YouTubeCrawlerTool: Aplicación web para habilitar el estudio del movimiento antivacuna

Javier Sánchez Mendoza
Grado de ingeniería informática
Health IT

Carlos Luis Sánchez Bocanegra
José Antonio Morán Moreno

Junio de 2018

Quiero dedicar este trabajo especialmente a:

Carolina
Por empujarme a retomar mis estudios y, lo que es mas importante, motivarme durante todo este tiempo.

Amy, Luke y Jim
Por obligarme a salir a la calle de vez en cuando y estar siempre ahí.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*YouTubeCrawlerTool: Aplicación web para habilitar el estudio del movimiento antivacuna*
Nombre del autor:	*Javier Sánchez Mendoza*
Nombre del consultor/a:	*Carlos Luis Sánchez Bocanegra*
Nombre del PRA:	*José Antonio Morán Moreno*
Fecha de entrega:	*06/2018*
Titulación:	*Grado de ingeniería informática*
Área del Trabajo Final:	*Health IT*
Idioma del trabajo:	*Español*
Palabras clave:	*antivacuna, crawler, YouTube.*

Resumen del Trabajo:

En este proyecto final de grado en ingeniería informática se procede a realizar un recolector de contenidos (o *crawler* en ingles) para habilitar el estudio del movimiento antivacuna en redes sociales.

Gracias en parte a las redes sociales, actualmente el activismo antivacunas goza de una alta popularidad, hecho que a propiciado la aparición de enfermedades que se consideraban erradicadas y las cuales, en el peor de los casos, han llegado ha cobrarse victimas mortales. Con el propósito de conocer mejor dicho movimiento y luchar contra la desinformación generada al respecto, se pretende habilitar una aplicación para analizar su divulgación en las redes sociales.

Para conseguir tal objetivo, se trabaja conjuntamente con una alumna en el desarrollo de su trabajo final de máster. En nuestro proyecto, dicha alumna actúa con el rol de clienta y usuaria de la aplicación. Debido a este escenario, se adopta una metodología centrada en el usuario que nos permite garantizar, en la medida de lo posible, la máxima satisfacción de la usuaria. Y para garantizar la ejecución del proyecto, se aplica el marco ágil de desarrollo *Scrum*.

Como resultado se obtiene una aplicación web que permite la recolección de vídeos de la red social *YouTube* y su posterior categorización. La aplicación cumple con las expectativas generadas por la clienta, permitiendo el análisis de los contenidos obtenidos mediante una visualización en formato de grafo y su exportación a fichero csv. Así mismo, la herramienta es lo suficientemente genérica para ser utilizada en otros contextos.

Abstract:

In this final project in computer engineering degree, we will make a crawler to enable the study of the anti-vaccination movement in social networks.

Thanks in part to the social networks, currently antivaccines activism enjoys a high popularity, a fact that leads to the occurrence of lethal diseases that were considered eradicated. In order to better understand this movement and fight against the misinformation generated, an application it will be developed to analyze its dissemination in social networks.

To achieve this goal, we work together with a student in the development of her master final project. In our common project, this student will have the role of client and user of the application. Due this scenario, a user-centered methodology is adopted that allows us to guarantee, as far as possible, the maximum satisfaction of the user. And to ensure the execution of the project, it will be developed using agile methodology Scrum.

As a result, a web application is obtained that allows the collection of videos from the YouTube social network and its subsequent categorization. The application complies with the expectations generated by the client, allowing the analysis of the contents obtained through a visualization in graph format and its export into a csv file. Likewise, the tool is generic enough to be used in other contexts.

Índice

Índice de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Desde la introducción de la vacunación como método preventivo de enfermedades han existido entidades y grupos de personas que se han opuesto a ella y han dudado de su efectividad o propósito [1]. Hoy en día el activismo anti-vacunación (conocido también como movimiento antivacunas) ha vuelto a la actualidad y se encuentra en auge en algunas regiones tales como Europa o Estados Unidos, cobrándose en el peor de los escenarios victimas mortales a causa de enfermedades que se creían erradicadas y que han vuelto a surgir [2][3].

Para hacer posible el estudio y comprensión de las motivaciones del movimiento antivacuna y luchar contra su desinformación, se propone el desarrollo de una aplicación que permita la recolección de grandes cantidades de datos de la actividad realizada por parte de este colectivo en redes sociales con el fin de hacer posible su posterior tratamiento y estudio para obtener valor añadido. Para tal fin, en este proyecto contamos con la colaboración de Johanna Milena Rodríguez Vera estudiante de master en Telemedicina que asume el rol de analista de datos (*data scientist* [4]) en el desarrollo de su trabajo final de máster titulado *Evaluación de la información sanitaria en vacunas disponible en las redes sociales - YouTube* y que actúa a la vez como clienta de la aplicación desarrollada en el presente proyecto.

Hoy en día las redes sociales han puesto al alcance de los analistas de datos una gran cantidad de información disponible para ser analizada, una de las problemáticas a las que se quiere hacer frente es la obtención de dichos datos de forma efectiva. Para ello se propone hacer uso de interfaces de programación de aplicaciones (abreviado como *API* [5] en ingles) ofrecidas públicamente por distintas redes sociales de tal forma que el proceso resulte transparente para el usuario final, permitiéndole la extracción a este problema.

La obtención de grandes volúmenes de datos nos lleva también a la problemática que surge en su almacenamiento en bases de datos tradicionales y su posterior procesamiento. Para habilitar al usuario final el correcto acceso a la información obtenida se estudían las ventajas que aporta el uso de bases de datos *NoSQL* [6] para este cometido, al ser diseñadas especialmente para manejar enormes cantidades de datos. Proyecto que se enmarca dentro de la problemática de la obtención, almacenamiento y procesamiento de grandes volúmenes de datos (*Big Data* [7]) y su posterior acceso.

1.2. Objetivos del Trabajo

El objetivo principal del proyecto es proporcionar una aplicación web que permita a la clienta obtener de forma usable y transparente la información que necesite de la red social *YouTube* para poder llevar a cabo el estudio de patrones de comportamiento entre las diferentes movimientos anti y pro vacuna.

Entre los objetivos secundarios del proyecto se encuentra la exploración de otras redes sociales y proporcionar una herramienta lo suficientemente genérica para que pueda ser utilizada en la investigación realizada por Johanna así como en otras investigaciones futuras de distinta temática.

Algunos objetivos concretos que se han querido lograr son los siguientes:

- Investigar que funcionalidades aportan las *API* públicas ofrecidas por *YouTube* y analizar como se pueden utilizar para la obtención de la información requerida.
- Determinar como almacenar y acceder de forma eficiente a la gran cantidad de información que se obtendrá.
- Permitir la recolección de información según criterios de búsqueda proporcionados por el usuario final.
- Habilitar la gestión, visualización y exportación de datos obtenidos en distintos procesos de extracción para su posterior análisis en herramientas especializadas.
- Ofrecer herramientas de visualización para el análisis y comprensión de los datos obtenidos.
- Proporcionar una interfaz de usuario usable que permita realizar las acciones requeridas por el usuario final.

1.3. Enfoque y método seguido

Para la realización del proyecto se ha seguido el marco ágil de desarrollo *scrum* [8]. Al adoptar esta metodología como marco de trabajo nos ha permitido, a diferencia de otras metodologías lineales de desarrollo como pueden ser los modelos en cascada, poder desarrollar el proyecto de forma flexible permitiéndonos adaptar la planificación inicial del proyecto en los casos necesarios para adecuarse a los nuevos requerimientos.

La forma en la cual se aplico la metodología *scrum* en el proyecto esta condicionada por los integrantes del equipo de desarrollo, en el cual la figura

del *product owner*, *scrum master* y desarrollador recaen sobre la figura del alumno que presenta el actual proyecto descrito (Javier Sánchez Mendoza), mientras que la figura del cliente estará representada por una analista de datos (Johanna Milena Rodríguez Vera) y el consultor de los mismos (Carlos Luis Sánchez Bocanegra) como *stakeholder*.

Siguiendo la metodología *scrum*, se realizaron iteraciones (comúnmente conocidos como *sprints*) de una semana de duración donde, en la finalización de los mismos, se realizaron reuniones online para revisar y aprobar las tareas realizadas (*sprint review*) y definir las tareas para la siguiente iteración (*sprint planning*). Para gestionar las tareas a realizar (*backlog*) se decidio utilizar la herramienta online *Trello* [9]:

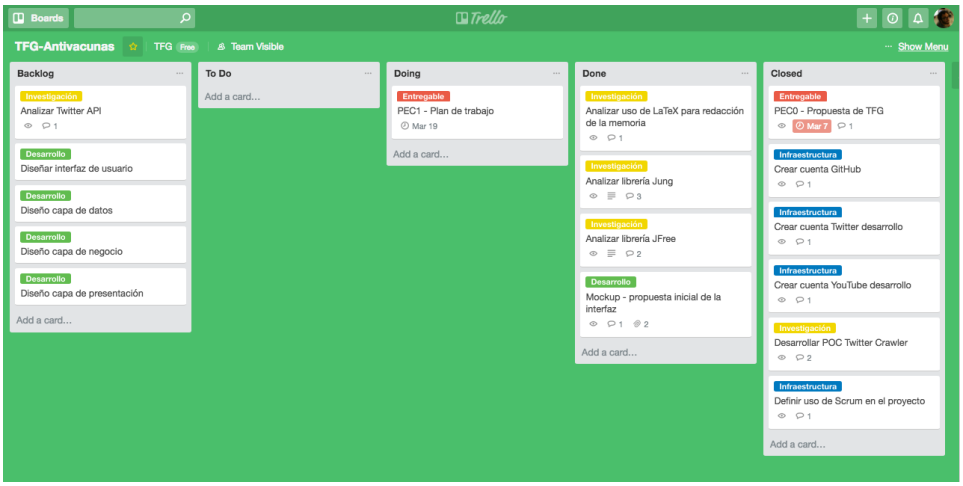
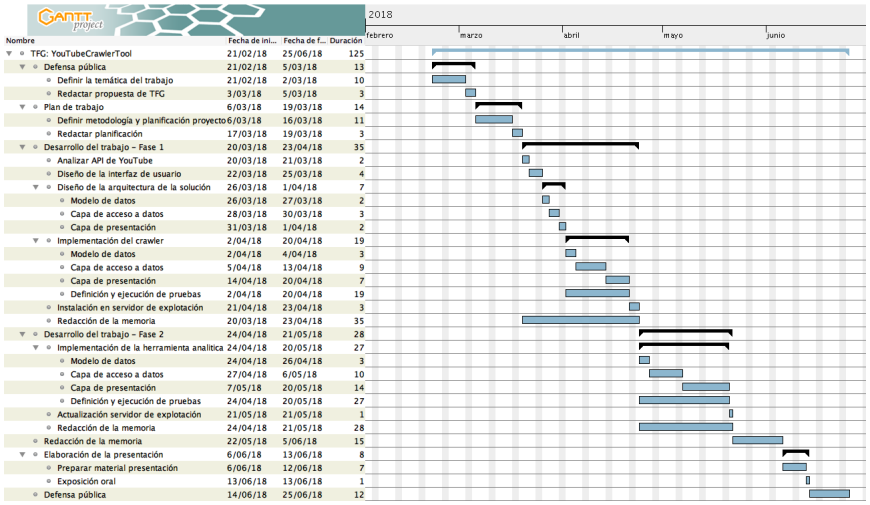


Figura 1: Ejemplo del backlog del proyecto en Trello

1.4. Planificación del Trabajo

En la realización del proyecto se propuso inicialmente seguir una planificación tentativa tal y como se detalla en el diagrama de *Grantt* facilitado en las figuras dos y tres:



Nombre	Fecha de ini...	Fecha de f...	Duración
TFG: YouTubeCrawlerTool	21/02/18	25/06/18	125
Defensa pública	21/02/18	5/03/18	13
Definir la temática del trabajo	21/02/18	2/03/18	10
Redactar propuesta de TFG	3/03/18	5/03/18	3
Plan de trabajo	6/03/18	19/03/18	14
Definir metodología y planificación proyecto	6/03/18	16/03/18	11
Redactar planificación	17/03/18	19/03/18	3
Desarrollo del trabajo - Fase 1	20/03/18	23/04/18	35
Analizar API de YouTube	20/03/18	21/03/18	2
Diseño de la interfaz de usuario	22/03/18	25/03/18	4
Diseño de la arquitectura de la solución	26/03/18	1/04/18	7
Modelo de datos	26/03/18	27/03/18	2
Capa de acceso a datos	28/03/18	30/03/18	3
Capa de presentación	31/03/18	1/04/18	2
Implementación del crawler	2/04/18	20/04/18	19
Modelo de datos	2/04/18	4/04/18	3
Capa de acceso a datos	5/04/18	13/04/18	9
Capa de presentación	14/04/18	20/04/18	7
Definición y ejecución de pruebas	2/04/18	20/04/18	19
Instalación en servidor de explotación	21/04/18	23/04/18	3
Redacción de la memoria	20/03/18	23/04/18	35
Desarrollo del trabajo - Fase 2	24/04/18	21/05/18	28
Implementación de la herramienta analítica	24/04/18	20/05/18	27
Modelo de datos	24/04/18	26/04/18	3
Capa de acceso a datos	27/04/18	6/05/18	10
Capa de presentación	7/05/18	20/05/18	14
Definición y ejecución de pruebas	24/04/18	20/05/18	27
Actualización servidor de explotación	21/05/18	21/05/18	1
Redacción de la memoria	24/04/18	21/05/18	28
Redacción de la memoria	22/05/18	5/06/18	15
Elaboración de la presentación	6/06/18	13/06/18	8
Preparar material presentación	6/06/18	12/06/18	7
Exposición oral	13/06/18	13/06/18	1
Defensa pública	14/06/18	25/06/18	12

Figura 2: Listado de tareas y diagrama de Gantt

Nombre	Fecha de ini...	Fecha de f...	Duración
TFG: YouTubeCrawlerTool	21/02/18	25/06/18	125
Defensa pública	21/02/18	5/03/18	13
Definir la temática del trabajo	21/02/18	2/03/18	10
Redactar propuesta de TFG	3/03/18	5/03/18	3
Plan de trabajo	6/03/18	19/03/18	14
Definir metodología y planificación proyecto	6/03/18	16/03/18	11
Redactar planificación	17/03/18	19/03/18	3
Desarrollo del trabajo - Fase 1	20/03/18	23/04/18	35
Analizar API de YouTube	20/03/18	21/03/18	2
Diseño de la interfaz de usuario	22/03/18	25/03/18	4
Diseño de la arquitectura de la solución	26/03/18	1/04/18	7
Modelo de datos	26/03/18	27/03/18	2
Capa de acceso a datos	28/03/18	30/03/18	3
Capa de presentación	31/03/18	1/04/18	2
Implementación del crawler	2/04/18	20/04/18	19
Modelo de datos	2/04/18	4/04/18	3
Capa de acceso a datos	5/04/18	13/04/18	9
Capa de presentación	14/04/18	20/04/18	7
Definición y ejecución de pruebas	2/04/18	20/04/18	19
Instalación en servidor de explotación	21/04/18	23/04/18	3
Redacción de la memoria	20/03/18	23/04/18	35
Desarrollo del trabajo - Fase 2	24/04/18	21/05/18	28
Implementación de la herramienta analítica	24/04/18	20/05/18	27
Modelo de datos	24/04/18	26/04/18	3
Capa de acceso a datos	27/04/18	6/05/18	10
Capa de presentación	7/05/18	20/05/18	14
Definición y ejecución de pruebas	24/04/18	20/05/18	27
Actualización servidor de explotación	21/05/18	21/05/18	1
Redacción de la memoria	24/04/18	21/05/18	28
Redacción de la memoria	22/05/18	5/06/18	15
Elaboración de la presentación	6/06/18	13/06/18	8
Preparar material presentación	6/06/18	12/06/18	7
Exposición oral	13/06/18	13/06/18	1
Defensa pública	14/06/18	25/06/18	12

Figura 3: Listado de tareas

De la planificación inicial cabe destacar que la redacción de la memoria se diseño como una tarea evolutiva que se desarrollaría durante todas las fases del proyecto pero mas intensamente durante la antepenúltima fase dedicada exclusivamente a su redacción. Además, durante las dos fases de desarrollo se planificaron dos tareas recurrentes para la definición y realización de pruebas de calidad del producto a realizar durante todo el ciclo de desarrollo.

Como se puede observar, en el diagrama facilitado en cada entrega se pretendían conseguir unos hitos concretos. La relación de los mas destacables por entrega son los siguientes:

- **Definición de los contenidos del trabajo:** Redacción propuesta TFG.
- **Plan de trabajo:** Redacción planificación.
- **Desarrollo del trabajo – Fase 1:** Instalación en servidor de explotación de la primera versión de la aplicación con funcionalidad de *crawler* implementada.
- **Desarrollo del trabajo – Fase 2:** Actualización en servidor de explotación de la versión final de la aplicación con funcionalidad analítica implementada.
- **Redacción de la memoria:** Entrega de la memoria del proyecto.
- **Elaboración de la presentación:** Realizar exposición oral.
- **Defensa pública:** Defender públicamente el proyecto.

Los riesgos detectados en la planificación inicial se concentraban principalmente en la consecución del hito definido en la primera fase de desarrollo. Para permitir a la clienta de la aplicación poder empezar a recopilar datos para su investigación lo antes posible, se decidió realizar la instalación de la aplicación desarrollada en un entorno de explotación en dos fases distintas, una con la funcionalidad del *crawler* y otra con la funcionalidad analítica implementada. La demora en la primera fase de desarrollo podría comprometer el éxito de la investigación del cliente. Para mitigar este riesgo se realizo el seguimiento del mismo durante las diferentes iteraciones en esa fase de desarrollo.

Cabe destacar que el diagrama proporcionado en esta sección representa la estimación inicial de la planificación del proyecto de forma tentativa y fue sujeto a modificaciones al inicio de cada nueva fase de desarrollo para garantizar el éxito del proyecto. En la sección de conclusiones del presente documento en el apartado 5.3 se realiza una comparación entre la planificación inicial y sus respectivas revisiones.

1.5. Arquitectura tecnológica

Para la consecución de los objetivos definidos se ha desarrollado una aplicación web bautizada como *YouTubeCrawlerTool* la cual ha sido diseñada por capas.

En la capa de datos se utiliza una base de datos orientada a grafos [10] la cual nos permite persistir los vídeos obtenidos en forma de grafo junto con sus relaciones ademas de otras informaciones derivadas y necesarias para el uso y funcionamiento de la aplicación.

La capa de aplicación ha sido desarrollada con tecnología Java EE [11] implementada mayormente utilizando proyectos del marco de desarrollo *Spring Framework* [12] entre otros. En esta capa se hace uso intensivo de servicios externos en forma de API pública ofrecida por *YouTube* con el fin de consumir dicha API para recolectar los vídeos requeridos por el usuario para su estudio.

Finalmente, en la capa de cliente las vistas desarrolladas utilizan lenguaje JavaScript [13] con un gran uso de *JQuery* [14] para modificar el DOM y realizar llamadas asíncronas a la API REST habilitada en la capa de aplicación para tales efectos.

En los siguientes apartados se profundiza en la arquitectura y el diseño de la aplicación introducidos en esta sección.

1.6. Resumen de capítulos

En los próximos capítulos se detalla el trabajo realizado juntamente con los productos obtenidos y sus conclusiones. La relación de capítulos es la siguiente:

- **Análisis y diseño:** Explicación de la metodología de diseño escogida así como los pasos que se llevaron a cabo para definir una propuesta a la clienta y su posterior análisis para acabar definiendo la arquitectura de la solución.
- **Desarrollo:** Descripción del desarrollo efectuado y de las decisiones realizadas durante el proceso.
- **Implementación y puesta en funcionamiento:** Detalle de los productos obtenidos con las instrucciones para su correcta puesta en funcionamiento ademas de las acciones de formación realizadas.
- **Conclusiones:** Sumario de resultados obtenidos y conclusiones sobre el trabajo realizado.

2. Análisis y diseño

2.1. Metodología

Para realizar el diseño de la aplicación se opto de entre diferentes posibilidades el enfoque definido por la filosofía del 'Diseño centrado en el usuario' [15].

El diseño centrado en el usuario, como bien indica su nombre, se caracteriza por conocer a fondo a los futuros usuarios de una aplicación para diseñar un producto que resuelva sus necesidades y expectativas buscando en todo momento conseguir la mayor satisfacción del usuario posible. Se trata de un proceso iterativo y cíclico por fases en donde en cada una de ellas se utilizan distintas técnicas para conseguir los objetivos propuestos.

En nuestro proyecto se ha decidido seguir esta y no otra metodología de diseño tales como el 'Diseño centrado en la actividad' o el 'Diseño centrado en el uso' debido a que, como en nuestro caso la clienta de la aplicación va a ser también la usuaria final de la misma, se ha decidido realizar el análisis de la aplicación centrada en ella y sus necesidades por encima de la actividad que se llevara a cabo o el uso.

La relación de fases y técnicas utilizadas ha sido la siguiente:

- **Definir contexto de uso:** El objetivo de esta fase es el de determinar que necesidades pretende la usuaria final que la aplicación resuelva y a que se va a destinar su uso. La técnica escogida para la realización de esta fase fueron las entrevistas con la usuaria final que se realizaron mediante videoconferencia sobretodo durante las diferentes *sprint reviews* al termino de cada *sprint*.
- **Especificar requisitos:** En la siguiente fase se definen los requisitos del sistema a partir de la información recogida en la fase previa. Para los requisitos funcionales se opto por recogerlos como 'casos de uso' [16].
- **Diseñar el producto:** En esta fase se diseñan y implementan los requisitos definidos en la fase anterior ya sea con el objetivo de proporcionar una solución final o una propuesta de solución a ser refinada en sucesivas iteraciones. Las técnicas utilizadas en esta fase fueron la creación de maquetas para evaluar la solución y el prototipo mediante pruebas de concepto realizadas con el objetivo de estudiar posibles soluciones antes de realizar su implementación en la aplicación.

- **Poner a prueba el producto:** Finalmente, en esta fase se pone a prueba el producto obtenido. Para hacerlo, se definieron y ejecutaron pruebas de integración diseñadas teniendo en cuenta los casos de uso definidos previamente y se realizaron test con usuarios para evaluar su grado de satisfacción.

Gracias al enfoque escogido fue posible encontrar respuestas a preguntas sobre las expectativas que la usuaria tenia depositadas sobre la aplicación y que fueron de gran ayuda a la hora de diseñar la solución final. Algunas de las principales preguntas fueron:
¿Quiénes son los usuarios de la aplicación?
¿Cuáles son las tareas a realizar?
¿Qué funcionalidades se necesitan?
¿Qué información se necesita?

También cabe destacar que gracias a que tanto *Scrum* como el diseño centrado en el usuario son dos procesos iterativos, resulto fácil integrar esta metodología dentro del marco ágil de desarrollo.

2.2. Propuesta de la solución

2.2.1. Reuniones y entrevistas

Tal y como se ha introducido en la sección 2.1 sobre la metodología de diseño utilizada, para definir el contexto de uso y conocer las necesidades a ser cubiertas por la aplicación, se realizaron varias entrevistas con la clienta y los *stakeholders* donde la gran mayoría de ellas fueron dentro del contexto de *scrum* como reuniones de *sprint review* y *sprint planning*.

A continuación se resumen las entrevistas y reuniones realizadas junto con los principales temas tratados y decisiones tomadas:

Fecha: 05/03/2018
Hora de inicio: 21:00
Hora final: 21:45
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta
- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Que conocimiento sobre el movimiento antivacunas se quiere obtener.

- Que se pretende hacer con la información recolectada.
- Definición de los objetivos de la aplicación.

Decisiones:

- La clienta definirá los criterios de búsqueda y el conocimiento que debe ser obtenido por la aplicación.
- El análisis de los datos los realizara la clienta con herramientas especializadas.
- El objetivo principal de la aplicación es la obtención de datos mediante búsquedas en redes sociales.
- La aplicación debe ofrecer funcionalidad para exportar los datos recolectados a otras herramientas.

Fecha: 12/03/2018
Hora de inicio: 21:00
Hora final: 21:30
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta
- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Alcance del estudio a realizar.
- Sobre que redes sociales debe centrarse el estudio.

