar la sálida de Ffmpeg al Frontend, para corroborar su correcto funcionamiento.
- Progreso subida de Youtube: La información acerca del progreso durante una subida de un video a Youtube, también será enviada por websockets.

Para ello primero tenemos que hacer que Spring tenga habilitada una ruta por la cual conectarse a este Websocket. Para ello creamos una clase aparte con la notación `@Configuration` para que Spring sepa que es una clase de configuración:

```
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

  @Autowired
  WebSocketRecordHandler recordPCHandler;

  @Autowired
  WebSocketProcessHandler processHandler;

  @Autowired
  WebSocketYoutubeUpload youtubeProgress;

  @Override
  public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
      registry.addHandler(recordPCHandler, "/recordpc").setAllowedOrigins("*");
      registry.addHandler(processHandler, "/process/info").setAllowedOrigins("*");
      registry.addHandler(youtubeProgress, "/youtube/progress").setAllowedOrigins("*");
  }
}
```

En esta clase inyectamos todas las clases que implementan la lógica de los Websockets y se los asignamos a la ruta específica para cada uno en el método `registerWebSocketHandlers()`. A continuación se va a explicar la implementación del WebSocket más importante de la aplicación: `WebSocketRecordHandler`

WebSocket grabación

En la parte del backend para crear el WebSocket de grabación hemos creado la siguiente clase:

```
@Component
public class WebSocketRecordHandler extends TextWebSocketHandler {

  // Variable to save sessions connected
  private List<WebSocketSession> sessions = new ArrayList<>();

  //Recording Variables
  private TimeCounter timeCounter = new TimeCounter();
```

```
9   private TimeCounterPause timeCounterforFrontends = new TimeCounterPause();
10  private boolean onPause = false;
11  private boolean mobileRecording;
12  private ArrayList<Cut> cuts = new ArrayList<>();
13  private String previousTime;
14  private String actualTime;
15  private String videoName;
16
17  @Autowired
18  private FfmpegService ffmpegService;
19
20  @Autowired
21  private ClassRecProperties classRecProperties;
22 ...
23 }
```

Se pueden observar en las líneas 8-15 los diferentes atributos que utilizamos para controlar la grabación, saber en que estado se encuentra:

- `timeCounter`: Se encarga de gestionar el tiempo de las pausas y grabaciones.
- `timeCounterforFrontends`: Es un simple contador de tiempo para que todos los clientes tengan un contador de tiempo que pueda verse en pantalla.
- `onPause`: True si la grabación está pausada.
- `mobileRecording`: True si la grabación comenzó desde un móvil.
- `cuts`: Lista con los cortes que lleva la grabación. Al finalizar la grabación se genera un fichero como el siguiente:

  ```
  {
    "cuts": [
      {
        "start": "00:00:00",
        "end": "00:03:10"
      },
      {
        "start": "00:03:17",
        "end": "00:03:50"
      }
    ]
  }
  ```

  Este fichero indica las partes del video que deben ser procesadas. Cuando una grabación se pausa se genera un metadato con la información de inicio de ese corte y final de ese corte, para posteriormente poder cortar el vídeo automáticamente o editarlo a traves de la UI.

  - `previousTime`: Variable de tiempo para controlar el punto último en que se realizó una acción de grabación, ya sea pausar o continuar.

- `actualTime`: Momento de tiempo actual.
- `videoName`: Nombre del video.

Podemos ver también nuestro servicio de Ffmpeg en la línea 18 inyectado en el WebSocket que habíamos explicado en secciones anteriores. Ahora el WebSocket podrá utilizar este servicio, y el programador puede hacer llamadas a Ffmpeg sin preocuparse de ningun comando, y es gestionado el estado del proceso por el propio servicio.

Uno de las atributos más importantes de esta clase que no hemos mencionado es el `sessions` de la linea 5. En este `ArrayList` de `WebSocketSession` guardamos todas las conexiones de los dispositivos cliente que intenten conectarse a la aplicación. La clase hereda de `TextWebSocketHandler` con el cual implementamos el siguiente método:

```
1  @Override
2    public void afterConnectionEstablished(WebSocketSession session) throws IOException {
3      Gson gson = new Gson();
4      log.info(ConsMsg.CONNECTION_OK);
5      sessions.add(session);
6      String messageToSend = gson.toJson(
7        new WebSocketRecordMessageServer(ConsMsg.CONNECTION_OK, false),
8        WebSocketRecordMessageServer.class);
9      session.sendMessage(new TextMessage(messageToSend));
10   }
```

Este método es ejecutado cada vez que un cliente solicita una sesión de WebSocket en la ruta `/recordpc`. En la ejecución de este método guardamos en un Array dicha sesión para que por cada evento que suceda, enviarle información del estado de la grabación a todos los clientes conectados.