Decisiones:

- Se decide adoptar el marco de trabajo *scrum* para la realización del proyecto.
- El estudio se centrara inicialmente en una sola red social a determinar.
- En la aplicación sera posible ejecutar varios procesos de recolección de datos al mismo tiempo.
- Se incorporara una sección para analizar los datos obtenidos de forma visual (por definir).

- El *product owner* realizara una propuesta inicial de la interfaz de usuario.

Fecha: 19/03/2018
Hora de inicio: 21:00
Hora final: 21:30
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta

Temas tratados:

- Comentarios sobre la propuesta inicial de la interfaz de usuario realizada por el *product owner*.
- Pros y contras sobre la elección de *Twitter* como red social para el estudio.

Decisiones:

- Elección de *Twitter* como red social a utilizar en el estudio.
- La aplicación incluirá una visualización en forma de grafo para poder analizar visualmente las relaciones existente en la información recolectada y descubrir patrones.
- El *product owner* debe estudiar la viabilidad de utilizar *Twitter* para la consecución de los objetivos.

Fecha: 26/03/2018
Hora de inicio: 21:00
Hora final: 21:45
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta

Temas tratados:

- Debido a las limitaciones de uso detectadas en la API de *Twitter*, se propone utilizar la red social *YouTube* como alternativa.

- Definición del grafo a utilizar y que elementos actuaran como nodo y aristas.

Decisiones:

- Elección de *YouTube* como red social a utilizar en el estudio.
- El *product owner* debe estudiar la viabilidad de utilizar *YouTube* para la consecución de los objetivos.
- Actualizar la propuesta de interfaz de usuario para reflejar el cambio de red social.

Fecha: 05/04/2018
Hora de inicio: 21:00
Hora final: 21:45
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta
- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Comentarios sobre la propuesta inicial de la interfaz de usuario realizada por el *product owner*.
- Analizar criterios de búsqueda y información devuelta por la API de *YouTube*
- Detección de vídeos duplicados.
- Detectar usuarios *influencers* a partir de la información obtenida.
- Avances en la definición de los componentes del grafo.

Decisiones:

- Criterios de búsqueda de *YouTube* a utilizar para recolectar los vídeos.
- Campos a almacenar de cada vídeo.
- Incorporar funcionalidad para ver resumen de las búsquedas realizadas junto con la información recolectada con pre visualización de vídeos.
- Utilizar una variable pre calculada (bautizada como "*scopeRange*") para determinar el tamaño de los nodos al ser visualizados en el grafo.

Fecha: 12/04/2018
Hora de inicio: 21:30
Hora final: 22:15
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta
- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Comentarios sobre la visualización en grafo.
- Identificación de vídeos pro y anti vacunación.

Decisiones:

- Queda definida la visualización en grafo.
- Los contenidos a visualizar en el grafo podrán ser filtrados.
- Se define funcionalidad para etiquetar los vídeos recolectados utilizando categorías previamente definidas por el usuario en la aplicación.
- En la aplicación habrá dos tipos de usuarios, usuarios anónimos que no podrán realizar acciones de escritura ni borrado y usuarios registrados que podrán realizar todas la acciones.
- Incorporar visualizaciones estadísticas sobre el uso de las categorías.
- Realización de una ultima propuesta de interfaz de usuario que recoja los últimos cambios propuestos junto con la visualización en grafo.

Fecha: 19/04/2018
Hora de inicio: 21:30
Hora final: 22:00
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta
- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Comentarios sobre la propuesta de interfaz de usuario.
- Priorización de funcionalidades.

Decisiones:

- La clienta da por aprobada la propuesta de interfaz de usuario.
- Definición de valores por defecto al realizar las búsquedas de contenidos.
- Las funcionalidades estadísticas y de visualización de canales quedan asignadas con una prioridad secundaria en relación a otras funcionalidades.
- Se va a buscar la colaboración de un estadista para definir la formula de la variable "*scopeRange*".

Fecha: 26/04/2018
Hora de inicio: 21:00
Hora final: 21:30
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta
- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Seguimiento de la implementación de la aplicación
- Seguimiento de la definición de la fórmula para calcular el alcance de los vídeos ("*scopeRange*").

Decisiones:

Fecha: 03/05/2018
Hora de inicio: 21:00
Hora final: 21:30
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta

- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Seguimiento de la implementación de la aplicación.
- Seguimiento de la definición de la fórmula para calcular el alcance de los vídeos ("*scopeRange*").

Decisiones:

Fecha: 10/05/2018
Hora de inicio: 20:30
Hora final: 21:00
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta
- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Seguimiento de la implementación de la aplicación.
- Seguimiento de la definición de la fórmula para calcular el alcance de los vídeos ("*scopeRange*").

Decisiones:

- Instalación de la aplicación en entorno de explotación durante la próxima semana.
- Mientras no se disponga de la formula para la variable "*scopeRange*", utilizar formula alternativa definida por la clienta.

Fecha: 17/05/2018
Hora de inicio: 20:30
Hora final: 21:30
Asistentes:

- Carlos Luis Sánchez Bocanegra: *stakeholder*
- Johanna Milena Rodríguez Vera: clienta

- Javier Sánchez Mendoza: *product owner*, *scrum master* y desarrollador

Temas tratados:

- Presentación de la aplicación desarrollada y formación a la clienta.

Decisiones:

- Se decide utilizar la formula alternativa propuesta por la clienta para la variable "*scopeRange*".
- Añadir nueva funcionalidad de favoritos.
- Realización de cambios en la visualización de los listados de los vídeos.

En los siguientes apartados se detallan algunas de las decisiones tomadas en estas reuniones las cuales tuvieron mas repercusión en el resultado final de la aplicación.

2.2.2. Elección de YouTube como red social

A la hora de escoger una red social para realizar el estudio del movimiento antivacuna nos encontramos con una gran variedad de opciones, tales como *Facebook*, *Twitter* o *YouTube* para nombrar solo algunas.

Johanna, clienta de la aplicación y la encargada de realizar el estudio, mostró su predilección inicial por *Twitter*. En su criterio, *Twitter* presentaba una estructuración de la información que favorecía su posterior análisis y estudio al basarse, principalmente, un mensaje (*tweet*) de los campos titulo y descripción, hecho que facilitaba la categorización de los contenidos obtenidos. Por otro lado, el uso extensivo de etiquetas en esta red social (*hashtags*) simplificaría el proceso de búsqueda de contenidos. Por lo referente al alcance de la red social, Johanna consideraba que a nivel de usuarios y actividad relacionada con la vacunación, *Twitter* superaba a otras redes sociales tales como *YouTube*.

Debido a estas consideraciones, *Twitter* fue escogida inicialmente como la red social a utilizar en el proyecto. Por este motivo, inicialmente la aplicación desarrollada se llamaba *TwitterCrawlerTool* y las primeras versiones de la propuesta de interfaz de usuario se habían realizado teniendo en consideración a *Twitter* como red social escogida. Incluso, una prueva de concepto llego a desarrollarse: POCTwitterCrawler.

Con la elección realizada y teniendo desarrollada una prueba de concepto, se procedió al estudio en profundidad de la API de *Twitter* y fue entonces

cuando se descubrieron limitaciones sobre su uso. Concretamente, *Twitter* define tres niveles de uso: *Standard*, *Premium* y *Enterprise*; de todas ellas solo la opción *Standard* es completamente gratuita pero, en este caso, con unas severas restricciones de uso a la hora de buscar mensajes [17]. Algunas de las restricciones son:

- Máximo de 100 *"tweets"* por búsqueda.
- Información disponible solo de los últimos 7 días.
- Solo permite búsqueda textual, no se permite búsqueda temporal.

A causa de estas restricciones, resultaba imposible poder recabar información con una antigüedad inferior a siete días o realizar comparaciones en el tiempo sobre la evolución de los movimientos pro y anti vacunas, hecho que es requerido en el estudio. Ante los descubrimientos realizados se decidió explorar otras alternativas al uso de *Twitter*.

Fue entonces que la opción de utilizar *YouTube* como red social de estudio en el proyecto se considero en profundidad. Y es que aunque se llegara anteriormente a la conclusión de que *Twitter* tenia un alcance mayor en numero de usuarios y potencial de contenidos a obtener, no se debe desestimar tampoco el alcance de *YouTube* que, si bien inicialmente no era considerada como una red social al uso, a día de hoy el numero de usuarios que no solo visualizan sus vídeos sino que también comparten contenidos y comentarios crece día a día, convirtiendo a *YouTube* como una buena opción para encontrar la información requerida. Y en lo referente a la estructuración de la información, debido a que juntamente con los vídeos se provee un titulo y una descripción no era necesario cambiar el enfoque dado inicialmente en este sentido.

Por lo referente a la API ofrecida por *YouTube* para la obtención de contenidos, los criterios de búsqueda disponibles son mucho mas amplios que los ofrecidos por *Twitter*, permitiéndonos entre otras posibilidades, filtrar los contenidos a obtener por rangos de fechas. No existe limitación a la hora de obtener contenidos independientemente de su fecha de publicación (en todo caso posterior a 14/02/2005, fecha de fundación de *YouTube*). Y en relación a limitaciones en el volumen de información a obtener, *YouTube* no impone limitaciones por búsqueda, lo que posibilita obtener toda la información requerida sobre un termino en concreto. En su defecto, *YouTube* utiliza un sistema de cuotas que se aplica a periodos de tiempo en concreto [18], por ejemplo, en su versión gratuita permite realizar hasta 1.000.000 de operaciones de lectura por día que, tal y como se ha demostrado, han sido suficientes para el uso dado a la aplicación desarrollada. Para estudiar su viabilidad en el proyecto, se realizo una prueba de concepto con resultados

satisfactorios: POCYouTubeCrawler.

En el apartado 3.2.1 se detalla en profundidad los servicios de la API de *YouTube* utilizados y el modo en que la aplicación los consume para obtener contenidos.

2.2.3. Visualización de vídeos en grafo

Otra de las decisiones de diseño mas debatidas fue la definición de una vista en la aplicación que permitiera de forma visual analizar como se relacionan los movimientos anti y pro vacuna en la red social de *YouTube*.

La motivación principal de la misma es la de proporcionar una funcionalidad con la cual fuera posible estudiar las vías de acceso a los contenidos y observar que movimiento tiene mas alcance de audiencia en *YouTube*. Para tales efectos, se llego a la conclusión que una visualización en formato de grafo donde fuera posible distinguir los dos movimientos a estudio seria la forma mas efectiva de representar dicha información.

Para difinir el grafo, necesitábamos analizar que componentes actuarían como nodo y que representarían las aristas entre ellos, ademas de tomar otras decisiones como si se incorporarían pesos al grafo y si este seria dirigido o no. Para ayudar en la definición del grafo, se realizaron propuestas que se apoyaron con pruebas realizadas manualmente por parte de la clienta con contenidos reales:

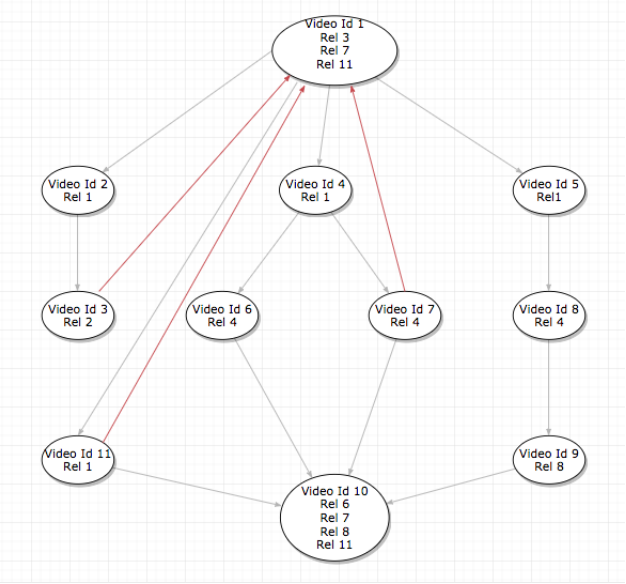


Figura 4: Propuesta grafo dirigido

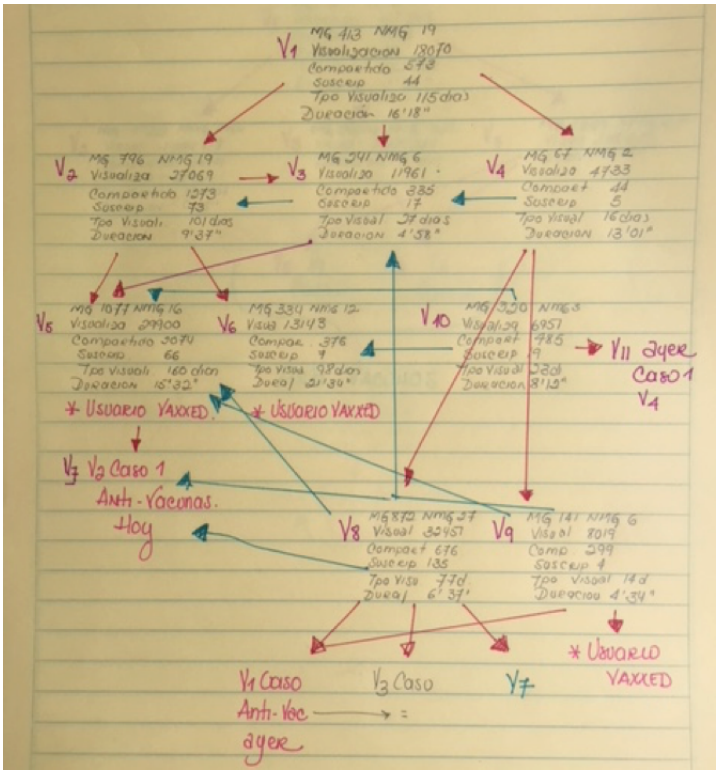


Figura 5: Maqueta de grafo dirigido

Como conclusiones del análisis realizado se definió el grafo a representar con las siguientes características:

- **Tipo:** Grafo dirigido sin pesos.
- **Nodos:** Vídeos y canales (canales de forma opcional).
- **Aristas:** Entre dos vídeos representa que desde el vídeo origen es posible acceder al vídeo destino (mediante la recomendación realizada por *YouTube* como vídeos relacionados). Y entre un vídeo y un canal representa que el vídeo (nodo origen) ha sido publicado por el canal relacionado (nodo destino).
- **Visualización:** En nodos tipo vídeo, su tamaño sera determinado por su alcance de audiencia y el color de representación sera determinado por su categoría (anti o pro vacuna). En el caso de nodos tipo canal su tamaño y representación no sera determinado por ninguna de sus características, por lo que todos los canales se visualizaran con el mismo formato pero diferenciados de los nodos tipo vídeo. Sera posible ver una pre visualización del contenido al hacer clic en el.

Para estudiar su viabilidad, se realizo una prueba de concepto con resultado favorable: POCYouTubeCrawler.

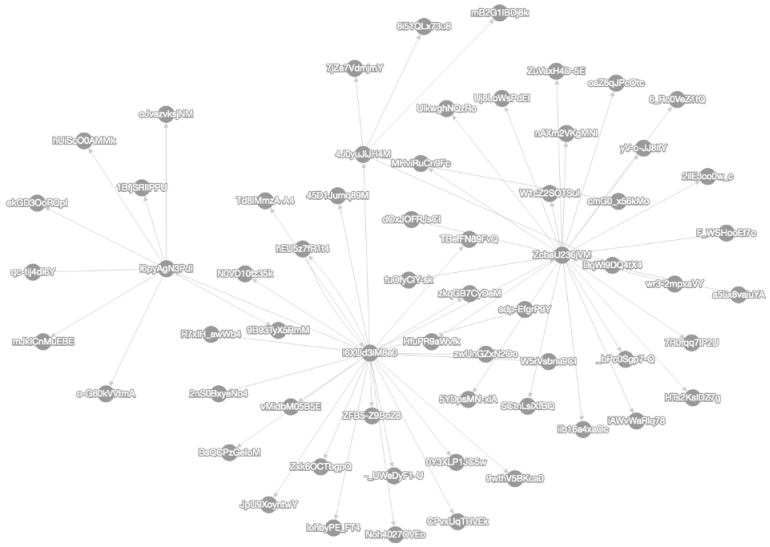


Figura 6: Prueba de concepto de grafo dirigido

En la propuesta de grafo detallada, se detectaron dos requerimientos encubiertos necesarios para su realización: la necesidad de categorizar vídeos y la definición de una variable que nos permitiera determinar el alcance de audiencia de un vídeo para definir su tamaño en el grafo.

Para la categorización de los vídeos, se diseño una funcionalidad genérica que permitiera al usuario definir las categorías necesarias con las cuales posteriormente poder etiquetar los vídeos. El inconveniente de esta solución es que para poder visualizar correctamente el grafo y identificar los grupos de nodos, se requiere realizar previamente una tarea de categorización manual de los vídeos. Después de analizar la problemática, debido a que el volumen de datos requeridos para realizar el estudio se determino en una cifra menor a mil, se acepto la solución como viable pero, en este caso, se debía tener presente esta circunstancia en la planificación del proyecto.

Por otro lado, para definir el tamaño de visualización del nodo se estudio la creación de una variable apodada como *scopeRange*. Dicha variable, debía representar dentro de un rango de valores valido, el alcance o popularidad obtenido por un vídeo en concreto. Para ello se estudio poder identificar a los usuarios mas influyentes de *YouTube* (conocidos como *influencers*), pero finalmente se decidió utilizar la información estadística ofrecida por *YouTube*

para cada vídeo, la cual se compone de los campos:

- ***viewCount*:** Numero de visualizaciones del vídeo.
- ***likeCount*:** Numero de personas a las cuales le ha gustado el vídeo.
- ***dislikeCount*:** Numero de personas a las cuales no le ha gustado el vídeo.
- ***commentCount*:** Numero de comentario que ha recibido el vídeo.

Para la definición de la formula se pidió la colaboración de un estadista. Pero debido a que la fecha final de desarrollo del proyecto se aproximaba y aun no se disponía de la colaboración, se opto por aplicar una formula definida por la clienta que en las pruebas realizadas demostró efectividad:

$$scopeRange = \frac{likeCount}{dislikeCount}$$

Con un valor mínimo definido de 10 para asegurar la visualización del vídeo en el grafo.

A modo ilustrativo, a continuación se adjunta una imagen de la implementación del grafo en la aplicación:

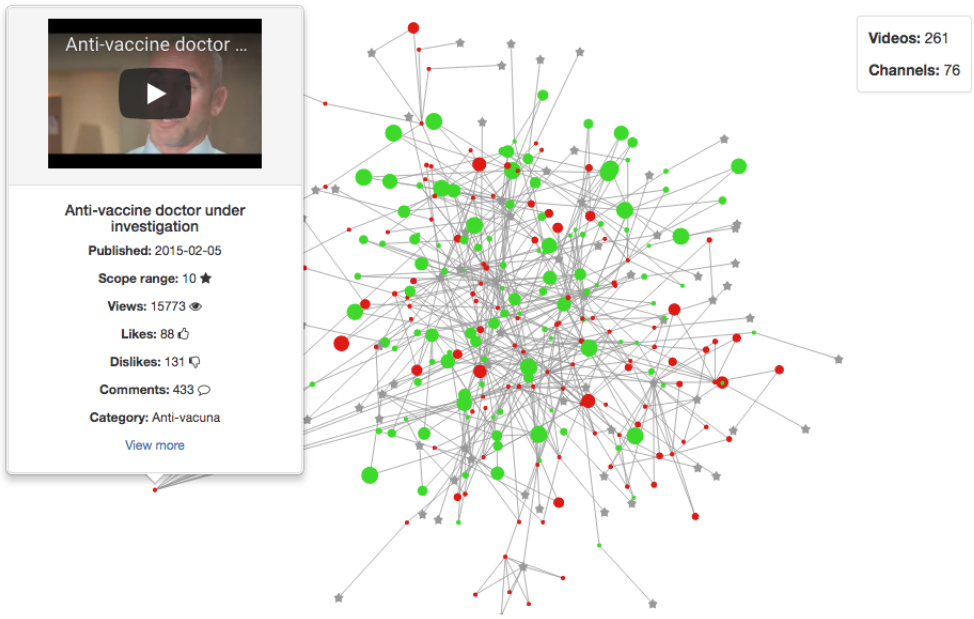


Figura 7: Implementación del grafo en la aplicación

2.2.4. Casos de uso

Hacer uso de un marco ágil de desarrollo no debe ser escusa para no disponer de una documentación adecuada de los requisitos de una aplicación y, aunque estos sufran modificaciones durante las diferentes iteraciones de desarrollo, estos deben de estar actualizados. Por esta razón se ha decidido realizar la toma de requerimientos de la aplicación en forma de casos de uso, en donde para facilitar la gestión de cambios se ha decidido hacer constar solo el titulo, actores y descripción en cada uno de ellos.

Así entonces y con lo expuesto en apartados anteriores, a continuación se expone una imagen representativa de los casos de uso identificados agrupados por componentes según la funcionalidad realizada y a continuación el listado completo de cada uno de ellos:

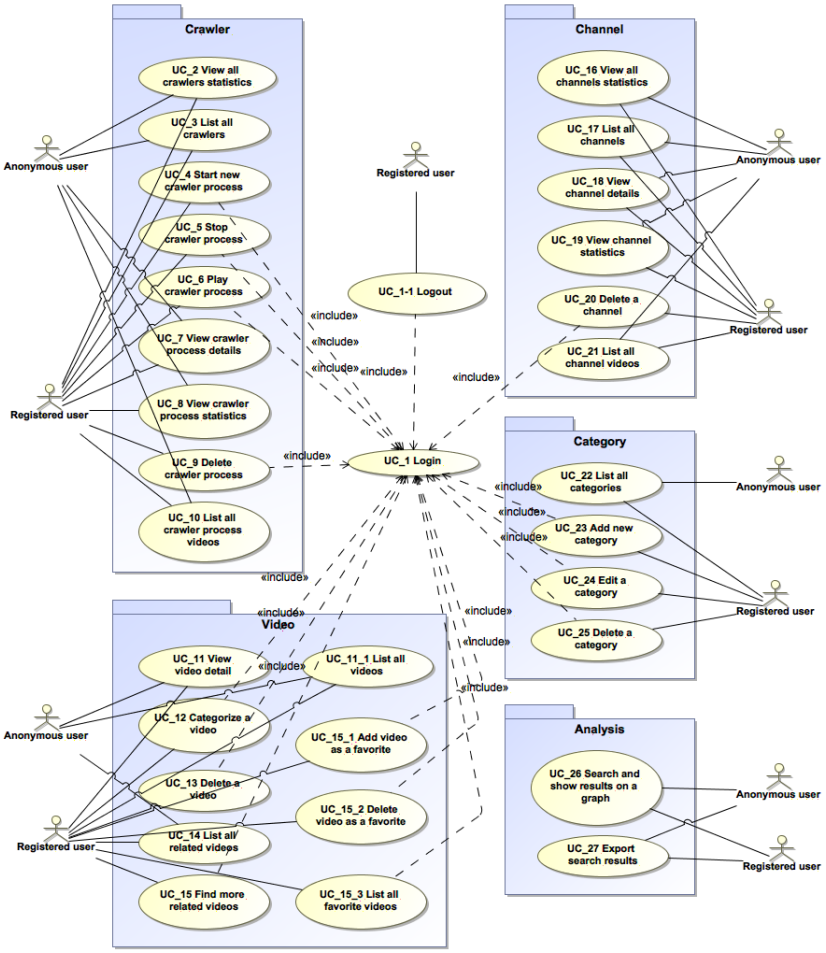


Figura 8: Casos de uso

Componente *User session*:

UC_1: Login
Actores: Usuario anónimo
Descripción: Como usuario anónimo, quiero identificarme en la aplicación mediante nombre de usuario y contraseña.

UC_1-1: logout
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero cerrar la sesión actual en la aplicación.

Componente *Crawler*:

UC_2: View all crawlers statistics
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver el total de vídeos recolectados, el porcentaje de categorizados y la distribución de las categorías.

UC_3: List all crawlers
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver todos los procesos de recolección que se han iniciado en el sistema.

UC_4: Start new crawler process
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero iniciar un nuevo proceso de recolección de vídeos según criterio de búsqueda introducido.

UC_5: Stop crawler process
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero parar un proceso de recolección de vídeos que este en ejecución.

UC_6: Play crawler process
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero poner en marcha un proceso de recolección previamente detenido.

UC_7: View crawler process details
Actores: Usuario anónimo, Usuario registrado

Descripción: Como usuario, quiero ver el detalle de un proceso de recolección.

UC_8: View crawler process statistics
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, dado un proceso de recolección en concreto quiero ver su total de vídeos recolectados, su porcentaje de categorizados y su distribución de las categorías.

UC_9: Delete crawler process
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero borrar un proceso de recolección en concreto y todos sus vídeos relacionados.

UC_10: List all crawler process videos
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver el listado completo de vídeos recolectados por un proceso de recolección en concreto.

Componente *Video*:

UC_11-1: List all videos
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver listados todos los vídeos del sistema.

UC_11: View video detail
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver el detalle de un vídeo en concreto.

UC_12: Categorize a video
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero cambiar la categoría asociada a un vídeo en concreto.

UC_13: Delete a video
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero borrar un vídeo en concreto.

UC_14: List all related videos
Actores: Usuario anónimo, Usuario registrado

Descripción: Como usuario, quiero ver listados todos los vídeos relacionados con un en concreto.

UC_15: Find more related videos
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero iniciar un nuevo proceso de recolección de vídeos para que encuentre vídeos relacionados a un vídeo en concreto.

UC_15-1: Add a video as a favorite
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero añadir un vídeo en concreto al listado de favoritos.

UC_15-2: Delete a video as a favorite
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero eliminar un vídeo en concreto del listado de favoritos.

UC_15-3: List all favorite videos
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero ver el listado de vídeos favoritos.

Componente *Channel*:

UC_16: View all channels statistics
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver el total de canales importados en el sistema.

UC_17: List all channels
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver listados todos los canales importado en el sistema.

UC_18: View channel details
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver el detalle de un canal en concreto.

UC_19: View channel statistics
Actores: Usuario anónimo, Usuario registrado

Descripción: Como usuario, dado un canal en concreto quiero ver su total de vídeos recolectados, su porcentaje de categorizados y su distribución de las categorías.

UC_20: Delete a channel
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero eliminar un canal en concreto y todos sus vídeos relacionados.

UC_21: List all channel videos
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, dado un canal en concreto quiero ver listados todos sus vídeos relacionados.

Componente *Category*:

UC_22: List all categories
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero ver todas las categorías dadas de alta en el sistema.

UC_23: Add new category
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero dar de alta una nueva categoría en el sistema definiendo su nombre y color.

UC_24: Edit a category
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero editar el color de una categoría en concreto.

UC_25: Delete a category
Actores: Usuario registrado
Descripción: Como usuario identificado, quiero eliminar una categoría en concreto del sistema y eliminar todos sus vídeos relacionados.

Componente *Analysis*:

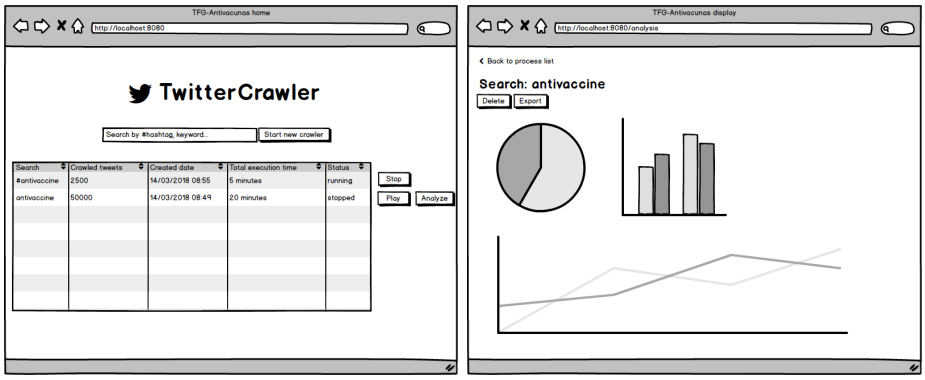
UC_26: Search and show results on a graph
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero realizar una búsqueda de vídeos en el sistema y visualizarlos en un grafo donde los nodos representen los vídeos encontrados y las aristas sus vídeos relacionados.

UC_27: Export search results
Actores: Usuario anónimo, Usuario registrado
Descripción: Como usuario, quiero realizar una búsqueda de vídeos en el sistema y exportarlos a fichero con formato *csv*.

2.2.5. Propuesta de interfaz de usuario

Finalmente, en la ultima etapa en lo que ha diseño del producto se refiere, se presentaron a la clienta diferentes propuestas de interfaz de usuario que se fueron refinando a medida que se modificaban o definían nuevas funcionalidades. En concreto se presentaron tres propuestas:

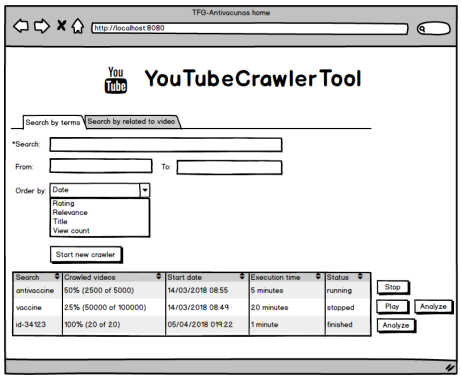
Propuesta inicial presentada el 15/03/2018:



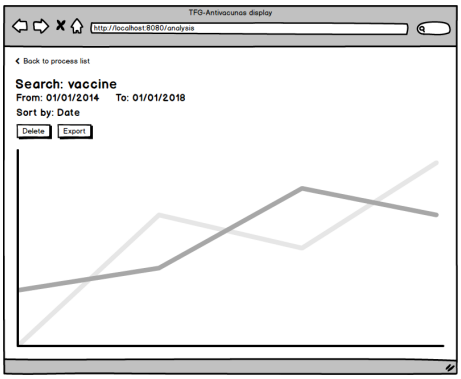
(a) *Twitter* búsqueda (b) *Twitter* análisis

Figura 9: Propuesta inicial interfaz de usuario

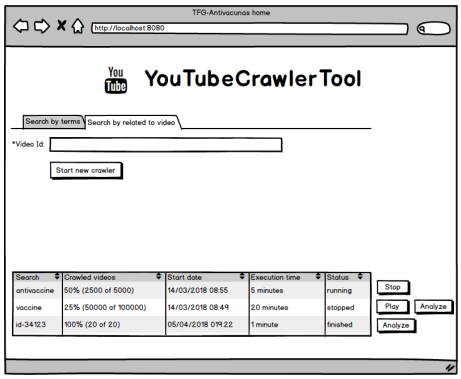
Segunda propuesta presentada el 05/04/2018:



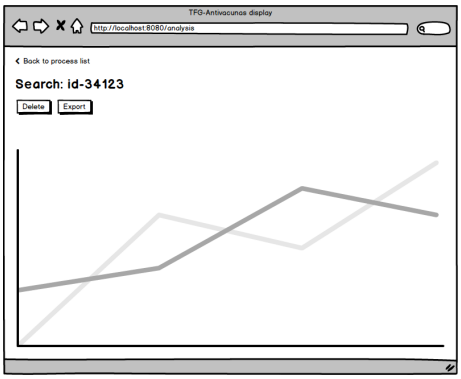
(a) *YouTube* búsqueda textual



(b) *YouTube* análisis textual



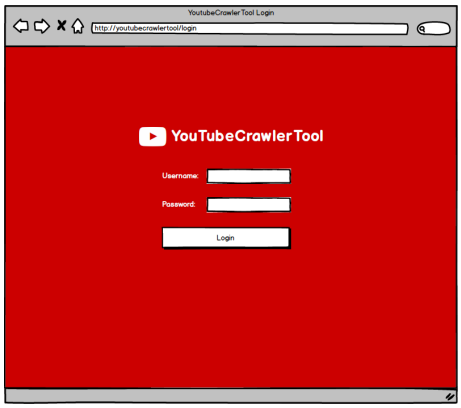
(c) *YouTube* búsqueda vídeo relacionado



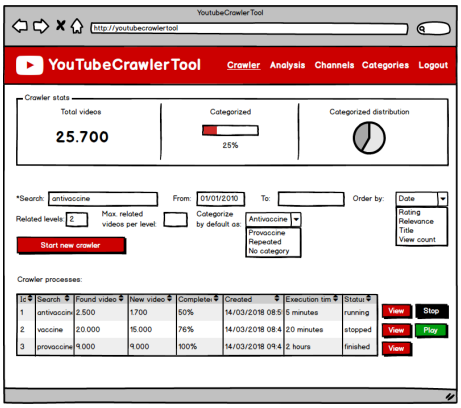
(d) *YouTube* análisis vídeo relacionado

Figura 10: Segunda propuesta interfaz de usuario

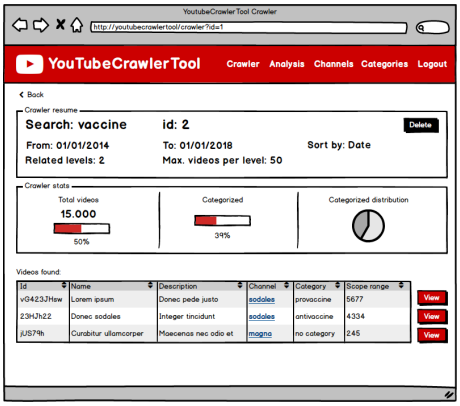
Tercera y ultima propuesta presentada el 15/04/2018:



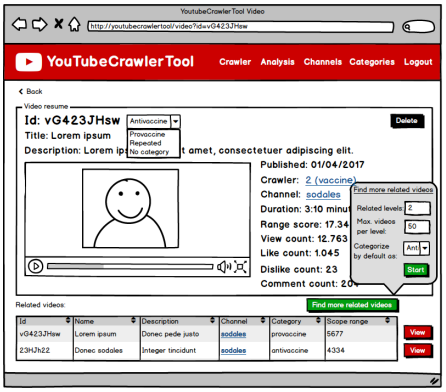
(a) Login



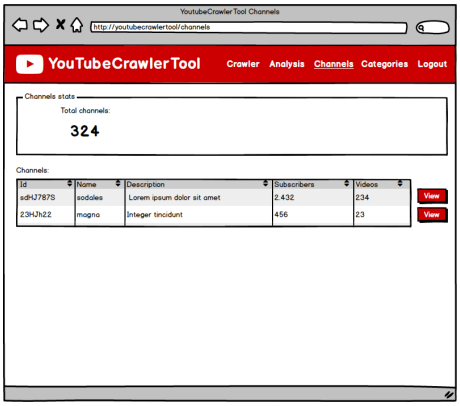
(b) Procesos de recolección



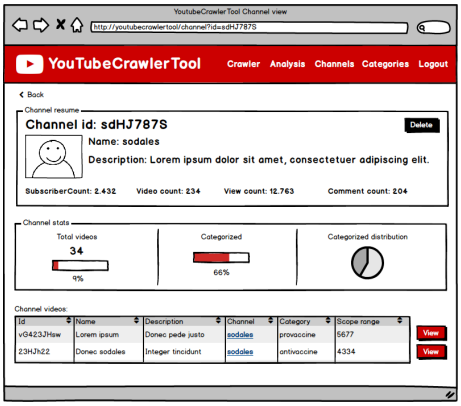
(c) Proceso de recolección



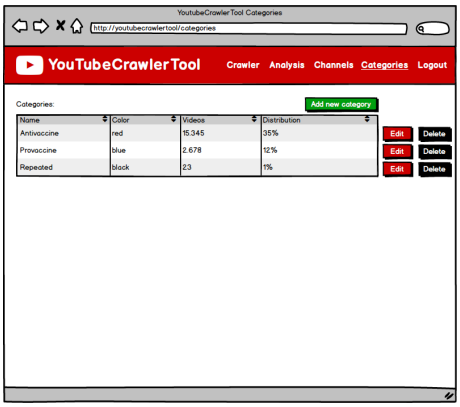
(d) Vista vídeo en detalle



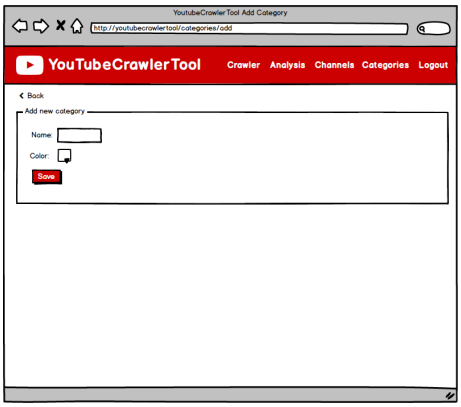
(e) Listado de canales



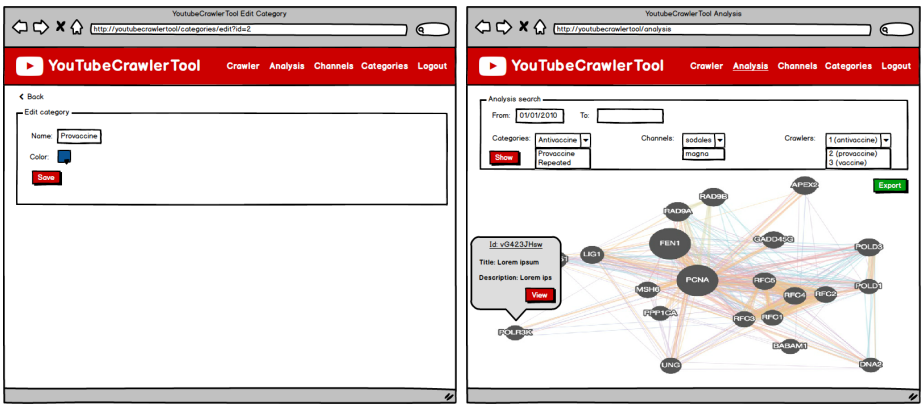
(f) Vista canal en detalle



(g) Listado de categorías



(h) Añadir nueva categoría



(i) Editar categoría (j) Vista de análisis

Figura 11: Tercera y ultima propuesta de interfaz de usuario

Cabe destacar que, debido a iteraciones de desarrollo realizadas posteriormente a la presentación y aprobación de la ultima propuesta de interfaz de usuario presentada, esta no refleja el estado final de la aplicación.

2.3. Arquitectura WEB

La decisión de realizar una aplicación Web viene motivada por las ventajas que aporta este medio en comparación con las aplicaciones de escritorio como por ejemplo, facilidades el trabajo colaborativo o centralizar la información y los procesos entre otros. Y de entre las diferentes arquitecturas Web, para el desarrollo del proyecto se escogió implementar una arquitectura REST por capas.

REST (proveniente del acrónimo en ingles *REpresentation State Transfer*) es un estilo arquitectónico en el desarrollo de aplicaciones Web que esta definido por una serie de principios. Algunos de los mas destacados son:

- Las solicitudes son sin estado.
- Se sirven recursos, no funcionalidades (tales como objectos de base de datos).
- Se accede a los recursos mediante URI que debe ser única por recurso.
- Las operaciones sobre los recursos se realizan mediante HTTP especificando métodos estándar (tales como *GET*, *PUT*, *POST* y *DELETE*).
- Como formato para el intercambio de recursos se suele utilizar JSON o XML.

La elección de esta arquitectura es a razón de los beneficios inherentes que aporta, tales como simplicidad en la arquitectura, escalable, extensible o la separación entre la capa de aplicación y capa de presentación. Gracias a esta arquitectura y de ser requerido en el futuro, ademas de disponer de un cliente Web como el diseñado en el proyecto actual, también seria factible, por ejemplo, proveer una aplicación móvil nativa sin tener que cambiar los servicios ofrecidos por el sistema.

Como alternativa al uso de REST se considero utilizar una arquitectura basada en RPC. La gran diferencia entre ambas (entre otras) reside en el hecho que REST se centra en recursos mientras que RPC en funcionalidades. Por facilidad de uso y simplicidad se escogió implementar una arquitectura REST en detrimento de RPC.

La aplicación de los principios REST conllevo a realizar otra decisión arquitectónica, dividir la aplicación por capas. En este caso, se escogió aplicar una arquitectura cliente-servidor de tres capas divididas en:

- **Capa de datos**: Donde se persistirá la información de la aplicación utilizando un SGBD.
- **Capa de aplicación**: En donde se llevara a cabo el acceso a datos, la lógica de negocio y de presentación que, a la vez, estará dividida entre el controlador de solicitudes de navegador (vistas HTML) y el controlador de solicitudes de recursos (API REST).
- **Capa cliente**: Consistente en una interfaz de usuario web.

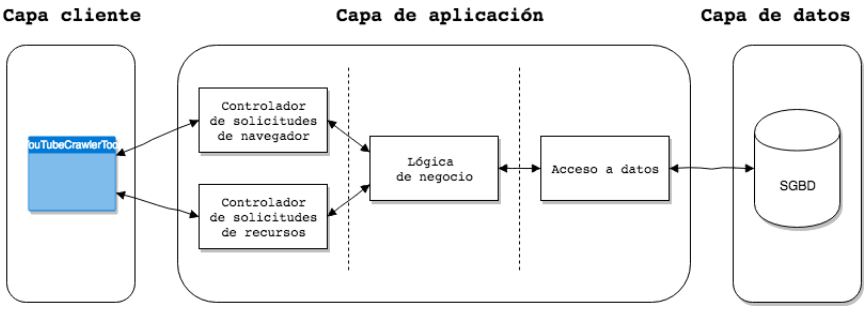


Figura 12: Arquitectura REST por capas

Gracias a esta distribución, se espera poder atorgar a la aplicación de una gran capacidad de reutilización y extensibilidad. En lo que ha distribución física se refiere, la capa cliente se ejecutara en los clientes Web de los usuarios y la capa de aplicación y de datos en un servidor. En los siguientes capítulos se detalla el diseño realizado para cada una de las capas.

2.4. Diseño capa de datos

En la capa de datos es donde se persistirá la información necesaria para el funcionamiento de la aplicación. Para tales efectos haremos uso de un sistema gestor de base de datos (SGBD) donde la implementación escogida de la misma se detalla en el apartado 3.2.2.

Para diseñar la capa de datos se proporciono un diagrama UML de clases del esquema invariante de la información:

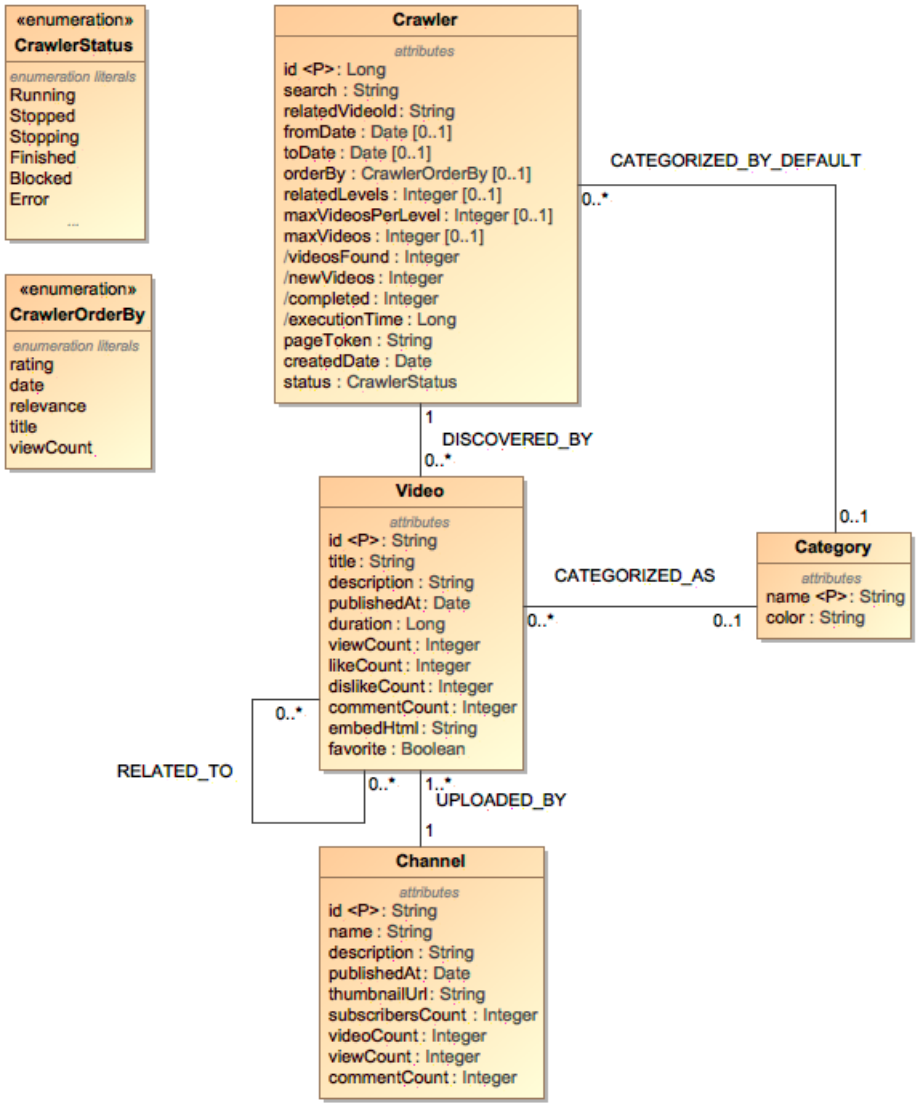


Figura 13: Diagrama de classes UML capa de datos

A continuación se detallan las entidades diseñadas:

- ***Crawler*:** Representa a un proceso de recolección de vídeos y contiene, ademas de los criterios de búsqueda introducidos por el usuario, información sobre el estado y resultado del proceso. Un proceso de recolección puede tener asignada una categoría por defecto y haber recolectado varios vídeos.
- ***Video*:** Representa un vídeo en la red social *YouTube*. Un vídeo puede tener asignada una categoría, ha sido recolectado por un proceso de recolección, ha sido publicado por un canal y puede estar relacionado con otros vídeos.
- ***Channel*:** Representa a un canal de publicación de vídeos en la red social *YouTube*. Un canal puede haber publicado varios vídeos.
- ***Category*:** Representa una categoría creada por el usuario de la aplicación. Una categoría puede etiquetar varios vídeos y puede estar asignada por defecto en varios procesos de recolección.

2.5. Diseño capa de aplicación

Para el diseño de la capa de aplicación se decidió definir un componente por cada grupo de funcionalidad identificada en los casos de uso definidos en el apartado 2.2.4. En concreto, los componentes diseñados fueron los siguientes:

- ***Crawler*:** Agrupa la funcionalidad del recolector de vídeos.
- ***Video*:** Agrupa la funcionalidad relacionada con la entidad *video*.
- ***Channel*:** Agrupa la funcionalidad relacionada con la entidad *channel*.
- ***Category*:** Agrupa la funcionalidad relacionada con la entidad *category*.
- ***Analysis*:** Agrupa la funcionalidad relacionada con la visualización en formato de grafo y exportación a fichero csv.

En donde en cada capa de la aplicación podemos encontrar representado el diseño realizado de cada componente.