Posteriormente definimos el metodo que controlará la grabación en función de los mensajes por WebSockets que reciba:

```
1  @Override
2  protected void handleTextMessage(WebSocketSession session, TextMessage message)
3                                      throws IOException {
4    Gson gson = new Gson();
5    WebSocketRecordMessageClient messageObjectClient = gson.fromJson(
6      message.getPayload(), WebSocketRecordMessageClient.class);
7    WebSocketRecordMessageServer messageObjectServer;
8    TextMessage messageToSend;
9    String action = messageObjectClient.getAction();
10   switch(action) {
11   case ConsMsg.REC_VID_AUD:
12     messageObjectServer = recordVideoAndAudio(messageObjectClient);
13     messageToSend = new TextMessage(gson.toJson(messageObjectServer,
14       WebSocketRecordMessageServer.class));
15     sendMessageToAllSenders(messageToSend);
16     break;
17   case ConsMsg.STOP:
18     messageObjectServer = stopRecording();
```

```
19      messageToSend = new TextMessage(gson.toJson(messageObjectServer,
20        WebSocketRecordMessageServer.class));
21      sendMessageToAllSenders(messageToSend);
22      break;
23    case ConsMsg.PAUSE:
24      messageObjectServer = pauseRecording();
25      messageToSend = new TextMessage(gson.toJson(messageObjectServer,
26        WebSocketRecordMessageServer.class));
27      sendMessageToAllSenders(messageToSend);
28      break;
29    case ConsMsg.CONTINUE:
30      messageObjectServer = continueRecording();
31      messageToSend = new TextMessage(gson.toJson(messageObjectServer,
32        WebSocketRecordMessageServer.class));
33      sendMessageToAllSenders(messageToSend);
34      break;
35    case ConsMsg.REC_VID:
36      messageObjectServer = recordVideo(messageObjectClient);
37      messageToSend = new TextMessage(gson.toJson(messageObjectServer,
38        WebSocketRecordMessageServer.class));
39      sendMessageToAllSenders(messageToSend);
40      break;
41    }
42  }
```

En función de cada mensaje, se ejecutará un método correspondiente para la grabación. Como podemos ver hay 5 casos, los cuales, segun el método que ejecuten y el estado de los atributos, harán cortes, pararán la grabación, etc. Al final de cada caso, siempre enviamos a todos los clientes un mensaje con información del estado de la grabación en el método `sendMessageToAllSenders(messageToSend)`.

Como son bastante los métodos que manejan la grabación en el `WebSocketRecordHandler` vamos a comentar el más importante: el método `recordVideoAndAudio(messageObjectClient)` que es llamado en la linea 12:

```
1   private WebSocketRecordMessageServer recordVideoAndAudio(
2     WebSocketRecordMessageClient messageObject) {
3     if(ffmpegService.isFfmpegWorking()) {
4       return new WebSocketRecordMessageServer(ConsMsg.CANT_RECORD, true);
5     }
6     setConfigurationFfmpeg(messageObject);
7     timeCounter.restart();
8     timeCounterforFrontends.restart();
9     actualTime = timeCounter.getTimeCounter();
10    try {
11      ffmpegService.setDirectory(classRecProperties.getVideosFolder().toString());
12      ffmpegService.startRecordingVideoAndAudio();
13      return new WebSocketRecordMessageServer(ConsMsg.RECORDING, false);
14    } catch (IOException e) {
15      return new WebSocketRecordMessageServer(ConsMsg.IO_EXCEPTION
```

```
16        + " " + e.getMessage(), true);
17    } catch (FfmpegException e) {
18      return new WebSocketRecordMessageServer(ConsMsg.FFMPEG_EXCEPTION
19        + " " + e.getMessage(), true);
20    } catch (ICommandFileExistException e) {
21      return new WebSocketRecordMessageServer(ConsMsg.VIDEO_EXISTS_EXCEPTION, true);
22    } catch (ICommandException e) {
23      e.printStackTrace();
24      return new WebSocketRecordMessageServer(ConsMsg.ICOMMAND_EXCEPTION, true);
25    }
26  }
```

Es muy importante tener en cuenta todos los posibles momentos en que podemos ejecutar la grabación de video. Es por eso que al principio en la línea 2 comprobamos si Ffmpeg está ya en ejecución y devolvemos el mensaje pertintente de error. Después configuramos el servicio de Ffmpeg en la línea 6 con el mensaje que ha enviado el cliente (el cual incluye el nombre del video, el framerate y el formato de vídeo). Reseteamos las variables de tiempo y comenzamos la grabación en la línea 12.