A continuación se facilita la figura donde se representa el diagrama de componentes de la aplicación al completo y en próximos apartados se detallan cada una de las capas juntos con los diferentes diagrama refinados:

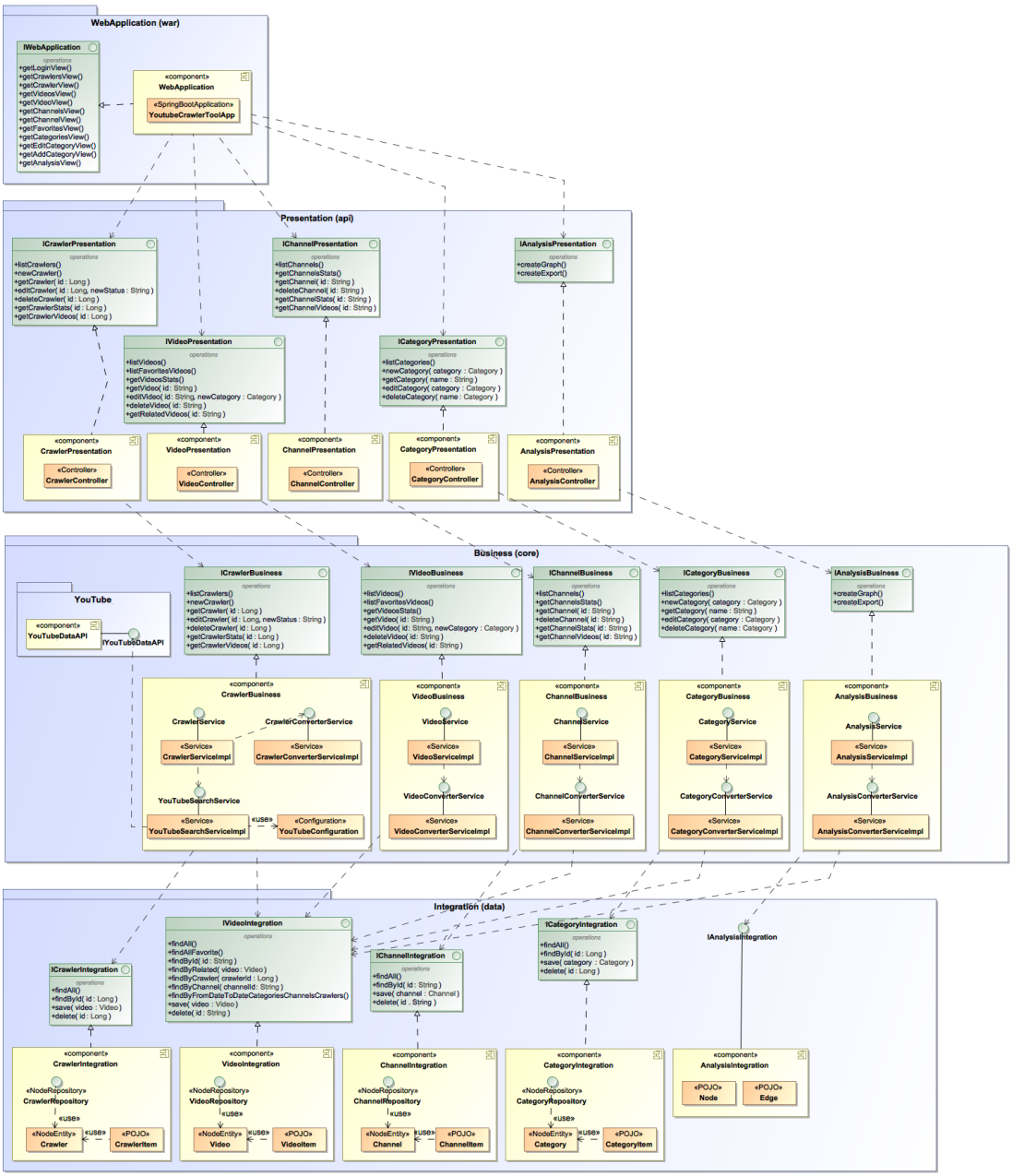


Figura 14: Diagrama de componentes

2.5.1. Capa de acceso a datos

Esta capa es la encargada de proporcionar acceso a la aplicación a los datos realizando y gestionando las conexiones y peticiones al SGDB.

Definición de componentes

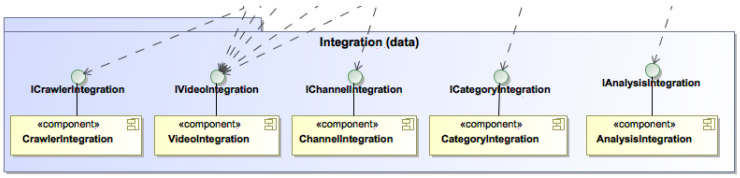


Figura 15: Diagrama de componentes capa acceso a datos - componentes

Definición de interfaces

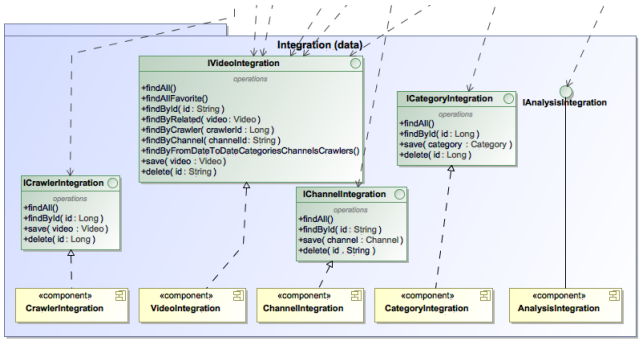


Figura 16: Diagrama de componentes capa acceso a datos - interfaces

Definición de implementaciones

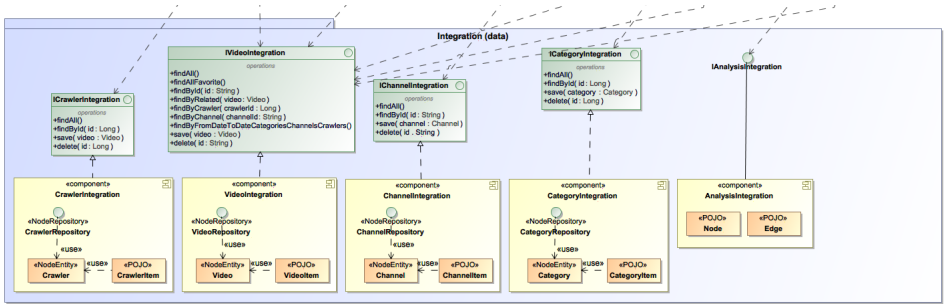


Figura 17: Diagrama de componentes capa acceso a datos - implementaciones

2.5.2. Capa de lógica de negocio

La capa de lógica de negocio es la encarga de actuar como mediadora entre la capa de acceso a datos y la capa de presentación añadiendo la lógica necesaria sobre los datos que los requerimientos de la aplicación necesiten. Es en esta capa también, donde se realizara la conexión con el servicio externo en forma de API de *YouTube* para obtener los vídeos.

Definición de componentes

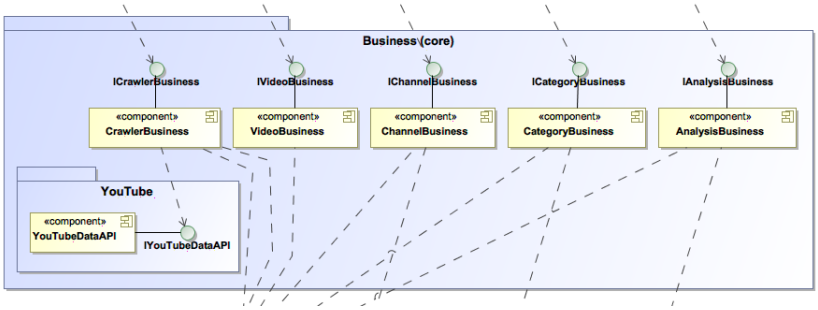


Figura 18: Diagrama de componentes capa de lógica de negocio - componentes

Definición de interfaces

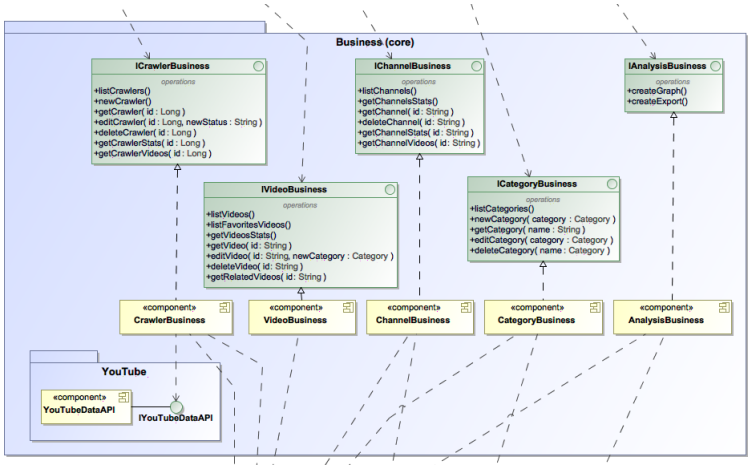


Figura 19: Diagrama de componentes capa de lógica de negocio - interfaces

Definición de implementaciones

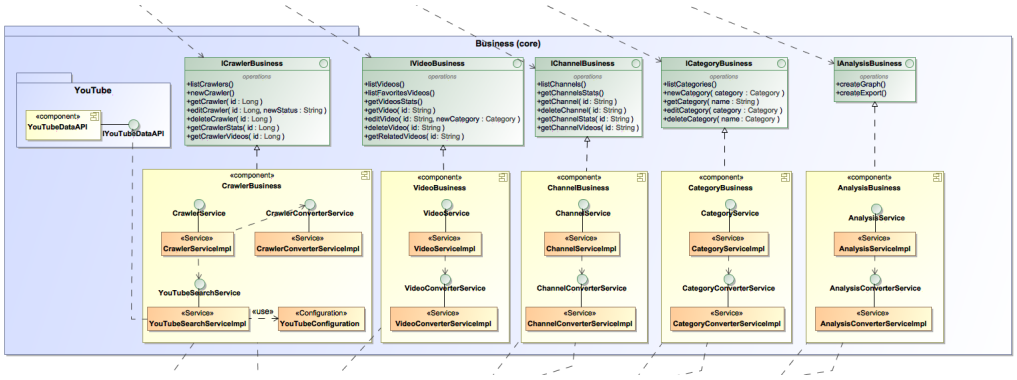


Figura 20: Diagrama de componentes capa de lógica de negocio - implementaciones

2.5.3. Capa de presentación

En la capa de presentación es donde habitan el controlador de solicitudes de recursos (paquete '*Presentation (api)*') y el controlador de solicitudes de navegador (paquete '*WebApplication (war)*') aplicando el patrón modelo vista controlador (MVC) [19].

Definición de componentes

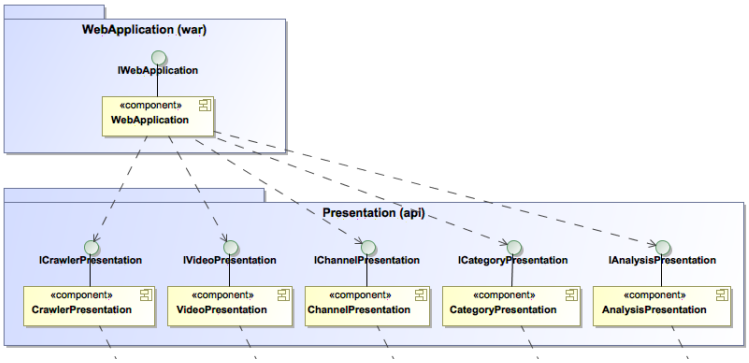


Figura 21: Diagrama de componentes capa de presentación - componentes

Definición de interfaces

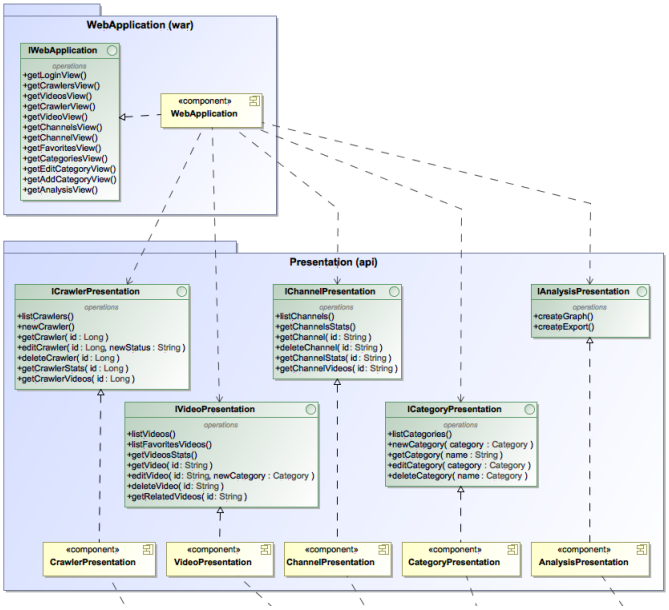


Figura 22: Diagrama de componentes capa de presentación - interfaces

Definición de implementaciones

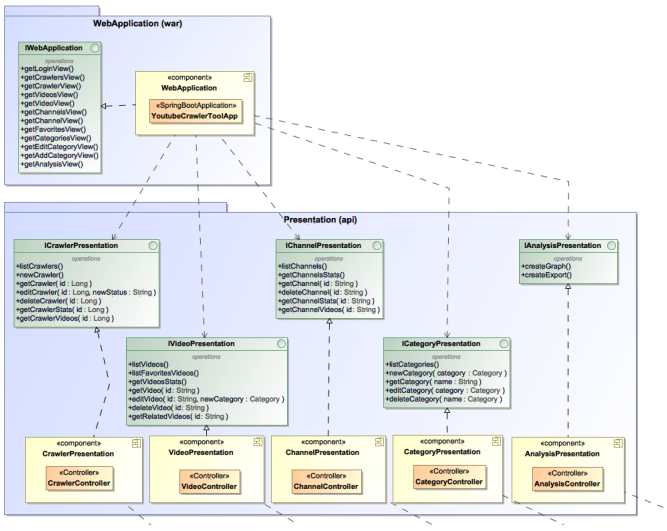


Figura 23: Diagrama de componentes capa de presentación - implementaciones

2.5.4. API REST

A continuación se facilita el diseño realizado de la API REST de la aplicación detallando la URI del recurso, el método de ejecución y una breve descripción.

La API se ha diseñado respetando los patrones de diseño definidos por la arquitectura REST [20]:

Componente *crawler*:

Metodo	**Uri**	**Descripción**
GET	api/crawlers	Lista todos los procesos de recolección.
POST	api/crawlers	Inicia un nuevo proceso de recolección. Requiere autentificación.
GET	api/crawlers/{id}	Devuelve un proceso de recolección.
PUT	api/crawlers/{id}	Editar un proceso de recolección. Requiere identificación.
DELETE	api/crawlers/{id}	Borra un proceso de recolección. Requiere identificación.
GET	api/crawlers/{id}/stats	Devuelve las estadísticas de un proceso de recolección.
GET	api/crawlers/{id}/videos	Lista todos los vídeos descubiertos por un proceso de recolección.

Cuadro 1: Componente *crawler* API

Componente *video*:

Metodo	**Uri**	**Descripción**
GET	api/videos	Lista todos los vídeos.
GET	api/videos/stats	Devuelve las estadísticas de todos los vídeos.
GET	api/videos/favorites	Devuelve los vídeos favoritos.
GET	api/videos/{id}	Devuelve un vídeo.
PUT	api/videos/{id}	Editar un vídeo. Requiere identificación.
DELETE	api/videos/{id}	Borra un vídeo. Requiere identificación.
GET	api/videos/{id}/videos	Lista todos los vídeos relacionados con un vídeo.

Cuadro 2: Componente *video* API

Componente *channel*:

Metodo	**Uri**	**Descripción**
GET	api/channels	Lista todos los canales
GET	api/channels/{id}	Devuelve un canal.
DELETE	api/channels/{id}	Borra un canal. Requiere identificación
GET	api/channels/{id}/stats	Devuelve la estadísticas de un canal
GET	api/channels/{id}/videos	Lista todos los vídeos relacionados con un canal.

Cuadro 3: Componente *channel* API

Componente *category*:

Metodo	Uri	Descripción
GET	api/categories	Lista todas las categorías
POST	api/categories	Crea una nueva categoría. Requiere identificación.
GET	api/categories/{name}	Devuelve una categoría.
PUT	api/categories/{name}	Editar una categoría. Requiere identificación.
DELETE	api/categories/{name}	Borra una categoría.
GET	api/categories/{name}/stats	Devuelve las estadísticas de una categoría.
GET	api/categories/{name}/videos	Lista todos los vídeos categorizados por una categoría.

Cuadro 4: Componente *category* API

Componente *analysis*:

Metodo	Uri	Descripción
POST	api/graphs	Crea un nuevo grafo (no se persiste, se muestra por pantalla).

Cuadro 5: Componente *analysis* API

2.6. Diseño capa cliente

Finalmente, para el diseño de la capa cliente correspondiente a la interfaz de usuario de la aplicación web, se especificaron las vistas de acuerdo con los casos de uso definidos en el apartado 2.2.4 y a la propuesta de interfaz de usuario aprobada por la clienta 2.2.5:

- ***loginView.html*:** Permite la identificación en el sistema.
- ***crawlersView.html*:** Pagina principal de la aplicación. Lista los procesos de recolección existentes y permite gestionarlos y iniciar nuevos procesos.
- ***crawlerView.html*:** Vista detalle de un proceso de recolección.
- ***videosView.html*:** Listado de todos los vídeos del sistema.
- ***videoView.html*:** Vista detalle de un vídeo.
- ***channelsView.html*:** Lista de todos los canales del sistema.
- ***channelView.html*:** Vista detalle de un canal.
- ***analysisView.html*:** Permite visualizar los vídeos en un grafo y exportarlos a fichero csv.
- ***favoriteVideosView.html*:** Listado de todos los vídeos favoritos.

- ***categoriesView.html*:** Lista todas las categorías del sistema.
- ***addCategoryView.html*:** Permite crear una nueva categoría.
- ***editCategoryView.html*:** Permite editar una categoría.

Dichas vistas serán gestionadas por el controlador de solicitudes de navegador y en ellas, se realizaran peticiones asíncronas a recursos utilizando la API REST de la aplicación que sera gestionada por el controlador de solicitudes de recursos.

La figura a continuación ejemplifica el mapa web de la aplicación con los flujos de interacción posibles y funcionalidades previstas para cada vista:

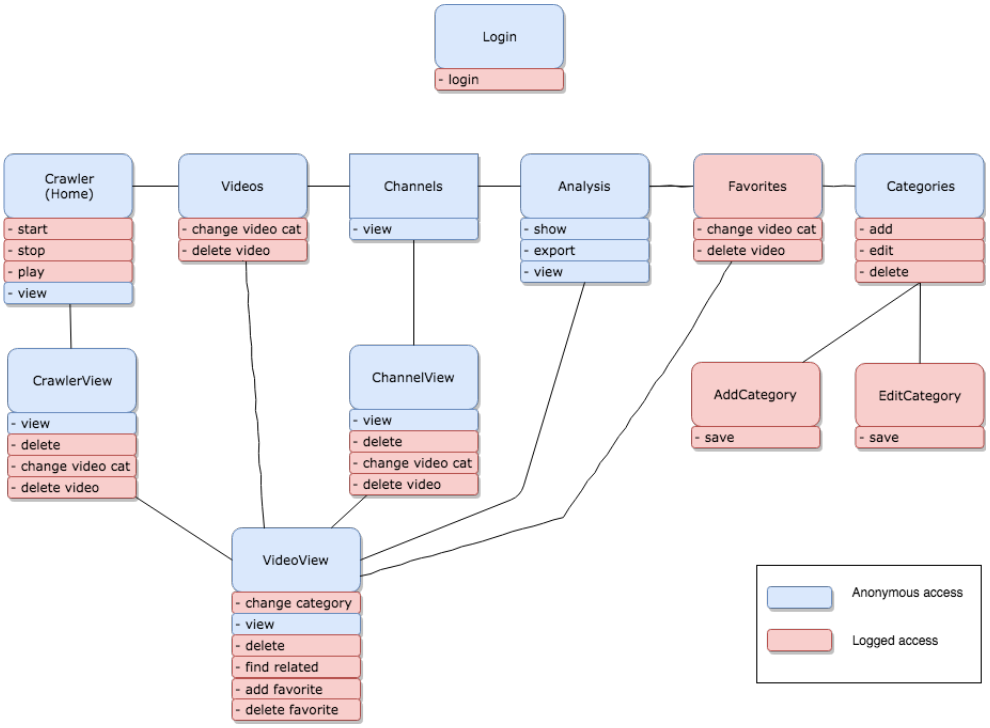


Figura 24: Diseño vistas capa cliente

3. Desarrollo

El código fuente de la aplicación desarrollada la cual se detalla a continuación, se encuentra disponible en un repositorio publico de *GitHub*: YouTubeCrawlerTool.

3.1. Entorno de desarrollo

A continuación se detalla la configuración del entorno de desarrollo utilizada en el desarrollo del proyecto. Algunas de las cuales no representan obligatoriedad siendo posible utilizar alternativas diferentes con los mismos resultados, en este caso solo se pretende exponer la configuración utilizada:

- **Sistema operativo:** MacOS.
- **Neo4J 3.3.5 Server Community version:** Instalación de servidor de base de datos Neo4J en local necesario para el desarrollo de la capa de datos.
- **Java JDK 8:** Kit de desarrollo de Java versión 8 [21] necesario para el desarrollo de la capa de aplicación.
- **Git y creación de cuenta en GitHub:** Como herramienta para el control de versiones se ha escogido Git [22]. Ademas, se creo una cuenta con perfil público en Github accesible desde el siguiente enlace: `https://github.com/jsanchezmend/TFGAntivacunas`.
- **Maven:** Herramienta para la gestión de proyectos Java [23], fue utilizada para la construcción del proyecto y para la gestión de dependencias.
- **Spring Tool Suite:** Entorno de desarrollo utilizado el cual se basa en una customización de Eclipse [24].
- **Tomcat:** Para probar la aplicación se utilizo el servidor de aplicaciones Tomcat [25], pero devido al uso de Spring Boot [26] no fue necesario realizar su instalación al incorporar esta solución una versión de Tomcat.
- **Google API Key:** Para poder realizar consultas a la API de *YouTube* es necesario obtener una clave de API que tendrá que ser utilizada cada vez que se haga uso de sus servicios. Para obtenerla, primero hay que acceder a la consola de desarrollo de *Google* utilizando una cuenta previamente existente, crear un nuevo proyecto de desarrollo y activar los servicios de *YouTube Data API v3* [27].

3.2. Decisiones previas al desarrollo

Antes de empezar con el desarrollo de la aplicación, fue necesario realizar algunas decisiones previas las cuales tendrían repercusión en la implementación final de la solución. Para ayudar en la toma de decisión y estudiar la viabilidad de las mismas, estas fueron respaldadas con pruebas de concepto detalladas en la sección 3.8.

En los próximos apartados se detallan algunas de las decisiones mas relevantes que se tuvieron que tomar, como fue el estudio y elección de la API de YouTube a consumir y el SGDB a utilizar.

3.2.1. YouTube Data API

Para ofrecer sus servicios a terceros, *YouTube* divide su funcionalidad en cuatro grupos de APIs distintas dentro de la plataforma *Google APIs*:

- ***YouTube Data API v3*:** Proporciona acceso y servicios de búsqueda a recursos tales como vídeos, canales, comentarios entre otros.
- ***YouTube Analytics API*:** Permite obtener información analítica sobre la cuenta de un usuario, tales como número de suscripciones o número de visualizaciones entre otros. Este servicio esta limitado a uso personal, siendo solo posible acceder a la información del propio usuario que realiza la petición y no es permitido realizar consultas sobre otros usuarios.
- ***YouTube Reporting API*:** Complementaria a la *YouTube Analytics API*, esta API permite al usuario de *YouTube* programar la generación de informes sobre su información analítica. De igual forma que su antecesora, su eso esta limitado al uso personal.
- ***YouTube Ad Reach API*:** Dedicada a anunciantes, permite el estudio del alcance obtenido por campaña de publicidad.

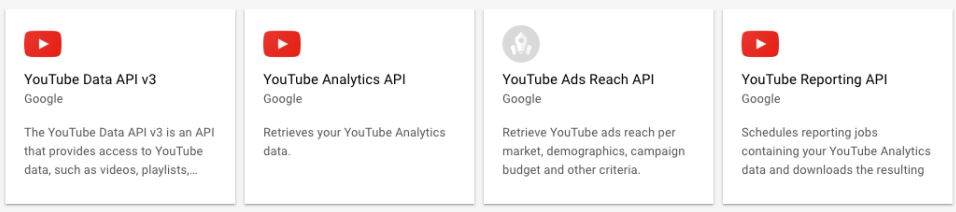


Figura 25: *YouTube* APIs

Así entonces y según lo expuesto, para poder realizar búsquedas y obtener los contenidos que sean requeridos por el usuario, la aplicación hara uso de los servicios web expuestos por la *YouTube Data API v3* [29].

Dichos servicios se dividen por tipos de recursos a los cuales acceder, en donde cada recurso representa un tipo de contenido de *YouTube* tales como vídeos, canales, listas de reproducción entre otros. Para poder obtener dichos recursos primeramente es necesario conocer los identificadores de los objectos que se quieren obtener, en donde para tales efectos hay habilitado un servició el propósito del cual es la obtención de dichos identificadores. Las únicas operaciones que se realizan en la aplicación son de lectura de recursos, no se requiere realizar operaciones de escritura. Así entonces, el modo en que nuestra aplicación utiliza estos servicios es de la siguiente forma:

- **1- Obtención de identificadores de vídeos:** Se realiza una búsqueda para obtener los identificadores de los vídeos que se correspondan con los criterios de búsqueda introducidos por el usuario. Los resultados obtenidos, dependiendo del volumen de coincidencias que la búsqueda haya producido, podrán ser paginados requiriendo realizar peticiones iterativas para obtener los consiguientes resultados. El servició utilizado es el de *'Seach'* y concretamente la operación *'list'* [30].
- **2- Obtención de vídeos:** Una vez obtenidos los identificadores de los vídeos que se quieren recuperar, se realiza una petición a la operación *'list'* del servicio *'Videos'* [31] con dichos identificadores para obtener finalmente su contenido. De igual forma que en la obtención de identificadores, las peticiones a este servició se realizaran de forma iterativa para recuperar los vídeos de forma paginada.
- **3- Obtención de canales:** Paralelamente a la obtención de vídeos para cada vídeo recuperado también se obtiene su canal. Para recuperarlo, de detectar que el canal en cuestión aun no se encuentra almacenado en el sistema, se realiza una petición al servicio *'Channels'* operación *'list'* [32] con el identificador del canal obtenido en la información recuperada del vídeo.

Como se ha comentado en el apartado 2.2.2, *YouTube* define un sistema de cuotas en el uso de sus servicios, en el cual cada petición realizada tiene un coste que puede variar dependiendo del tipo de recurso y la información solicitada. Para minimizar el uso de cuota es importante solicitar solo los recursos y los campos que realmente sean necesarios para el usuario. A continuación se detalla el uso realizado para cada tipo de petición:

Obtención de identificadores de vídeos
Petición HTTP: GET https://www.googleapis.com/youtube/v3/search

Criterios de búsqueda utilizados:

- ***type*:** Permite escoger el tipo de recurso a recuperar. Valor utilizado por defecto *'video'*.
- ***relevanceLanguage*:** Permite obtener resultados que son mas relevantes en el idioma introducido. Valor utilizado por defecto *'en'*.
- ***q*:** Se utiliza en caso de realizarse una búsqueda textual, permite buscar recursos que correspondan con el texto introducido. En el texto a buscar se pueden utilizar negaciones y conjunciones.
- ***relatedToVideoId*:** Alternativamente a la búsqueda textual, con este campo se puede realizar una búsqueda de contenidos que estén relacionados con el identificador solicitado.
- ***publishedBefore* y publishedAfter:** Permite acotar la búsqueda a realizar dentro de un rango de fechas determinado.
- ***order*:** Permite ordenar los resultados obtenidos por varios criterios: por fecha, rating, relevancia, titulo y numero de visualizaciones. Por defecto, los resultados son ordenados por relevancia.
- ***maxResults*:** Máximo de resultados obtenidos por pagina. El máximo valor aceptado es 50.
- ***pageToken*:** Identifica la pagina de resultados que debe ser devuelta. Como respuesta a la petición de búsqueda, juntamente con los recursos solicitados se devuelven los campos *'nextPageToken'* y *'prevPageToken'*, con la información devuelta en estos campos es posible navegar hacia la siguiente pagina de resultados o hacia la anterior.

Campos solicitados:

- ***id*:** Identificador de los vídeos encontrados.

Obtención de vídeos
Petición HTTP: GET https://www.googleapis.com/youtube/v3/videos

Criterios de búsqueda utilizados:

- ***id*:** Listado separado por comas de los identificadores de vídeos a recuperar.

Campos solicitados:

- ***id*:** Identificador de vídeo.
- ***snippet/title*:** Titulo del vídeo.
- ***snippet/description*:** Descripción del vídeo.
- ***snippet/publishedAt*:** Fecha de publicación del vídeo.
- ***snippet/channelId*:** Identificador del canal que ha publicado el vídeo.
- ***contentDetails/duration*:** Tiempo de duración del vídeo.
- ***statistics/viewCount*:** Número de visualizaciones del vídeo.
- ***statistics/likeCount*:** Número de 'me gusta' que ha recibido el vídeo.
- ***statistics/dislikeCount*:** Número de 'no me gusta' que ha recibido el vídeo.
- ***statistics/commentCount*:** Número de comentarios realizados sobre el vídeo.
- ***player/embedHtml*:** Reproductor del vídeo en un formato *iframe* html.

Obtención de canales
Petición HTTP: GET https://www.googleapis.com/youtube/v3/channels

Criterios de búsqueda utilizados:

- ***id*:** Identificador del canal recuperar.

Campos solicitados:

- ***id*:** Identificador del canal.
- ***snippet/title*:** Titulo del canal.
- ***snippet/description*:** Descripción del canal.
- ***snippet/publishedAt*:** Fecha de publicación del canal.
- ***snippet/customUrl*:** Dirección url del canal.
- ***snippet/thumbnails/default/url*:** Dirección url de la imagen representativa del canal.
- ***statistics/viewCount*:** Número de visualizaciones del canal.

- ***statistics/commentCount*:** Número de comentarios que ha recibido el canal.
- ***statistics/subscriberCount*:** Número de suscripciones al canal.
- ***statistics/videoCount*:** Número de vídeos publicados por el canal.

Antes de la implementación final de la solución, para probar los servicios aquí detallados se realizo con éxito la siguiente prueba de concepto: POCYouTubeCrawler.

3.2.2. Neo4j como SGBD

Como consecuencia de los requerimientos del proyecto y al futuro uso para el cual esta destinada la aplicación, se requiere el almacenamiento de grandes cantidades de información que pueden o no estar estructuradas y a las cuales se requiere una alta accesibilidad.

Por estos motivos, como gestor de bases datos se considero el uso de bases de datos NoSQL en vez de hacer uso de una base de datos relacional. Entre las principales características que ofrecen las bases de datos NoSQL que provocaron su elección en el proyecto destacamos las siguientes:

- **Manejo de grandes cantidades de datos:** Las bases de datos NoSQL al no garantizar completamente las características ACID (atomicidad, consistencia, aislamiento y durabilidad) [33], permiten el acceso a grandes cantidades de datos de forma mas eficiente que las bases de datos que si garantizan ACID.
- **Estructuras no fijas:** Para almacenar los datos no se requieren estructuras fijas como tablas, pueden almacenarse todo tipo de contenidos sin especificar previamente su estructura.
- **Escalabilidad:** Ofrecen buena escalabilidad horizontal que nos permitiría augmentar su capacidad fácilmente.
- **Operaciones de lectura:** Debido a sus características, las bases de datos NoSQL son especialmente idóneas en las operaciones de lectura, las cuales son las que se quieren potenciar en la aplicación.

Dentro de las bases de datos NoSQL existen subcategorías según la forma en la cual se almacenan los datos. Algunas de ellas son:

- **Bases de datos clave-valor:** Almacenan conjuntos donde a cada clave se le asigna un valor. Algunas de sus principales implementaciones son *Cassandra* [34] o *BigTable* de *Google*.

- **Bases de datos documentales:** Almacenan información en formato de documentos que responden a algún formato en concreto. Algunas de sus principales implementaciones son *MongoDB* [35] y *CouchDB* [36].
- **Bases de datos orientadas a grafos:** La información se almacena en formato de grafo en donde la información queda almacenada cono nodo y sus relaciones como aristas. Actualmente la implementacion mas extendida es *Neo4j* [37].

De entre las diferentes opciones, inicialmente se eligió utilizar una base de datos documental y, concretamente, el uso de la implementación ofrecida por *MongoDB*.

Esta decisión fue motivada, en su momento, después de analizar la información que era necesaria persistir en la aplicación en donde los contenidos a recuperar respondían a una estructura de documento. Por su lado, la elección de *MongoDB* como implenmentación de base de datos documental venia motivada principalmente por gozar de alta aceptación y de disponer una amplia documentación. Ademas, al trabajar *MongoDB* con documentos en formato *JSON* facilitaba su tratamiento y posterior presentación en la aplicación. Por estas razones, las dos primera pruebas de concepto desarrolladas se llevaron a cabo con *MongoDB*: POCTwitterCrawler y POCYouTubeCrawler.

Posteriormente y en etapas mas avanzadas del proyecto, la visualización de los contenidos recolectados en formato de grafo cobro mas importancia dentro de la aplicación, este hecho provoco la consideración del uso de bases de datos orientadas a grafos [38]. Entre las principales ventajas en su uso, destaca el poder aplicar teoría de grafos en las consultas a realizar y de esta forma recorrer las relaciones entre registros como si de un grafo se tratara para, por ejemplo, encontrar el camino mas corto entre dos vídeos que están relacionados mediante otros vídeos.

Por lo tanto, el uso de una base de datos orientada a grafo nos permitiría añadir capacidades analíticas al grafo obtenido en la aplicación que serian difíciles de proporcionar con otro tipo de base de datos, habilitando que en un futuro se puedan añadir mas funcionalidades al grafo para facilitar su estudio. Ademas, al estar orientada a grafos nos permite realizar consultas de forma mas natural a la hora de obtener la información a representar.

Y como implementación de base de datos orientada a grafo, se escogió el uso de *Neo4j* al ser esta la solución mas extendida actualmente. Para poner a prueba su uso en la aplicación se adaptaron las pruebas de concepto realizadas anteriormente con *MongoDB* a *Neo4j* con resultado favorable:

POCYouTubeCrawlerNeo4j

Debido a sus características, para habilitar su uso en la aplicación no se requirió modificar el diseño de la capa de datos detallado en el apartado 2.4, al poderse realizar la de forma natural la transformación de entidad a nodo y de relación a aristas tal y como se ilustra en la siguiente imagen:

Figura 26: Diseño capa de datos con nodos y aristas

Como curiosidad, *Neo4j* define su propio lenguaje de consultas llamado *Cypher* y proporciona una consola de administración web donde realizar consultas y visualizar los resultados en formato de grafo entre otros:

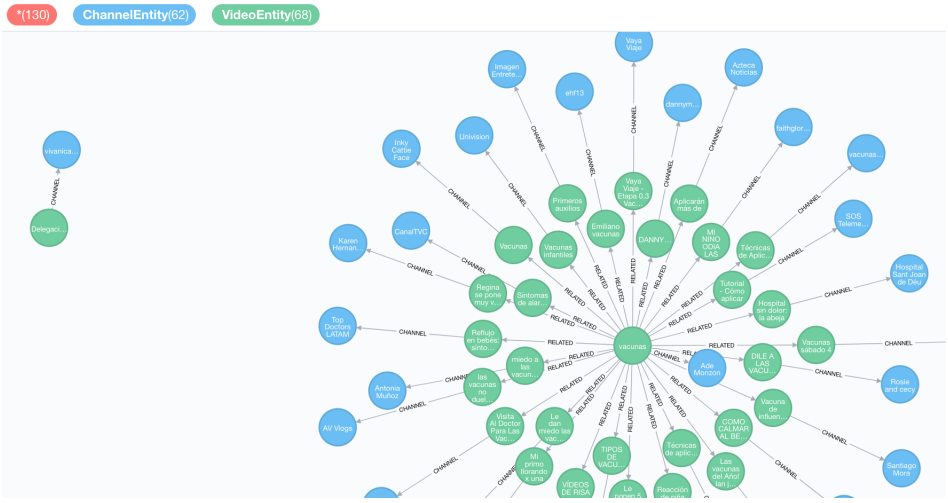


Figura 27: Consola de administración *Neo4j*

3.3. Estructura de la aplicación

Tal y como se ha avanzado anteriormente, para la creación de la estructura del proyecto en *Java* se hizo uso de la herramienta *Maven* (o conocido popularmente como *mvn*).

Con esta herramienta se creo la definición del proyecto juntamente con sus módulos utilizando los *Project Object Model* (definidos mediante ficheros de configuración '*pom.xml*') con los cuales es posible definir la estructura de la aplicación a construir. Ademas, *Maven* nos permite gestionar las dependencias a otros módulos o librerías de terceros añadiendo versionado a nuestra aplicación.

Así entonces, utilizando *Maven* se definió la estructura de la aplicación separada por módulos independientes tomando como referencia la separación por paquetes realizada en el diseño de la capa de aplicación de la sección 2.5:

- **youtube-crawler-tool:** Modulo padre de la aplicación.
- **youtube-crawler-tool-data:** Implementación de la capa de acceso a datos.
- **youtube-crawler-tool-core:** Implementación de la capa de lógica de negocio.
- **youtube-crawler-tool-api:** Implementación de la capa de presentación.
- **youtube-crawler-tool-war:** Implementación de la capa cliente.

Gracias a esta separación por módulos, la solución implementada puede ser fácilmente ampliada para, por ejemplo, definir una capa de acceso a datos alternativa que utilice una tecnología de persistencia diferente.

Figura 28: Estructura de la aplicación

A seguir se facilitan algunos extractos de los ficheros "*pom.xml*" definidos para cada modulo:

youtube-crawler-tool

```
...
<groupId>uoc.edu.jsanchezmend.tfg</groupId>
<artifactId>youtube-crawler-tool</artifactId>
<version>1.0.0</version>
<packaging>pom</packaging>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.1.RELEASE</version>
</parent>

<!-- Modules -->
<modules>
  <module>youtube-crawler-tool-data</module>
  <module>youtube-crawler-tool-core</module>
  <module>youtube-crawler-tool-api</module>
  <module>youtube-crawler-tool-war</module>
</modules>
...
```

youtube-crawler-tool-data

```
...
<artifactId>youtube-crawler-tool-data</artifactId>
<packaging>jar</packaging>

```

```
<parent>
  <groupId>uoc.edu.jsanchezmend.tfg</groupId>
  <artifactId>youtube-crawler-tool</artifactId>
  <version>1.0.0</version>
</parent>

<dependencies>
  <!-- Spring Data Neo4j -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-neo4j</
       artifactId>
  </dependency>
  <!-- Json parser -->
  <dependency>
    <groupId>com.google.http-client</groupId>
    <artifactId>google-http-client-jackson2</artifactId>
    <version>1.23.0</version>
  </dependency>
</dependencies>
...
```

youtube-crawler-tool-core

```
...
<artifactId>youtube-crawler-tool-core</artifactId>
<packaging>jar</packaging>

<parent>
  <groupId>uoc.edu.jsanchezmend.tfg</groupId>
  <artifactId>youtube-crawler-tool</artifactId>
  <version>1.0.0</version>
</parent>

<dependencies>
  <!-- YouTubeCrawlerTool data -->
  <dependency>
    <groupId>uoc.edu.jsanchezmend.tfg</groupId>
    <artifactId>youtube-crawler-tool-data</artifactId>
    <version>${parent.version}</version>
  </dependency>
  <!-- YouTube Data V3 support -->
  <dependency>
    <groupId>com.google.apis</groupId>
    <artifactId>google-api-services-youtube</artifactId>
    <version>v3-rev193-1.23.0</version>
  </dependency>
</dependencies>
```

youtube-crawler-tool-api

```
...
<artifactId>youtube-crawler-tool-api</artifactId>
<packaging>jar</packaging>

<parent>
  <groupId>uoc.edu.jsanchezmend.tfg</groupId>
  <artifactId>youtube-crawler-tool</artifactId>
  <version>1.0.0</version>
</parent>

<dependencies>
  <!-- YouTubeCrawlerTool core -->
  <dependency>
    <groupId>uoc.edu.jsanchezmend.tfg</groupId>
    <artifactId>youtube-crawler-tool-core</artifactId>
    <version>${parent.version}</version>
  </dependency>
  <!-- Spring Boot Web -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <!-- Spring Security -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
</dependencies>
...
```

youtube-crawler-tool-war

```
...
<artifactId>youtube-crawler-tool-war</artifactId>
<packaging>war</packaging>

<parent>
  <groupId>uoc.edu.jsanchezmend.tfg</groupId>
  <artifactId>youtube-crawler-tool</artifactId>
  <version>1.0.0</version>
</parent>

<dependencies>
  <!-- YouTubeCrawlerTool api -->
  <dependency>
    <groupId>uoc.edu.jsanchezmend.tfg</groupId>
```

```
15      <artifactId>youtube-crawler-tool-api</artifactId>
16      <version>${parent.version}</version>
17    </dependency>
18    <!-- Spring Boot Thymeleaf -->
19    <dependency>
20      <groupId>org.springframework.boot</groupId>
21      <artifactId>spring-boot-starter-thymeleaf</artifactId
            >
22    </dependency>
23    <!-- optional, it brings userful tags to display spring
          security stuff -->
24    <dependency>
25      <groupId>org.thymeleaf.extras</groupId>
26      <artifactId>thymeleaf-extras-springsecurity4</
            artifactId>
27    </dependency>
28    <!-- hot swapping, disable cache for template, enable
         live reload -->
29    <dependency>
30      <groupId>org.springframework.boot</groupId>
31      <artifactId>spring-boot-devtools</artifactId>
32      <optional>true</optional>
33    </dependency>
34  </dependencies>
35  ...
```

3.4. Acceso a datos

En el modulo *youtube-crawler-tool-data* se implemento la capa de acceso a datos de la aplicación. Al utilizar *Neo4j* como SGBD en la capa de datos, esta capa debía de realizar y gestionar la conexión a esta base de datos i proporcionar acceso a la capa de lógica de negocio.

Para su implementanción se utilizo el marco de desarrollo *Spring Data* y, mas concretamente, el proyecto *Spring Data Neo4j* [39]. Al utilizar este marco de desarrollo, ademas de poder utilizar las características que aporta *Spring Data* para dar soporte a JPA [40], como el uso de entidades para definir los objetos de dominio y uso de repositorios para la implementación de DAOs (Data Access Objects) [41], también podemos utilizar características propias que aporta el uso de *Neo4j*, tales como anotaciones para la definición de grafos o soporte para el lenguaje de consultas *Cypher*.

La relación de entidades (o objetos de dominio) definidos fue:

- **Crawler.java:** Representación en la aplicación de un proceso de recolección.
- **Video.java:** Representación en la aplicación de un vídeo de *YouTube*.
- **Channel.java:** Representación en la aplicación de un canal de *YouTube*.
- **Category.java:** Representación en la aplicación de una categoría.

A modo ilustrativo, se añade a continuación un extracto de la entidad *Video* en donde se observan algunas de las anotaciones propias del proyecto *Spring Data Neo4j*:

```
1  @NodeEntity
2  public class Video {
3
4    @Id
5    protected String id;
6    ...
7    @Relationship(type = "UPLOADED_BY", direction =
         Relationship.OUTGOING)
8    protected Channel channel;
9
10   @Relationship(type = "RELATED_TO", direction =
         Relationship.OUTGOING)
11   public Set<Video> related;
12   ...
```

En donde la anotación *@NodeEntity* identifica esta clase como un objeto de dominio o nodo en *Neo4j* y la anotación *@Relationship* permite definir el tipo de relación o arista entre dos nodos.

Y por lo que se refiere a los repositorios para dar acceso a estas entidades en el proyecto se definieron los siguientes:

- **CrawlerRepository.java:** Proporciona acceso a objetos de dominio de tipo *Crawler*.
- **VideoRepository.java:** Proporciona acceso a objetos de dominio de tipo *Video*.
- **ChannelRepository.java:** Proporciona acceso a objetos de dominio de tipo *Channel*.
- **CategoryRepository.java:** Proporciona acceso a objetos de dominio de tipo *Category*.

Y de igual forma que con las entidades, a continuación se proporciona un extracto del repositorio *VideoRepository* para ilustrar su implementación:

```
1  public interface VideoRepository extends Neo4jRepository<
       Video, String> {
2
3    @Depth(1)
4    @Query("MATCH (video:Video) WHERE video.favorite=true
         WITH video "
5      + "MATCH v=(video)-[:UPLOADED_BY|DISCOVERED_BY|
           CATEGORIZED_AS]->() RETURN video, nodes(v), rels(
           v)")
6    List<Video> findAllFavorite();
7    ...
8    @Query("MATCH (v:Video) -[:DISCOVERED_BY]-> (c:Crawler)
         WHERE ID(c)={0} DETACH DELETE v")
9    void removeByCrawlerId(Long crawlerId);
10   ...
```

En el cual, al extender de *Neo4jRepository* nos brinda muchas funcionalidades estandard que se suelen requerir a los objectos de acceso a datos, tales como operaciones de CRUD [42]. Pero ademas, se nos permite definir nuestras propias consultas las cuales pueden estar definidas en lenguaje *Cypher*.

Finalmente, en esta capa también se definen los objetos planos de *Java* conocidos como POJOs [43], los cuales en su gran mayoría son representaciones de las entidades definidas anteriormente y su uso esta destinado a actuar como mecanismo de comunicación (previamente serializados como JSON) entre la capa de presentación y la capa cliente.

3.5. Lógica de negocio

En el modulo *youtube-crawler-tool-core* esta implementada la capa de lógica de negocio. En ella, diferentes componentes que se ofrecen como servicios a ser utilizados por la capa de presentación, acceden a los recursos definidos en la capa de acceso a datos o en servicios de terceros para implementar la lógica de negocio de la aplicación.

3.5.1. Servicios

Así entonces, en el modulo *youtube-crawler-tool-core* se definen tantos servicios como componentes se definieron para la aplicación ademas de otros servicios que, se podría decir, actúan como complementos de los servicios principales. Entre los servicios complementarios se encuentran los denominados *converters* que son los encargados de transformar una entidad de base

de datos a su respectivo POJO y, en el caso de ser una entidad proveniente de *YouTube*, también realizan la transformación de entidad de *YouTube* a POJO. Ademas de los *converters*, entre los servicios complementarios también encontramos el servicio *YouTubeSearchService* que es el encargado de realizar las peticiones a la API de *YouTube*.

Aplicando esta estructuración, la aplicación respeta los principios de diseño SOLID [44] al conseguirse, entre otras características, la especialización de los servicios favoreciendo la extensibilidad y la reutilización de código.

A continuación se listan los servicios principales a los cuales se acceden desde la capa de presentación:

- **CrawlerService.java:** Actúa como interfaz del componente *CrawlerBusiness* y en cuya implementación se encuentra la lógica de negocio necesaria para la recolección de vídeos.
- **VideoService.java:** Actúa como interfaz del componente *VideoBusiness* y en cuya implementación se encuentra la lógica de negocio relacionada con los vídeos de la aplicación.
- **ChannelService.java:** Actúa como interfaz del componente *ChannelBusiness* y en cuya implementación se encuentra la lógica de negocio relacionada con los canales de la aplicación.
- **CategoryService.java:** Actúa como interfaz del componente *CategoryBusiness* y en cuya implementación se encuentra la lógica de negocio relacionada con las categorías de la aplicación.
- **AnalysisService.java:** Actúa como interfaz del componente *AnalysisBusiness* y en cuya implementación se encuentra la lógica de negocio relacionada con la representación en grafo de vídeos y canales de la aplicación.

Y por lo que se refiere a los servicios que actúan como *converters*, su listado es el siguiente:

- **CrawlerConverterServiceImpl.java:** Es el encargado de transformar entidades del tipo *Crawler* a su respectivo POJO (*CrawlerItem*) y viceversa.
- **VideoConverterServiceImpl.java:** Es el encargado de transformar entidades del tipo *Video* a su respectivo POJO (*VideoItem*) y viceversa. Ademas, también realiza la trasformaciones de la entidad de *Video* de *YouTuve* a POJO.

- **ChannelConverterServiceImpl.java:** Es el encargado de transformar entidades del tipo *Channel* a su respectivo POJO (*ChannelItem*) y viceversa. Ademas, también realiza la trasformaciones de la entidad de *Channel* de *YouTuve* a POJO.

- **CategoryConverterServiceImpl.java:** Es el encargado de transformar entidades del tipo *Category* a su respectivo POJO (*CategoryItem*) y viceversa.