Es importante mencionar que cualquier tipo de excepción devuelta por el servicio es controlada y enviada con su mensaje correspondiente para que todos los dispositivos reciban la causa del problema en caso de darse.

En los clientes, la grabación es bastante sencilla de controlar. Como ambos hacen uso de Angular podemos incluso implementar el mismo servicio en el frontend de PC y el de Ionic (Aplicación móvil). El servicio implementado en ambos clientes es el siguiente:

```
1  @Injectable()
2  export class WebSocketRecord {
3      public messages: Subject<string>;
4
5      constructor(private wsService: WebsocketRecordService) {
6          let WS_URL;
7          console.log(environment);
8          if (environment.production) {
9              WS_URL = `ws://${document.location.host}/recordpc`;
10         } else {
11             WS_URL = `${environment.webSocketUrl}/recordpc`;
12         }
13         this.messages = <Subject<string>>wsService
14             .connect(WS_URL)
15             .pipe(map((response: MessageEvent): string => {
16                 const data = response.data;
17                 return data;
18             })
19         );
20     }
21
22     sendMessage(msg: WebSocketRecordMessage) {
23         this.wsService.sendMessage(msg);
```

```
24      }
25  }
```

De aquí es importante mencionar dos partes. Por un lado el método `sendMessage(msg: WebSocketRecordMessage)` el cual nos va a permitir enviar los mensajes necesarios para controlar las grabaciones. También hay que mencionar el atributo público `messages`, el cual es un Observable el cual está conectado al WebSocket. Cualquier componente de Angular que se suscriba a ese atributo recibira los mensajes de respuesta del servidor, con los cuales notificamos visualmente al usuario del estado de la grabación.

4.4. Evaluación y pruebas

En esta sección explicaremos en detalle las pruebas que se han realizado, manuales y automatizadas, y los pasos que se dan en cada commit subido a Github.

Como hemos explicado en la sección 3.4 todo el sistema de desarrollo y de pruebas está dockerizado.

Hemos configurado travis para que ofrecta un servicio de travis:

```
services:
  - docker
  - mysql
```

Y ejecutamos un script que levanta todo el sistema y realiza los test

```
script:
  - bash scripts_ci/travis.sh
```

El cual es el siguiente:

```
git clone https://github.com/Class-Recorder/docker-class-recorder
cd docker-class-recorder/docker-runnables/crecorder-ci
./docker_run.sh
```

El script `docker_run.sh` lo que hace es ejecutar todos los test en una imagen docker y preparar los ficheros en una carpeta concreta si todos los test se han realizado correctamente. Travis está configurado para subir los ejecutables de dicha release.

```
deploy:
  provider: releases
  api_key:
    secure: <api_key>
  file_glob: true
  file: docker-class-recorder/docker-runnables/crecorder-ci/release/*
  skip_cleanup: true
  on:
    tags: true
```

Al ejecutarse el script `docker_run.sh` todos los ficheros generados acaban en la carpeta

`docker-class-recorder/docker-runnables/crecorder-ci/release/` y su contenido es subido como release en Github por cada tag realizada.

Los test que se han realizado son de tipo e2e en el browser, con `protactor`. Aunque por el momento solo he sido capaz de automatizar los test del frontend a nivel de grabaciones. Las pruebas de móvil son pruebas manuales.

5. Conclusiones y trabajos futuros

2 a 4 páginas

6. Referencias

[1] J. S. Ken Schwaber, "La guía de scrum." https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf, 2013.

[2] I. Sommerville, *Ingeniería del software*. pp. 62–68.

[3] "Semantic versioning 2.0.0." https://semver.org.

[4] T. L. I. Project, "X11 definition." http://www.linfo.org/x11.html, 2005.

[5] "About pulseaudio." https://www.freedesktop.org/wiki/Software/PulseAudio/About/, 2014.

[6] R. M. Agut, *Especificación de requisitos software según el estándar de ieee 830.*.

7. Anexos

7.1. Anexo 1

7.2. Anexo 2

7.3. Anexo 3

7.4. Anexo 4