Para conectar todos estos servicios entre si y habilitarlos en la capa de presentación, en el proyecto se hace un uso extensivo del patrón de inyección de independencias proporcionado por el marco de desarrollo *Spring Core*, el cual se encarga de instanciar objetos y inyectarlos como dependencias allí donde sean necesarias. De esta forma, se consigue un alto desacoplamiento entre los distintos componentes lo cual facilita el mantenimiento y la reusablilidad del código.

Para aplicar el patrón de inyección de independencias se ha utilizado el mecanismo de anotaciones habilitado por *Spring Core*. En el cual, para definir una nueva instancia a injectar se ha utilizado la anotación *@Service* y para inyectar una dependencia la anotación *@Autowired* tal y como se demuestra en el siguiente extracto de la implementación del servicio *VideoService*.

```
1  @Service("videoService")
2  public class VideoServiceImpl implements VideoService {
3
4    @Autowired
5    private VideoRepository videoRepository;
6    ...
7    @Autowired
8    @Qualifier("videoConverterService")
9    private YouTubeConverterService<com.google.api.services
         .youtube.model.Video, Video, VideoItem>
         videoConverterService;
10   ...
```

3.5.2. Crawler de YouTube

Una de las funcionalidades mas destacadas que se implementaron en la capa de lógica de negocio fue el proceso recolector (*crawler* en ingles) de vídeos de *YouTube*. En esta sección se detallan las características mas relevantes de dicha implementación.

Antes de nada, para poder desarrollar un proceso de recolección había que definir el modo en el cual la aplicación realizaría peticiones a la *YouTube Data API*. Para tales efectos, se definió en la aplicación el servicio ***YouTubeSearchService*** el cual tiene la responsabilidad de realizar dichas peticiones.

Para realizar las peticiones a la API de *YouTube* se decidió hacer uso de la librería *Java* cliente de la *YouTube Data API* definida por *Google* [45]. Gracias a esta librería podíamos extraernos de realizar y gestionar las peticiones HTTP a los servicios requeridos de la API de *YouTube* ya que las comunicaciones con este servicio extorno se podían realizar como si de otro servicio de la aplicación se tratase.

Para importar en el proyecto la librería cliente de la *YouTube Data API* fue necesario añadir la siguiente dependencia en el fichero *pom.xml* del modulo *youtube-crawler-tool-core* de la siguiente forma:

```
...
<!-- YouTube Data V3 support -->
<dependency>
  <groupId>com.google.apis</groupId>
  <artifactId>google-api-services-youtube</artifactId>
  <version>v3-rev193-1.23.0</version>
</dependency>
...
```

Y para inicializar el servicio de *YouTube* que realizaría nuestras peticiones a la API de *YouTube*, se añadió la siguiente instancia al contexto de *Spring* mediante una clase de configuración:

```
@Configuration
public class YouTubeConfiguration {
  protected static final String YOUTUBE_APPLICATION_NAME=
     "youtube-crawler-tool-app";

  @Bean
  public YouTube youTube() {
    // This object is used to make YouTube Data API
       requests.
    final YouTube youtube = new YouTube.Builder(new
       NetHttpTransport(), new JacksonFactory(),
      (request) -> {}).setApplicationName(
         YOUTUBE_APPLICATION_NAME).build();
    return youtube;
  }
}
```

Ahora, cada vez que se requiriera realizar una petición a la API de *YouTube*, se podía realizar utilizando el servició definido indicando en cada petición la *Google API Key* obtenida al configurar el entorno de desarrollo en el apartado 3.1 y que esta configurada en la aplicación mediante la propiedad *'ytct.youtube.api.key'*.

La relación completa de los métodos definidos en el servicio *YouTubeSearchService* es la siguiente:

- **searchVideos():** Realiza la petición al servicio *'Search: list'* de la API de *YouTube* para realizar una búsqueda textual y obtener un listado de identificadores de vídeos como respuesta.
- **searchRelatedVideos():** Realiza la petición al servicio *'Search: list'* de la API de *YouTube* para realizar una búsqueda por vídeo relacionado y obtener un listado de identificadores de vídeos como respuesta.
- **findVideos():** Realiza la petición al servicio *'Videos: list'* de la API de *YouTube* para recuperar el contenido de los vídeos dado un listado de identificadores de vídeos.
- **findChannel():** Realiza la petición al servicio *'Channels: list'* de la API de *YouTube* para recuperar el contenido de un canal dado un identificador de canal.

Habiendo resuelto el modo en el que se realizarían las peticiones a la API de *YouTube* mediante el servicio *YouTubeSearchService*, ahora era el momento de definir el proceso de recolección propiamente dicho. Para tales efectos se diseño el servicio ***CrawlerService*** el cual tiene la responsabilidad de realizar y gestionar los procesos de recolección.

A continuación se proporciona un pseudocódigo resumido a modo ilustrativo del proceso de recolección implementado en el servicio *CrawlerService*:

```
1  Funcion doCrawler(crawler)
2    crawler.status <- RUNNING;
3    Hacer
4      busqueda <- youTubeSearchService.searchVideos(crawler
           );
5
6      videos <- youTubeSearchService.findVideos(busqueda.
           videoIds);
7      Para i <- 0 Hasta i = videos.size() Hacer
8        Si videoRepository.findBy(video.id) = null Entonces
9          videoRepository.save(video);
10
11         Si channelRepository.findBy(video.channelId) =
               null Entonces
```

```
12              channel <- youTubeSearchService.findChannel(
                    video.channelId);
13              channelRepository.save(channel);
14            Fin Si
15          Fin Si
16
17          Si crawler.nivelRelacionado > 0) Entonces
18            nuevoCrawler <- new Crawler();
19            nuevoCrawler.videoRelacionado <- video.id;
20            nuevoCrawler.nivelRelacionado <- crawler.
                  nivelRelacionado -1;
21            nuevoCrawler.videosRelacionados <- crawler.
                  videosRelacionados;
22            doCrawler(nuevoCrawler);
23          Fin Si
24
25          i <- i+1;
26        Fin Para
27
28        Si busqueda.siguientePagina = null Entonces
29          crawler.status <- FINISHED;
30        Si no Entonces
31          crawler.pagina <- busqueda.siguientePagina;
32        Fin Si
33
34      Mientras crawler.status = RUNNING
35
36  Fin Funcion
```

Uno de los requerimientos del proyecto era poder realizar varios procesos de recolección de forma simultanea y controlar su ejecución. Por lo tanto, los procesos de recolección debían de ejecutarse en hilos de ejecución (o *Threads* en ingles) distintos al hilo principal de la aplicación para, de este modo, no bloquear el funcionamiento de la herramienta y permitir la ejecución asíncrona en paralelo de varios procesos de recolección.

Para habilitar la ejecución asíncrona de varios procesos de recolección, de nuevo se utilizaron las capacidades ofrecidas por el marco de trabajo *Spring* para la creación y gestión de los hilos de ejecución. Concretamente, para su configuración fue necesario instanciar un *TaskExecutor* mediante la clase de configuración *AsyncExecutorConfiguration* y anotar posteriormente con la anotación *@Async* el método deseado. De esta forma, cada vez que se ejecutara el método anotado con la anotación *@Async*, el *TaskExecutor* configurado crearía y gestionaría un nuevo hilo de ejecución para su procesado.

Así entonces, en nuestro caso el método que iniciaba la ejecución de un proceso de recolección tenia que ser anotado con la anotación *@Async* de la siguiente forma:

```
1   @Async
2   @Override
3   public CompletableFuture<CrawlerItem> executeCrawler(
        Long crawlerId) {
```

Finalmente, el uso de varios hilos de ejecución puede producir errores de concurrencia si mas de un hilo intenta acceder al mismo recurso al mismo tiempo. Para evitar este tipo de problemas, en el código se protegieron los recursos que eran susceptibles de ser accedidos por varios hilos de ejecución al mismo tiempo con la sentencia *synchronized*. De esta forma, cuando un hilo de ejecución entra dentro de un bloque con la sentencia *synchronized*, evita que cualquier otro hilo acceda también dentro de dicho bloque o a los recursos del mismo.

Por ejemplo, durante la ejecución de un proceso de recolección el usuario puede cambiar su estado para detenerlo. Para asegurar que ningún proceso esta cambiando el estado del mismo proceso de recolección al mismo tiempo y evitar problemas de consistencia, se utiliza la sentencia *synchronized* de la siguiente forma:

```
1   synchronized (this) {
2     final Crawler crawler = this.crawlerRepository.
          findById(id).orElse(null);
3     ...
4     crawler.setStatusByEnum(newStatus);
5     this.crawlerRepository.save(crawler);
6     ...
7   }
```

3.6. Presentación

En el modulo *youtube-crawler-tool-api* es donde se implemento el controlador de solicitudes de recursos para habilitar la API REST de la aplicación y en el modulo *youtube-crawler-tool-war* el controlador de solicitudes de navegador y la capa cliente de la aplicación web. A continuación en los siguientes apartados se detallan las características mas relevantes de su implementación.

3.6.1. Solicitudes de recursos

Para habilitar la API REST de la aplicación definida en el apartado 2.5.4, en el modulo *youtube-crawler-tool-api* se hace uso del marco de desarrollo *Spring MVC* [46]. Gracias al cual, fue posible definir la API REST, los recursos disponibles y sus métodos mediante el uso de anotaciones en cada una de las clases de tipo *controller* que fueron definidas para cada componente.

Es en estas clases tipo *controller* es donde se realizan peticiones a los servicios habilitados en la capa de lógica de negocio para responder a las peticiones de recursos realizadas por los usuarios los cuales son devueltos en formato JSON. El listado completo de *controllers* definidos es el siguiente:

- **CrawlerController.java:** Gestiona peticiones de recursos del tipo procesos de recolección. Responde a peticiones realizadas a la ruta *'/api/crawlers'*.
- **VideoController.java:** Gestiona peticiones de recursos del tipo vídeos. Responde a peticiones realizadas a la ruta *'/api/videos'*.
- **ChannelController.java:** Gestiona peticiones de recursos del tipo canales. Responde a peticiones realizadas a la ruta *'/api/channels'*.
- **CategoryController.java:** Gestiona peticiones de recursos del tipo categorías. Responde a peticiones realizadas a la ruta *'/api/categories'*.
- **AnalysisController.java:** Gestiona peticiones de recursos del tipo grafos. Responde a peticiones realizadas a la ruta *'/api/graphs'*.

A modo de ejemplo, se incluye a continuación un extracto del controlador *VideoController.java*:

```
1  @Controller
2  @RequestMapping("/api/videos")
3  public class VideoController {
4
5    @Autowired
6    private VideoService videoService;
7
8    @ResponseBody
9    @RequestMapping(value = "/favorites", method =
       RequestMethod.GET)
10   public List<VideoItem> listFavoritesVideos() {
11     return this.videoService.listFavoritesVideos();
12   }
13   ...
```

La API REST habilitada mediante estos controladores es consumida por la interfaz web implementada en la capa cliente. En ella, las diferentes vistas de la aplicación realizan peticiones HTTP para acceder a los recursos necesarios. Dichas peticiones se realizan de forma asíncrona utilizando la librería de *JavaScript JQuery* y, mas concretamente, haciendo uso de la función *ajax()* [47].

Para evitar la repetición de código en la capa cliente, se definió una librería *JavaScript* llamada *youtube-crawler-tool.js* en la cual se implementan de forma genérica las llamadas *ajax*. Por ejemplo, el código *JavaScript* para realizar una petición *ajax* de tipo GET se define de la siguiente forma:

```
var doGet = function (requestUrl, callback) {
  console.log("doGet for url: " + requestUrl);
  $.ajax({
    url: requestUrl,
    method: 'GET',
    success: callback
  });
}
```

En donde, de querer obtener el listado de vídeos favoritos realizaríamos una petición a la API REST de tipo GET de la siguiente forma:

```
doGet("/api/videos/favorites", function (favorites) {
  ...
});
```

3.6.2. Solicitudes de navegador

En el modulo *youtube-crawler-tool-war* es donde se encuentra implementada la capa cliente de la aplicación y el controlador de solicitudes de navegador.

Dicho controlador es implementado en la clase *YouTubeCrawlerToolController.java* en donde, gracias al uso de *Spring MVC*, se definen las peticiones a las diferentes vistas de la aplicación. A diferencia de los *controllers* definidos para las solicitudes de recursos, en el controlador de solicitudes de navegador como respuestas se devuelven redirecciones a las vistas definidas para cada caso.

A modo de ejemplo, se incluye a continuación un extracto de la clase *YouTubeCrawlerToolController.java* en el cual se gestiona la petición a la vista en detalle de un vídeo en concreto:

```
@Controller
@RequestMapping("/")
public class YouTubeCrawlerToolController {
  ...
  private static final String VIDEOS_VIEW = "videosView.
     html";
  ...
  @RequestMapping(value="videos/{id}")
  public String getVideoView(Model model, @PathVariable(
     value = "id", required = true) String id) {
    model.addAttribute("activeMenuOption", VIDEOS_VIEW);
    model.addAttribute("videoId", id);
    return VIDEO_VIEW;
  }
  ...
```

Para la construcción de las vistas html de la aplicación, se utilizo el motor de plantillas para Java *Thymeleaf* [48], el cual, es el recomendado actualmente por la plataforma *Spring*. Con *Thymeleaf* es posible definir plantillas HTML permitiendo inyectar código servidor de una forma muy natural ademas de ofrecer integración con *Spring*. Utilizando esta tecnología fue posible implementar las vistas definidas en el apartado 2.6.

Para no partir de cero en el desarrollo de dichas vistas, como punto de partida se decidió buscar una plantilla html que se adaptara fácilmente a las funcionalidades diseñadas para la aplicación y que tuviera una licencia libre de uso. La plantilla escogida para tales efectos fue *SB Admin 2* [49].

3.6.3. Seguridad de vistas y recursos

Uno de los requerimientos de la aplicación era la definición de dos tipos de usuarios distintos:

- **Usuario anónimo:** Solo se le permite realizar acciones de lectura en la aplicación y generar grafos.
- **Usuario identificado:** Se le permite realizar cualquier acción en la aplicación.

Este requerimiento conllevaba proveer a la aplicación de la funcionalidad necesaria para la identificación de usuarios mediante inicio de sesión y añadir una capa de seguridad para proteger los recursos y vistas que no fueran de dominio publico. Para conseguirlo, la aplicación hizo uso del marco de trabajo *Spring Security* [50].

Al añadir *Spring Security* al proyecto, por defecto se provee a la aplicación de la funcionalidad necesaria para identificar y crear sesiones de usuario en la aplicación realizando peticiones de tipo POST */login* tal y como se realiza en la vista *loginView.html*. Para probar dicha funcionalidad, en la clase de configuración *YouTubeCrawlerToolSecurityConfiguration.java*, al levantarse la aplicación se añade al contexto de *Spring* un usuario de prueba cuyo nombre y contraseña se definen en las propiedades *ytct.security.default.username* y *ytct.security.default.password*.

Es también en la clase de configuración *YouTubeCrawlerToolSecurityConfiguration.java* en donde se protegen los recursos ofrecidos por la API REST de la aplicación. En este caso, para cumplir con los requerimientos de la aplicación, teníamos que proteger todas las peticiones a recursos que no fueran de tipo GET para asegurarnos que el usuario que las realiza es un usuario identificado. Dicha configuración se facilita a continuación:

```
1   @Override
2   protected void configure(HttpSecurity http) throws
        Exception {
3     http
4       .csrf().disable()
5         .authorizeRequests()
6           // Crawler component security
7           .antMatchers(HttpMethod.POST, "/api/crawlers").
              authenticated()
8           .antMatchers(HttpMethod.PUT,  "/api/crawlers").
              authenticated()
9           .antMatchers(HttpMethod.DELETE,       "/api/
              crawlers").authenticated()
10          // Video component security
11          .antMatchers(HttpMethod.PUT,  "/api/videos").
              authenticated()
12          .antMatchers(HttpMethod.DELETE,       "/api/
              videos").authenticated()
13          // Channel component security
14          .antMatchers(HttpMethod.DELETE,       "/api/
              channels").authenticated()
15          // Category component security
16          .antMatchers(HttpMethod.POST, "/api/categories"
              ).authenticated()
17          .antMatchers(HttpMethod.PUT,  "/api/categories"
              ).authenticated()
18          .antMatchers(HttpMethod.DELETE,       "/api/
              categories").authenticated()
19          // Allow any other request
20          .anyRequest().permitAll()
21          .and()
```

```
22              .formLogin()
23                .loginPage("/login")
24                .permitAll()
25                .and()
26              .logout()
27                .logoutSuccessUrl("/")
28                .permitAll();
29    }
```

Finalmente, para proteger las vistas a las cuales es necesario estar identificado para poder acceder (por ejemplo, en el caso de la vista de vídeos favoritos), gracias a la integración de *Thymeleaf* con *Spring* es posible definir enlaces en las vistas que solo serán renderizados en el caso de cumplirse unos requerimientos de seguridad en concreto. Por ejemplo, en el caso del enlace para acceder a la vista de vídeos favoritos, mediante la etiqueta *sec:authorize* es posible definir que el código html en donde esta definido esta etiqueta solo se renderice en caso que el usuario este identificado:

```
1  <li class="dropdown" sec:authorize="isAuthenticated()">
2    <a class="dropdown-toggle" th:href="@{/videos/favorites
       }" id="favoriteVideosView">
3      Favorites <i class="fa fa-star fa-fw"></i>
4    </a>
5  </li>
```

3.6.4. Visualización de vídeos en grafo

En la capa cliente de la aplicación, en las vistas html se utilizan varias librerías *JavaScript* que aportan funcionalidades de presentación mejoradas a la interfaz gráfica. La mas destacada de ellas es la librería *Cytoscape.js* [51] que es utilizada para la visualización del grafos en la vista de análisis.

Dicha librería consume una fuente de datos en formato JSON en donde se especifican los nodos y las aristas para la construcción del grafo. Gracias a su facilidad de uso y a las posibilidades de personalización que hicieron posible cumplir los requisitos de visualización de la clienta definidos en el apartado 2.2.3, se eliguio su uso en la aplicación por encima de otras alternativas como *D3.js* [52] o *Sigma.js* [53].

Para obtener los nodos y las aristas en el formato JSON requerido por la librería, la interfaz de usuario realiza una petición asíncrona al recurso *POST api/graphs* el cual devuelve los vídeos y canales que hayan sido recolectados previamente por el sistema y que sean resultados de unos criterios de búsqueda introducidos por el usuario.

A continuación se facilita un ejemplo de la respuesta JSON obtenida por el recurso *POST api/graphs* y que cumple con el formato requerido por la librería *Cytoscape.js*:

```
{
  "elements": {
    "nodes": [
      {
        "data": {
          "id": "v-haqi4xvjvKo",
          "resourceId": "haqi4xvjvKo",
          "typeCode": "v",
          "shape": "ellipse",
          "color": "#dd1313",
          "size": 25,
          "video": {
            //campos propios del objecto video
          }
        }
      },
      ...
    ],
    "edges": [
      {
        "data": {
          "source": "v-L0jQz6jqQS0",
          "target": "v-5d667Bb_iYA",
          "outgoing": "L0jQz6jqQS0",
          "incoming": "5d667Bb_iYA",
          "outgoingType": "v",
          "incomingType": "v"
        }
      },
      ...
    ]
  }
}
```

Una vez obtenida la respuesta, en la aplicación se genera un nuevo grafo inicializando *Cytoscape.js* con los elementos obtenidos de la siguiente forma:

```
function initCytoscape(elements) {
  cy = cytoscape({
    container: document.getElementById('cy'),
    style: cytoscape.stylesheet()
      .selector('node')
        .css({
          'shape': 'data(shape)',
```

```
            'background-color': 'data(color)',
            'width': 'mapData(size, 0, 100, 10, 60)',
            'height': 'mapData(size, 0, 100, 10, 60)'
          })
        .selector('edge')
          .css({
            'curve-style': 'haystack' // fast edges!
          }),
        layout: {
          name: 'cose',
          animate: false,
          idealEdgeLength: 100,
          nodeOverlap: 20,
          refresh: 20,
          fit: true,
          padding: 30,
          randomize: false,
          componentSpacing: 100,
          nodeRepulsion: 400000,
          edgeElasticity: 100,
          nestingFactor: 5,
          gravity: 80,
          numIter: 20,
          initialTemp: 200,
          coolingFactor: 0.95,
          minTemp: 1.0
        },
        elements: elements
    });

    cy.nodes().forEach(function(node){
      nodeJson = node.json();
      cy.getElementById(node.id()).qtip({
        content: generateNodeContent(nodeJson.data),
        position: {
          my: 'top center',
          at: 'bottom center'
        },
        style: {
          classes: 'qtip-bootstrap',
          tip: {
            width: 16,
            height: 8
          }
        }
      });
    });
    ...
  };
```

3.7. Definición y ejecución de pruebas

Durante el desarrollo de la aplicación y a medida que se iba finalizando la implementación de los distintos componentes, se fueron definiendo y ejecutando pruebas de integración para probar la funcionalidad desarrollada.

Las pruebas a realizar se definieron según los casos de uso documentados en la sección 2.2.4 asignando un código de identificación para cada uno de ellos. Ademas, al agrupar las pruebas a realizar por componentes nos permitió evaluar la calidad de cada componente por separado tan pronto estaba disponible.

En el apartado anexo 8.3 se facilita el resultado completo de las pruebas de integración realizadas.

3.8. Pruebas de concepto

Tal y como se ha comentado en anteriores apartados, en el desarrollo de la aplicación se realizaron varias pruebas de concepto para poner a prueba distintas soluciones y estudiar la viabilidad se ser incorporadas en el sistema.

En esta apartado se resume brevemente cada una de ellas junto con su finalidad y los resultados obtenidos:

POCTwitterCrawler
Descripción: Prueba de concepto desarrollada para probar la integración con la API de *Twitter* y guardar los contenidos recolectados en *MongoDB*.
Resultados: Aunque la prueba resulto satisfactoria y fue posible obtener mensajes de *Twitter* y guardarlos en *MongoDB*, debido a la posterior elección de *YouTube* como red social a utilizar y a *Neo4j* como SGBD, no se pudo aprovechar el conocimiento adquirido para la implementación final.
Enlace: https://github.com/jsanchezmend/TFGAntivacunas/tree/master/POCTwitterCrawler.

POCYouTubeCrawler
Descripción: Prueba de concepto desarrollada para probar la integración con la API de *YouTube* y guardar los contenidos recolectados en *MongoDB*. Posteriormente también se añadio la visualización en grafo utilizando librería JavaScript '*Cytoscape.js*'.
Resultados: La prueba resulto satisfactoria y fue posible obtener mensajes de *YouTube* y guardarlos en *MongoDB* ademas de su visualización en formato de grafo. De esta prueba de concepto en la implementación de la aplicación se aprovecho el conocimiento adquirido a la hora de utilizar la API de *YouTube* y de mostrar los videos en formato de grafo.

Enlace: https://github.com/jsanchezmend/TFGAntivacunas/tree/master/POCYouTubeCrawler.

POCYouTubeCrawlerNeo4j
Descripción: Prueba de concepto desarrollada para probar la persistencia de contenidos recolectados en *Neo4j*.
Resultados: La prueba resulto satisfactoria y fue posible guardar los contenidos recolectados en *Neo4j*. El conocimiento adquirido a la hora de implementar el acceso a los datos utilizando *Neo4j* como capa de datos fue utilizado en la implementación final de la aplicación.
Enlace: https://github.com/jsanchezmend/TFGAntivacunas/tree/master/POCYouTubeCrawlerNeo4j.

4. Implementación y puesta en funcionamiento

4.1. Servidor de explotación

Para dar acceso a la clienta y *stakeholders*, se habilito un servidor de explotación desde donde el cual poder acceder a la aplicación.

Fue con la aplicación instalada en este servidor donde la clienta pudo realizar el estudio sobre el movimiento antivacunas en la red social *YouTube* recolectando vídeos y analizando sus relaciones. La aplicación instalada en este servidor fue actualizándose a medida que se avanzaba en el desarrollo.

Para realizar la instalación de la aplicación se realizo el procedimiento detallado en el manual de instalación descrito en el apartado 8.1 del anexo a este documento. También en el anexo en la sección 8.2, se encuentran ilustraciones de la puesta en funcionamiento de la aplicación con imágenes de la ultima versión instalada.

Inicialmente, para tales efectos se habilito una aplicación Web en el servicio *Microsoft Azure* con el dominio: http://youtubecrawlertoolwebapp.azurewebsites.net. Pero debido a los recursos necesarios para la instalación de Neo4j y al escaso presupuesto que *Microsoft Azure* ofrece a las cuentas de estudiante, rápidamente este servició quedo bloqueado por falta de presupuesto disponible.

4.2. Formación y sensibilización

En toda aplicación, ya sea de escritorio, móvil o web, para el usuario existe una primera fase de aprendizaje o de sensibilización que es mas o menos prolongada en medida de lo usable y bien diseñada que este una aplicación.

YouTubeCrawlerTool no es una excepción y, aunque la aplicación goza de una alta usabilidad bajo el criterio de la propia clienta, se han llevado a cabo sesiones de formación para acelerar este proceso.

Dichas sesiones se realizaron de forma online en las cuales asistieron la clienta y los *stakeholders* de la aplicación, llegando a enregistrarse una de las sesiones en *YouTube*: https://youtu.be/ETWtbywxOmI.

Gracias a estas sesiones, ademas de sensibilizar a los futuros usuarios de la aplicación, sirvieron para recoger sugerencias y mejoras con las cuales se

llevaron a cabo posteriores refinamientos de la aplicación, como por ejemplo la adicción de la funcionalidad de favoritos o una visualización diferente para los listados de vídeos.

4.3. Funcionalidades no implementadas

De las funcionalidades diseñadas y recogidas como requisitos de la aplicación en forma de casos de uso en la sección 2.2.4, las únicas que no han sido finalmente implementadas en la aplicación son las relativas a la habilitación de información estadística sobre el estado de la aplicación y la categorización de vídeos realizada. Concretamente los casos de uso no implementados han sido:

- **UC_2:** View all crawlers statistics.
- **UC_8:** View crawler process statistics.
- **UC_16:** View all channels statistics.
- **UC_19:** View channel statistics.

Las razones por las que estos casos de uso no se llegaron a implementar son debido a falta de tiempo para su desarrollo y la priorización de otras funcionalidades en detrimento de estas. En la sección 5.3 se proporcionan más detalles sobre estas decisiones.

5. Conclusiones

5.1. Conclusiones del trabajo

En la realización de este trabajo ademas de darme la oportunidad de poner a prueba los conocimientos adquiridos durante el grado en ingeniería informática, me ha permitido experimentar como es el proceso de desarrollo de software en contacto directo con los usuarios finales y, en este caso, con la clienta de la solución.

Gracias a esta circunstancia, he podido obtener lecciones muy valiosas que creo que me serán de utilidad en el futuro. Ya que, al tener contacto con la usuaria final de la aplicación durante todo el desarrollo de la misma, he podido aprender de su perspectiva y no centrarme únicamente en el desarrollo de la solución. Una de las conclusiones mas valiosas que he obtenido, es que en un proyecto de software siempre se pueden producir cambios (sobretodo en etapas finales), y debido a ello, hay que estar siempre atento y prevenirlos para minimizar sus riesgos siempre que sea posible.

Como resultado de esta colaboración y la metodología utilizada, la aplicación obtenida ademas de cumplir los objetivos marcados al inicio del proyecto, a conseguido cumplir con las expectativas de la clienta permitiendo iniciar el estudio del movimiento antivacuna en la red social *YouTube*. En palabras de la propia Johanna Milena Rodríguez Vera clienta de la aplicación:

La herramienta tiene una interfaz visual que le facilita al usuario realizar las búsquedas y los filtros que son necesarios para encontrar la información que necesita.

Por las características que tiene al API de YouTube los resultados que se obtienen se encuentran dependiendo de la localización de quien realiza la búsqueda, siendo diferentes los vídeos que se proponen aunque tengan los mismos términos de búsqueda y filtrado. Este punto se puede convertir en un aspecto interesante a evaluar en el futuro por parte del investigador.

Debido a como YouTube relaciona sus contenidos, los vídeos que se muestran como resultado en muchas ocasiones no tienen relación alguna con el término utilizado. Con lo cual hace necesario un trabajo manual de filtrado de la información por parte del investigador para categorizar correctamente los vídeos que se obtienen como resultado.

La funcionalidad de grafos con la selección de múltiples variables, permite en éste escenario la obtención de grafos con diferente información que puede permitir de manera posterior su análisis de acuerdo a las características del estudio.

Ademas de lo expuesto hasta el momento, personalmente la realización de este proyecto me ha permitido desarrollar una aplicación de principio a fin, pasando des de la etapa de definición y diseño de la solución al desarrollo de la capa de datos, de aplicación y de la interfaz de usuario, obteniendo una visión global sobre el significado de desarrollar software. Por otro lado, he aprendido nuevas tecnologías tales como el uso de bases de datos NoSQL orientadas a grafos, que considero muy interesantes y con las cuales me gustaría trabajar en el futuro.

Así entonces, por los puntos destacados anteriormente y por el grado de satisfacción conseguido por la usuaria, considero que la realización del proyecto ha sido un éxito.

5.2. Grado de cumplimiento de los objetivos

Aunque algunas de las funcionalidades diseñadas inicialmente para el proyecto no se han llegado a implementar (tales como la incorporación de datos estadísticos sobre la categorización de los contenidos), se han podido lograr todos los objetivos planteados inicialmente en el apartado 1.2.

A continuación se facilita la relación de objetivos definidos y su grado de cumplimiento al termino del desarrollo del proyecto:

- **Investigar que funcionalidades aportan las API públicas ofrecidas por *YouTube* y analizar como se pueden utilizar para la obtención de la información requerida**: Se ha analizado la API de *YouTube* en la sección 3.2.1 y se ha estudiado conjuntamente con la clienta que información va a ser recolectada por la aplicación. Este objetivo se realizo durante la primera fase de desarrollo.
- **Determinar como almacenar y acceder de forma eficiente a la gran cantidad de información que se obtendrá**: Se ha tomado la decisión de utilizar una base de datos NoSQL enfocada a grafos (*Neo4j*) para poder acceder a la información de forma eficiente. Se ha diseñado un diagrama UML para representar el esquema invariante de la información y se ha adaptado al concepto de nodos y relaciones de las base de datos de grafos. Se realizo una prueba de concepto para estudiar la viabilidad de utilizar *Neo4j* como base de datos en el proyecto con resultado favorable. La prueba de concepto realizada fue utilizada como base para la implementación de la aplicación durante la segunda fase de desarrollo. Este objetivo se considera realizado.

- **Permitir la recolección de información según criterios de búsqueda proporcionados por el usuario final**: Conjuntamente con el cliente se han determinado los criterios de búsqueda a utilizar. Se ha desarrollado una prueba de concepto en la cual ha sido posible recolectar información de vídeos y canales utilizando la API de *YouTube*. La prueba de concepto realizada fue utilizada como base para la implementación de la aplicación durante la segunda fase de desarrollo. Este objetivo se considera realizado.

- **Habilitar la gestión, visualización y exportación de datos obtenidos en distintos procesos de extracción para su posterior análisis en herramientas especializadas**: Se ha diseñado una interfaz de usuario para proporcionar estas funcionalidades y se ha diseñado la arquitectura de la aplicación para hacerlas viables. En la segunda fase de desarrollo se ha adoptado y implementado la arquitectura propuesta en la primera fase de desarrollo y la interfaz de usuario necesarias para habilitar estas funcionalidades al usuario final. Este objetivo se considera realizado.

- **Ofrecer herramientas de visualización para el análisis y comprensión de los datos obtenidos**: Se ha decidido el uso de un grafo para la visualización de los datos obtenidos y se ha determinado cuales serán sus componentes. Se ha realizado una prueba de concepto en la cual ha sido posible visualizar los datos obtenidos a través de la API de *YouTube* en un grafo. La prueba de concepto realizada fue utilizada como base para la implementación final de la aplicación y se consiguió visualizar los datos obtenidos en un grafo acorde a las especificaciones del cliente. Este objetivo se considera realizado.

- **Proporcionar una interfaz de usuario usable que permita realizar las acciones requeridas por el usuario final**: Se ha realizado una propuesta de interfaz de usuario que ha sido aprobada por la cliente. En la segunda fase de desarrollo se implemento la interfaz gráfica propuesta con la aprobación final de la clienta. Este objetivo se considera realizado.

5.3. Seguimiento de la planificación y metodología

Debido a la naturaleza del proyecto desarrollado en donde se ha trabajado conjuntamente con la clienta y usuaria final de la aplicación, creo que la elección de una metodología de diseño centrado en el usuario ha sido la correcta al priorizar, por encima de todo, la satisfacción de los usuarios.

Por otro lado, la aplicación de una metodología de desarrollo ágil como *Scrum* nos ha permitido poder aplicar dicha metodología de forma efectiva produciendo componentes de software funcionales en cortos plazos de tiempo desde que fueran requeridos por la cliente y su inclusión en la aplicación, especialmente en las ultimas fases de desarrollo.

Durante la realización del proyecto se recibieron peticiones por parte de la clienta y de los *stakeholder* que afectaron al alcance de la aplicación o a su implementación final. La relación de los cambios más destacados efectuados con relación a la propuesta inicial fue la siguiente:

- **Cambio de red social:** Debido a las restricciones de uso de la API gratuita de Twitter tal y como se describe en el apartado 2.2.2, se decidió utilizar en substitución la red social *YouTube*.
- **Cambio de SGBD:** Inicialmente se había planteado el uso de la base de datos documental NoSQL MongoDB como el SGBD de la aplicación, pero debido a los motibos expuestos en el apartado 3.2.2, finalmente se opto por el uso de una bases de datos oriantada a grafos como *Neo4j*.
- **No inclusión de información estadística sobre la categorización:** En la propuesta de interfaz de usuario presentada, en algunas secciones de la aplicación se incluía información estadística sobre el estado de la aplicación y de la información recolectada (porcentaje de vídeos categorizados, distribución de categorías, etc.). Debido al ajustado tiempo disponible para la implementación de esta funcionalidad y a la baja prioridad dada por la clienta en comparación a otras funcionalidades, finalmente esta característica no fue incluida dentro de la aplicación.
- **Funcionalidad de favoritos:** En las ultimas fases de desarrollo y después de realizar la presentación de la aplicación a la clienta, debido a la gran cantidad de vídeos que pueden ser recolectados por la aplicación y para poder facilitar su identificación, se considero oportuno añadir la funcionalidad de favoritos para poder marcar los vídeos que la clienta considerara oportunos como favoritos para poder acceder a ellos de forma mas eficiente.

- **Nuevas funcionalidades en el listado de vídeos:** También en las ultimas fase de desarrollo, para hacer mas eficiente la tarea de categorización, en los diferentes listados de vídeos en la aplicación se añadió la funcionalidad para la categorización de vídeos. De esta forma, ya no es requerido acceder a un vídeo para cambiar su categoría.

Debido principalmente a las actividades no previstas detalladas, de la planificación inicial para la primera fase de desarrollo no se pudieron ejecutar las actividades relacionadas con la implementación del recolector de vídeos. Aunque en su momento represento una desviación importante en el proyecto de un total de 19 días, gracias al detallado diseño de la aplicación, el exhaustivo análisis de la API de *YouTube* y, sobretodo, a las diferentes pruebas de concepto realizadas, fue posible absorber estas tareas dentro de la segunda fase de desarrollo.

Debido a los motivos expuestos, la planificación inicial del proyecto facilitada en la sección 1.4 se vio afectada, debiendo ser actualizada por el siguiente diagrama de Gantt:

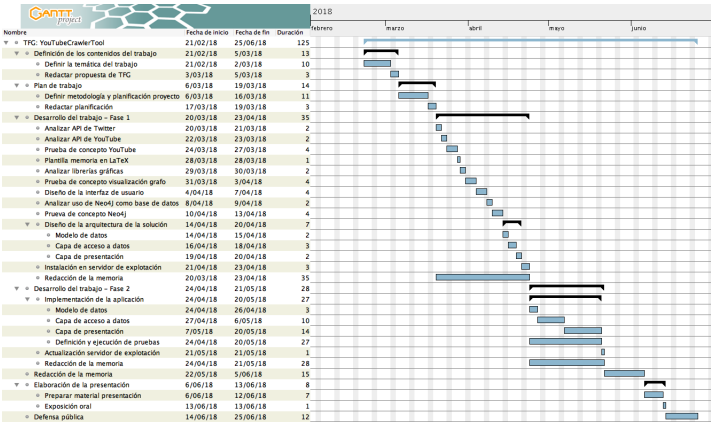


Figura 29: Listado final de tareas y diagrama de Gantt

Nombre	Fecha de inicio	Fecha de fin	Duración
TFG: YouTubeCrawlerTool	21/02/18	25/06/18	125
Definición de los contenidos del trabajo	21/02/18	5/03/18	13
Definir la temática del trabajo	21/02/18	2/03/18	10
Redactar propuesta de TFG	3/03/18	5/03/18	3
Plan de trabajo	6/03/18	19/03/18	14
Definir metodología y planificación proyecto	6/03/18	16/03/18	11
Redactar planificación	17/03/18	19/03/18	3
Desarrollo del trabajo – Fase 1	20/03/18	23/04/18	35
Analizar API de Twitter	20/03/18	21/03/18	2
Analizar API de YouTube	22/03/18	23/03/18	2
Prueba de concepto YouTube	24/03/18	27/03/18	4
Plantilla memoria en LaTeX	28/03/18	28/03/18	1
Analizar librerías gráficas	29/03/18	30/03/18	2
Prueba de concepto visualización grafo	31/03/18	3/04/18	4
Diseño de la interfaz de usuario	4/04/18	7/04/18	4
Analizar uso de Neo4J como base de datos	8/04/18	9/04/18	2
Prueva de concepto Neo4j	10/04/18	13/04/18	4
Diseño de la arquitectura de la solución	14/04/18	20/04/18	7
Modelo de datos	14/04/18	15/04/18	2
Capa de acceso a datos	16/04/18	18/04/18	3
Capa de presentación	19/04/18	20/04/18	2
Instalación en servidor de explotación	21/04/18	23/04/18	3
Redacción de la memoria	20/03/18	23/04/18	35
Desarrollo del trabajo – Fase 2	24/04/18	21/05/18	28
Implementación de la aplicación	24/04/18	20/05/18	27
Modelo de datos	24/04/18	26/04/18	3
Capa de acceso a datos	27/04/18	6/05/18	10
Capa de presentación	7/05/18	20/05/18	14
Definición y ejecución de pruebas	24/04/18	20/05/18	27
Actualización servidor de explotación	21/05/18	21/05/18	1
Redacción de la memoria	24/04/18	21/05/18	28
Redacción de la memoria	22/05/18	5/06/18	15
Elaboración de la presentación	6/06/18	13/06/18	8
Preparar material presentación	6/06/18	12/06/18	7
Exposición oral	13/06/18	13/06/18	1
Defensa pública	14/06/18	25/06/18	12

Figura 30: Listado final de tareas

5.4. Propuesta de mejoras

Como en toda aplicación, siempre existen propuestas que pueden mejorar la calidad y la satisfacción de los usuarios sobre el producto desarrollado, las cuales pueden ser funcionales o no funcionales. Las razones por las que estas propuestas no han sido incorporadas en el producto final entregado pueden ser varias, ya sea por ejemplo por falta de tiempo asignado para su desarrollo o por quedar fuera del alcance real del proyecto.

A continuación se facilita un listado de propuestas de mejora de la aplicación que fueron detectadas durante el desarrollo de la misma:

Propuestas funcionales:

- **Accesibilidad y usabilidad**: El propósito principal de la capa cliente desarrollada fue la de habilitar las funcionalidades desarrolladas a la clienta de la aplicación, por lo que la interfaz de usuario diseñada y implementada se centra en este objetivo. Para asegurar que la aplicación puede usarse de forma usable por un publico más amplio, debería someterse a un estudio de experiencia de usuario que tomara también en consideración la accesibilidad de la aplicación.

- **Añadir inteligencia artificial para la categorización de vídeos**: Ahora mismo, para mostrar grupos de contenidos bien diferenciados en el grafo se requiere realizar previamente un trabajo manual de categorización, el cual, dependiendo del volumen de datos sobre el que se requiera trabajar puede llegar a ser inviable. Para solucionar esta problemática se propone la adicción al proyecto de capacidades de inteligencia artificial que hagan posible la categorización automática de los contenidos recolectados. Como propuesta inicial a estudio, se podría considerar el uso de un modelo basado en reglas que, al aplicarlas sobre los metadatos de los vídeos (tales como titulo o descripción), se pudiera asignar automáticamente categorías a los vídeos.

Propuestas no funcionales:

- **Manejo de errores en la API REST:** Debido a que la API REST desarrollada inicialmente va a ser utilizada solo por un cliente web, esta no provee un manejo de errores adecuado delegando esta parte de la lógica en la interfaz web. Para que esta API pueda ser consumida por terceros o distintos tipos de clientes, esta debería proporcionar códigos y mensajes de error adecuados a cada situación.

- **Paginación de resultados en la API REST:** Parecido al punto anterior, actualmente la paginación de resultados se realiza en la interfaz de usuario. La API desarrollada debería incorporar la funcionalidad necesaria para permitir la paginación de listados de resultados para minimizar las llamadas a la API y el ancho de banda utilizado por la aplicación.

- **Mejorar los registros de la aplicación:** Aunque en la ejecución de la aplicación se genera un fichero de registro, la aplicación debería proporcionar mejor y un mayor nivel de registro por tal de facilitar su monitorización.

- **Realizar filtros en la búsqueda de *'Analysis'* por base de datos en vez de lógicamente:** En la consulta realizada a base de datos para la obtención de vídeos a ser presentados en el grafo, solo se realiza el filtrando de los campos obligatorios (por fecha), los demás filtros disponibles en esta funcionalidad (tales como por categoría o proceso de recolección) no son obligatorios y pueden venir informados o no. Por falta de tiempo en la investigación, no se encontró la forma de realizar consultas a *Neo4J* con parámetros que podían venir informados o no, como solución temporal se decidió realizar el filtrado de los parámetros opcionales de forma lógica posteriormente a la realización de la consulta a la base de datos. Esta solución temporal puede conllevar problemas de rendimiento en el futuro y, aunque como se comenta a

continuación no es el principal problema potencial de rendimiento en la visualización del grafo, debería encontrarse una alternativa mejor.

- **Mejorar el rendimiento de la representación del grafo:** Tal y como se comenta en la página web oficial de la librería *JavaScript Cytoscape.js* utilizada para la visualización del grafo, en los gráficos con una gran cantidad de nodos a representar el rendimiento de la librería empieza a degradarse [28]. En las pruebas realizadas se ha confirmado esta afirmación al, por ejemplo, tardar aproximadamente unos 34 segundos en visualizarse un grafo con mil nodos (4 segundos para obtener los vídeos a representar de la base de datos y a realizar el filtrado lógico de los mismos más 30 segundos para que la librería los visualice por pantalla). Debido al gran volumen de información con la que se puede trabajar con la aplicación, para mejorar el rendimiento en la visualización de grafos se deberían adoptar medidas correctoras (tales como disminuir la información a mostrar por cada nodo) o estudiar alternativas al uso de *Cytoscape.js* como librería *JavaScript* para la representación del grafo.

6. Glosario

ACID	*Atomicity, Consistency, Isolation and Durability*
AJAX	*Asynchronous JavaScript And XML*
API	*Application Programming Interface*
CSV	*Comma-Separated Values*
DAO	*Data Access Object*
DB	*DataBase*
DOM	*Document Object Model*
HTTP	*HyperText Transfer Protocol*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
MVC	*Model–View–Controller*
NoSQL	*Not Only SQL*
POC	*Proof Of Concept*
POJO	*Plain Old Java Object*
REST	*REpresentational State Transfer*
RPC	*Remote Procedure Call*
SGBD	*Sistema Gestor de Base de Datos*
SQL	*Structured Query Language*
TFG	*Trabajo Final de Grado*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
WAR	*Web Application Archive*
XML	*eXtensible Markup Language*

7. Bibliografía

Referencias

[1] https://es.wikipedia.org/wiki/Controversia_de_las_vacunas (07/03/2018)

[2] http://www.elmundo.es/cataluna/2015/06/27/558e5fb2e2704ea41e8b4576.html (07/03/2018)

[3] https://buenavibra.es/movida-sana/salud/italia-sarampion-movimientos-antivacunas (16/03/2018)

[4] https://es.wikipedia.org/wiki/Ciencia_de_datos (07/03/2018)

[5] https://es.wikipedia.org/wiki/Interfaz_de_programacion_de_aplicaciones (07/03/2018)

[6] https://es.wikipedia.org/wiki/NoSQL (07/03/2018)

[7] https://es.wikipedia.org/wiki/Macrodatos (07/03/2018)

[8] https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software) (16/03/2018)

[9] https://trello.com (16/03/2018)

[10] https://es.wikipedia.org/wiki/Base_de_datos_orientada_a_grafos (23/05/2018)

[11] http://www.oracle.com/technetwork/java/index.html (23/05/2018)

[12] https://spring.io/ (23/05/2018)

[13] https://es.wikipedia.org/wiki/JavaScript (23/05/2018)

[14] https://jquery.com/ (23/05/2018)

[15] https://es.wikipedia.org/wiki/Dise%C3%B1o_centrado_en_el_usuario (24/05/2018)

[16] https://es.wikipedia.org/wiki/Caso_de_uso (24/05/2018)

[17] https://developer.twitter.com/en/docs/tweets/search/overview (26/05/2018)

[18] https://developers.google.com/youtube/v3/getting-started?hl=es-419#quota (26/05/2018)

[19] https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador (29/05/2018)

[20] https://docs.microsoft.com/es-es/azure/architecture/best-practices/api-design (29/05/2018)

[21] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html (29/05/2018)

[22] https://git-scm.com/ (29/05/2018)

[23] https://maven.apache.org/ (29/05/2018)

[24] https://spring.io/tools (29/05/2018)

[25] http://tomcat.apache.org/ (29/05/2018)

[26] https://projects.spring.io/spring-boot/ (29/05/2018)

[27] https://developers.google.com/youtube/v3/getting-started#before-you-start (29/05/2018)

[28] http://js.cytoscape.org/#performance (31/05/2018)

[29] https://developers.google.com/youtube/v3/ (02/06/2018)

[30] https://developers.google.com/youtube/v3/docs/search/list (02/06/2018)

[31] https://developers.google.com/youtube/v3/docs/videos/list (02/06/2018)

[32] https://developers.google.com/youtube/v3/docs/channels/list (02/06/2018)

[33] https://es.wikipedia.org/wiki/ACID (02/06/2018)

[34] http://cassandra.apache.org/ (02/06/2018)

[35] https://www.mongodb.com/ (02/06/2018)

[36] http://couchdb.apache.org/ (02/06/2018)

[37] https://neo4j.com/ (02/06/2018)

[38] https://es.wikipedia.org/wiki/Base_de_datos_orientada_a_grafos (02/06/2018)

[39] https://projects.spring.io/spring-data-neo4j/ (02/06/2018)

[40] https://es.wikipedia.org/wiki/Java_Persistence_API (02/06/2018)

[41] https://es.wikipedia.org/wiki/Objeto_de_acceso_a_datos (02/06/2018)

[42] https://es.wikipedia.org/wiki/CRUD (02/06/2018)

[43] https://es.wikipedia.org/wiki/Plain_Old_Java_Object (02/06/2018)

[44] https://es.wikipedia.org/wiki/SOLID (02/06/2018)

[45] https://developers.google.com/api-client-library/java/apis/youtube/v3 (03/06/2018)

[46] https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc (03/06/2018)

[47] http://api.jquery.com/jquery.ajax/ (03/06/2018)

[48] https://www.thymeleaf.org/ (03/06/2018)

[49] https://startbootstrap.com/template-overviews/sb-admin-2/ (03/06/2018)

[50] https://projects.spring.io/spring-security/ (03/06/2018)

[51] http://js.cytoscape.org/ (03/06/2018)

[52] https://d3js.org/ (03/06/2018)

[53] http://sigmajs.org/ (03/06/2018)

8. Anexos

8.1. Manual de instalación y requerimientos

Requerimientos de *hardware*:

- **Sistema operativo:** La aplicación puede funcionar tanto en sistemas *Windows*, *Linux* o *Mac*, en cada caso hará falta la instalación de las distribuciones correctas según el sistema operativo escogido.
- **Memoria RAM:** Se recomienda un mínimo de 256Mb de memoria RAM dedicados a la ejecución de la aplicación.
- **Espacio de disco:** La aplicación en si misma tiene un peso de 30Mb en disco pero dependiendo del uso requerido de la aplicación y del volumen de información a recolectar, el espacio en disco necesario puede variar. Como dato para extrapolar, un volumen de 500 vídeos y 100 canales ocupa un espació en disco aproximado de 200Mb.

Requerimientos de *software*:

- ***Java JDK*:** Versión 8 o superior.
- ***Neo4J Server*:** Versión 3.3.5 o superior.
- **Contenedor de *servlets***: Puede utilizarse cualquier distribución compatible con Java 8, por ejemplo *Tomcat 8*.

Los pasos para realizar la instalación de la aplicación son los siguientes:

1- Instalación de Java JDK: Las instrucciones para la instalación de la misma pueden encontrarse detallas en la pagina web oficial: https://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html.

2- Instalación de *Neo4J Server*: La instalación consiste simplemente en obtener la distribución deseada y descomprimirla. Se puede arrancar como servicio ejecutando el comando: *bin/neo4j start*. Una vez el servicio esta arrancado, se puede visitar su consola de administración en cualquier explorador web mediante la url *http://localhost:7474* en donde al acceder por primera vez se preguntara para crear un nuevo password para el usuario administrador *"neo4j"*. Mas detalles sobre su instalación se pueden encontrar en la web oficial: https://neo4j.com/docs/operations-manual/current/installation/.

3- Instalación contenedor de *servlets*: La instalación dependerá del sistema operativo y del contenedor escogido. En el caso de *Tomcat 8*: https://tomcat.apache.org/tomcat-8.0-doc/setup.html.

4- Configurar conexión base de datos: La conexión de la aplicación con la base de datos Neo4J esta configurada mediante el archivo de propiedades 'application.properties', en donde los valores por defecto son:

```
#Neo4j connection
spring.data.neo4j.uri=bolt://localhost
spring.data.neo4j.username=neo4j
spring.data.neo4j.password=youtubecrawlertool
```

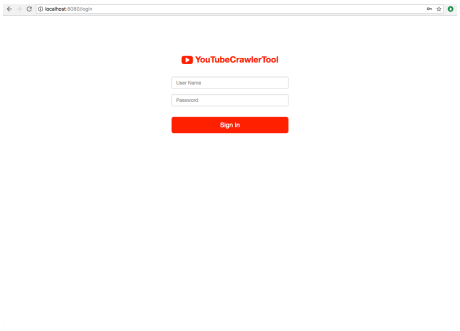
Los valores de esta configuración se pueden sobrescribir mediante un nuevo archivo de propiedades que debe de ser indicado como opciones de Java en el inició de contenedor de *servlets*:
JAVA_OPTS="-Dspring.config.location=$location_to_the_overriden_properties"

5- Instalar fichero WAR en el contenedor de *servlets*: El ejecutable de la aplicación Web en formato de fichero WAR se entrega junto con esta memoria. Para hacer funcionar la aplicación esta debe ser instalada en un contenedor de *servlets*. Seguir las instrucciones indicadas según el contenedor escogido para realizar su instalación. En el caso de *Tomcat 8*: https://tomcat.apache.org/tomcat-8.0-doc/deployer-howto.html.

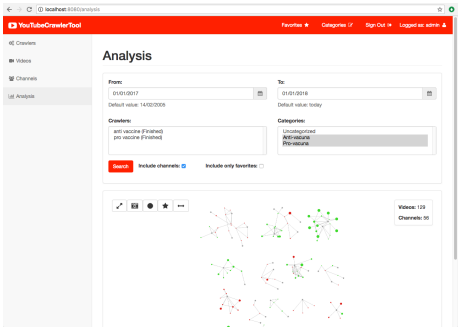
6- Abrir un explorador WEB: Y acceder a la aplicación recientemente instalada. Por defecto en *Tomcat 8* si se instala la aplicación en la raíz esta esta disponible en http://localhost:8080/.

8.2. Ilustraciones de la aplicación

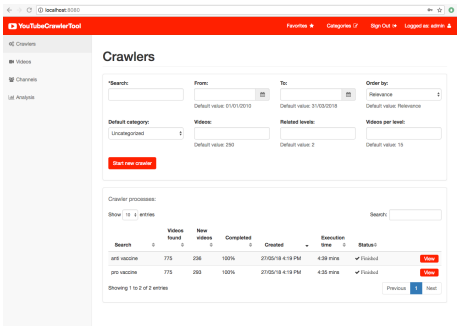
A continuación se incluyen imágenes ilustrativas de cada una de las vistas de la aplicación en funcionamiento:



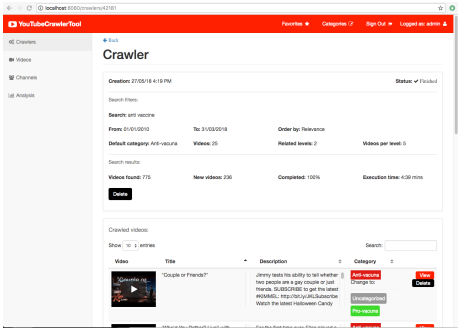
(a) Login.html



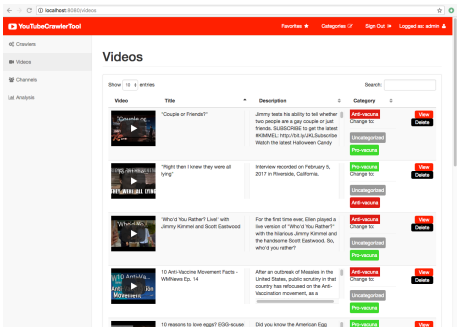
(b) Analysis.html



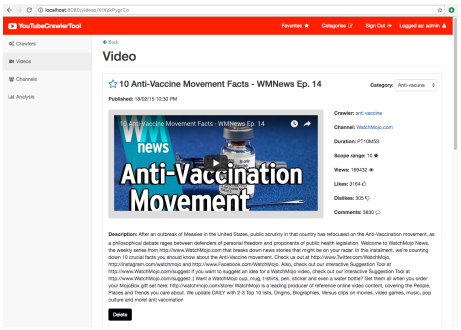
(c) Crawlers.html



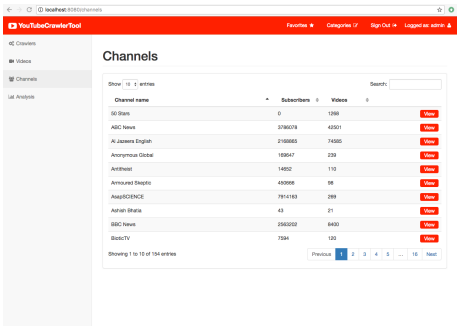
(d) Crawler.html



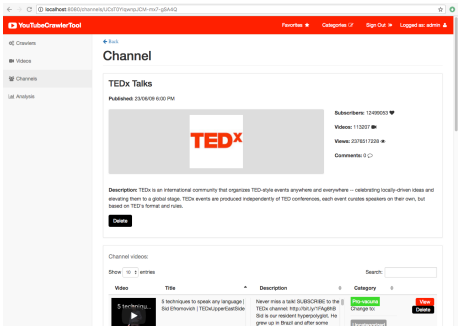
(e) Videos.html



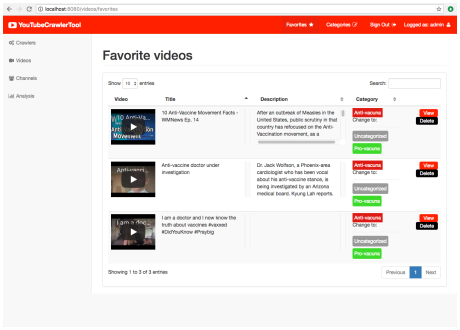
(f) Video.html



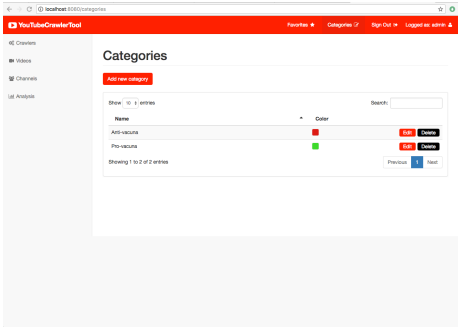
(g) Channels.html



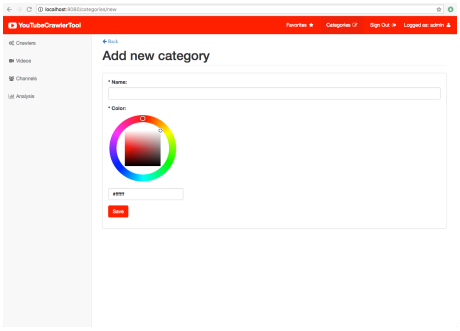
(h) Channel.html



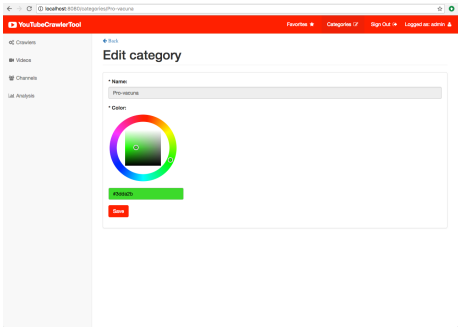
(i) Favorites.html



(j) Categories.html



(k) NewCategory.html



(l) EditCategory.html

Figura 31: Puesta en funcionamiento

8.3. Pruebas de integración

Resultados globales

YouTubeCrawlerTool	Resultado	
EVALUACIÓN	V	72
100 %	X	0

Componente *User session*

User session	Resultado	
EVALUACIÓN	V	5
100 %	X	0

Responsable	Javier Sánchez Mendoza		
Componente	User session		
Caso de uso	UC_1: Login		
Código	Acciones a verificar	Resultado esperado	Verificación
1	Identificarse al sistema como usuario valido.	Usuario identificado.	V
2	Identificarse al sistema ya estando identificado.	Acción no permitida por la aplicación.	V
3	Introducir datos nuevos o incorrectos.	Usuario no identificado.	V
EVALUACIÓN		V	3
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	User session		
Caso de uso	UC_1-1: Logout		
Código	Acciones a verificar	Resultado esperado	Verificación
1	Cerrar sesión de un usuario identificado.	El usuario actual ya no esta identificado en el sistema.	V
2	Cerrar sesión de un usuario no identificado.	Acción no permitida por la aplicación.	V
EVALUACIÓN		V	2
100 %		X	0

Componente *Crawler*

Crawler	Resultado	
EVALUACIÓN	V	18
100 %	X	0

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_2: View all crawlers statistics		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
NO IMPLEMENTADO			

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_3: List all crawlers		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder al listado de crawlers cuando aun no existe ninguno en el sistema.	No se debe mostrar ningún resultado.	V
2	Como usuario acceder al listado de crawlers cuando existen algunos en el sistema.	Se debe mostrar el estado actualizado de todos los crawlers que existan en el sistema.	V
EVALUACIÓN		V	2
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_4: Start new crawler process		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado iniciar un proceso de crawler.	Acción no permitida por la aplicación.	V
2	Como usuario identificado iniciar un nuevo proceso de crawler sin indicar el termino a buscar.	Acción no permitida por la aplicación.	V
3	Como usuario identificado iniciar un nuevo proceso de crawler indicando una fecha mayor en el campo “from” que el que hay informado en el campo “to”	Acción no permitida por la aplicación.	V
4	Como usuario identificado iniciar un nuevo proceso de crawler indicando el termino de búsqueda.	Nuevo proceso de crawler inicializado y almacenado con las opciones introducidas o con las de por defecto en los campos no requeridos.	V
EVALUACIÓN		V	4
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_5: Stop crawler process		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado parar un proceso de crawler.	Acción no permitida por la aplicación.	V
2	Como usuario identificado parar un proceso de crawler que este en un estado distinto a “Running”.	Acción no permitida por la aplicación.	V
3	Como usuario identificado parar un proceso de crawler que este en estado “Running”.	El proceso pasa al estado “Stopping” y en acabar la extracción de videos de la pagina actual del crawler el proceso pasa al estado “Stopped” si hay mas paginas para extraer, en caso contrario el crawler pasa al estado “Finished”.	V
EVALUACIÓN		V	3
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_6: Play crawler process		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado reiniciar un proceso de crawler.	Acción no permitida por la aplicación.	V
2	Como usuario identificado reiniciar un proceso de crawler que este en un estado distinto a “Stopped”.	Acción no permitida por la aplicación.	V
3	Como usuario identificado reiniciar un proceso de crawler que este en estado “Stopped”.	El proceso pasa al estado “Running” y se continua recolectando videos.	V
EVALUACIÓN		V	3
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_7: View crawler process details		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario accedo a la vista en detalle de un proceso de crawler.	Es posible ver la información actualizada del proceso de crawler seleccionado.	V
EVALUACIÓN		V	1
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_8: View crawler process statistics		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
NO IMPLEMENTADO			

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	UC_9: Delete crawler process		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado eliminar un proceso de crawler.	Acción no permitida por la aplicación.	V
2	Como usuario identificado eliminar un proceso de crawler en estado “Running” o “Stopping	Acción no permitida por la aplicación.	V
3	Como usuario identificado eliminar un proceso de crawler en estado “Finished”, “Stopped”, “Blocked” o “Error”.	El proceso de crawler como los vídeos que ha recolectado son eliminados del sistema además de los canales que como resultado de la acción no tengan ningún vídeo asociado.	V
EVALUACIÓN 100 %		V	3
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Crawler		
Caso de uso	CU_10: List all crawler process videos		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder al listado de videos recolectados por un crawler cuando el crawler no ha recolectado ningún vídeo.	No se debe mostrar ningún resultado.	V
2	Como usuario acceder al listado de videos recolectados por un crawler cuando el crawler ha recolectado algún vídeo.	Se deben mostrar todos y solo los vídeos recolectados por el proceso de crawler.	V
EVALUACIÓN 100 %		V	2
		X	0

Componente *Video*

Video	**Resultado**	
EVALUACIÓN 100 %	V	22
	X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_11-1: List all videos		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder al listado de vídeos cuando aun no existe ninguno en el sistema.	No se debe mostrar ningún resultado.	V
2	Como usuario acceder al listado de vídeos cuando existen algunos en el sistema.	Se deben mostrar todos los vídeos que existan en el sistema.	V
EVALUACIÓN 100 %		V	2
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_11: View video detail		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder a la vista en detalle de un video.	Es posible ver la información del video seleccionado.	V
EVALUACIÓN 100 %		V	1
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_12: Categorize a video		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado cambiar la categoría de un vídeo.	Acción no permitida por la aplicación.	V
2	Como usuario identificado cambiar la categoría de un vídeo.	El vídeo tiene asignada la nueva categoría seleccionada.	V
EVALUACIÓN 100 %		V	2
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_13: Delete a video		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado eliminar un vídeo.	Acción no permitida por la aplicación.	V
2	Como usuario identificado borrar un vídeo.	Acción no permitida por la aplicación.	V
3	Como usuario identificado borrar un vídeo el cual su canal solo contiene el video a borrar.	El vídeo ya no esta disponible en el sistema al igual que su canal.	V
EVALUACIÓN 100 %		V	3
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_14: List all related videos		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder al listado de vídeos relacionados de un vídeo cuando el vídeo no tiene ningún vídeo relacionado.	No se debe mostrar ningún resultado.	V
2	Como usuario acceder al listado de vídeos relacionados de un vídeo cuando el vídeo si tiene algún vídeo relacionado.	Se deben mostrar todos y solo los vídeos a los cuales el vídeo actual este relacionado.	V
EVALUACIÓN 100 %		V	2
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_15: Find more related videos		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado iniciar un nuevo proceso de crawler sobre un vídeo para encontrar vídeos relacionados.	Acción no permitida por la aplicación.	V
2	Como usuario identificado iniciar un nuevo proceso de crawler indicando los parámetros de búsqueda sobre un vídeo en concreto para encontrar sus relacionados.	Un nuevo proceso de crawler de búsqueda relacionada debe haberse dado de alta con la información introducida y estar un funcionamiento.	V
3	Como usuario identificado iniciar un nuevo proceso de crawler sin indicar ningún parámetro de búsqueda sobre un vídeo en concreto para encontrar sus relacionados.	Un nuevo proceso de crawler de búsqueda relacionada debe haberse dado de alta con los parámetros por defecto y estar un funcionamiento.	V
EVALUACIÓN		V	3
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_15-1: Add a video as a favorite		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado marcar un vídeo como favorito.	Acción no permitida por la aplicación.	V
2	Como usuario identificado marcar un vídeo como favorito un vídeo que ya esta marcado como favorito.	Acción no permitida por la aplicación.	V
3	Como usuario identificado marcar un vídeo como favorito.	El vídeo es añadido al listado de vídeos favoritos del usuario.	V
EVALUACIÓN		V	3
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_15-2: Delete a video as a favorite		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado eliminar un vídeo como favorito.	Acción no permitida por la aplicación.	V
2	Como usuario identificado eliminar un vídeo como favorito un vídeo que no esta marcado como favorito.	Acción no permitida por la aplicación.	V
3	Como usuario identificado eliminar un vídeo como favorito un vídeo que esta marcado como favorito.	El vídeo es eliminado al listado de vídeos favoritos del usuario.	V
EVALUACIÓN		V	3
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Video		
Caso de uso	CU_15-3: List all favorite videos		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado ver el listado de vídeos favoritos.	Acción no permitida por la aplicación.	V
2	Como usuario identificado ver el listado de vídeos favoritos cuando el usuario aun no ha marcado ningún vídeo como favorito.	No se debe mostrar ningún resultado	V
3	Como usuario identificado ver el listado de vídeos favoritos cuando el usuario ha marcado algún vídeo como favorito.	Se debe mostrar un listado con todos los vídeos marcados como favorito por parte del usuario.	V
EVALUACIÓN 100 %		V	3
		X	0

Componente *Channel*

Channel	**Resultado**	
EVALUACIÓN 100 %	V	7
	X	0

Responsable	Javier Sánchez Mendoza		
Componente	Channel		
Caso de uso	UC_16: View all channels statistics		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
NO IMPLEMENTADO			

Responsable	Javier Sánchez Mendoza		
Componente	Channel		
Caso de uso	CU_17: List all channels		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder al listado de canales cuando aun no existe ninguno en el sistema.	No se debe mostrar ningún resultado.	V
2	Como usuario acceder al listado de canales cuando existen algunos en el sistema.	Se deben mostrar todos los canales que existan en el sistema.	V
EVALUACIÓN 100 %		V	2
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Channel		
Caso de uso	CU_18: View channel details		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder a la vista en detalle de un canal.	Es posible ver la información del canal seleccionado.	V
EVALUACIÓN 100 %		V	1
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Channel		
Caso de uso	UC_19: View channel statistics		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
NO IMPLEMENTADO			

Responsable	Javier Sánchez Mendoza		
Componente	Channel		
Caso de uso	CU_20: Delete a channel		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado eliminar un canal.	Acción no permitida por la aplicación.	V
2	Como usuario identificado borrar un canal.	El canal y todos los vídeos con los que este relacionado ya no están disponibles en el sistema.	V
EVALUACIÓN 100 %		V	2
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Channel		
Caso de uso	CU_21: List all channel videos		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder al listado de vídeos de un canal en concreto cuando el canal no tiene ningún vídeo.	Acción no permitida por la aplicación debido a que, si un canal no tiene ningún vídeo asociado es eliminado del sistema.	V
2	Como usuario acceder al listado de videos de un canal en concreto cuando el canal tiene ningún vídeo.	Se deben mostrar todos y solo los vídeos asociados con el canal seleccionado.	V
EVALUACIÓN 100 %		V	2
		X	0

Componente *Category*

Category	**Resultado**	
EVALUACIÓN 100 %	V	12
	X	0

Responsable	Javier Sánchez Mendoza		
Componente	Category		
Caso de uso	CU_22: List all categories		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario acceder al listado de categorías cuando aun no existe ninguna en el sistema.	No se debe mostrar ningún resultado.	V
2	Como usuario acceder al listado de categorías cuando existe alguna en el sistema.	Se deben mostrar todos las categorías que existan en el sistema.	V
EVALUACIÓN 100 %		V	2
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Category		
Caso de uso	CU_23: Add new category		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado dar de alta una nueva categoría.	Acción no permitida por la aplicación.	V
2	Como usuario identificado dar de alta una nueva categoría sin especificar el nombre.	No se debe dar de alta una nueva categoría en el sistema.	V
3	Como usuario identificado dar de alta una nueva categoría sin especificar el color.	No se debe dar de alta una nueva categoría en el sistema.	V
4	Como usuario identificado dar de alta una nueva categoría sin especificando un nombre de categoría que ya existe previamente en el sistema.	No se debe dar de alta una nueva categoría en el sistema.	V
5	Como usuario identificado dar de alta una nueva categoría especificando nombre y color.	Una nueva categoría debe darse de alta en el sistema.	V
EVALUACIÓN		V	5
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Category		
Caso de uso	CU_24: Edit a category		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado editar la información de una categoría.	Acción no permitida por la aplicación.	V
2	Como usuario identificado modificar el nombre de una categoría.	Acción no permitida por la aplicación.	V
3	Como usuario identificado modificar el color de una categoría.	La categoría previamente existente en el sistema tiene ahora asignado un nuevo color.	V
EVALUACIÓN		V	3
100 %		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Category		
Caso de uso	CU_25: Delete a category		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario no identificado eliminar una categoría.	Acción no permitida por la aplicación.	V
2	Como usuario identificado eliminar una categoría.	La categoría así como todos los vídeos categorizados con la misma categoría que se esta eliminando son eliminados del sistema además de los canales que como resultado de la acción no tengan ningún vídeo asociado.	V
EVALUACIÓN		V	2
100 %		X	0

Componente *Analysis*

Analysis	Resultado	
EVALUACIÓN 100 %	V	8
	X	0

Responsable	Javier Sánchez Mendoza		
Componente	Analysis		
Caso de uso	CU_26: Search and show results on a graph		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario realizar una búsqueda de análisis sin indicar ningún filtro.	En el grafo resultante deben aparecer todos los vídeos existentes en el sistema y sus relaciones.	V
2	Como usuario realizar una búsqueda de análisis indicando diferentes combinaciones de filtros.	En el grafo resultante deben aparecer solo los vídeos existentes en el sistema que sean resultado de la búsqueda introducida y sus relaciones.	V
3	Como usuario realizar una búsqueda de análisis cuando no hay ningún vídeo en el sistema.	No se debe mostrar ningún resultado.	V
4	Como usuario realizar una búsqueda de análisis indicando una combinación de filtros que no produzca ningún resultado.	No se debe mostrar ningún resultado.	V
5	Como usuario realizar una búsqueda de análisis indicando que se quieren obtener los canales.	En el grafo resultante deben aparecer los vídeos resultado de la búsqueda introducida junto con sus canales de publicación y sus relaciones.	V
EVALUACIÓN 100 %		V	5
		X	0

Responsable	Javier Sánchez Mendoza		
Componente	Analysis		
Caso de uso	CU_27: Export search results		
Código	**Acciones a verificar**	**Resultado esperado**	**Verificación**
1	Como usuario exportar a fichero csv los vídeos resultantes de una búsqueda de análisis.	Los vídeos obtenidos como resultado de una búsqueda de análisis son exportados a un fichero csv y descargado por el usuario.	V
2	Como usuario exportar a fichero csv los canales resultantes de una búsqueda de análisis.	Los canales obtenidos como resultado de una búsqueda de análisis son exportados a un fichero csv y descargado por el usuario.	V
3	Como usuario exportar a fichero csv las relaciones entre nodos resultantes de una búsqueda de análisis.	Las relaciones entre nodos obtenidos como resultado de una búsqueda de análisis son exportados a un fichero csv y descargado por el usuario.	V
EVALUACIÓN 100 %		V	3
		X	0