Universidad Zaragoza
1542

Proyecto Fin de Carrera

Aragón Open SocialData

Una API de datos sociales abiertos y estructurados sobre la Comunidad Autónoma de Aragón

Autor

Alberto Alcolea Ursua

Director y ponente

Director: Gonzalo Ruiz Manzanares
Ponente: Sergio Ilarri Artigas

Escuela de Ingeniería y Arquitectura
2014

Aragón Open SocialData

El auge de dispositivos conectados a Internet y la explosión de las redes sociales han hecho crecer de forma exponencial la información creada por los ciudadanos en la Red. Este intercambio de contenidos de todo tipo (datos, opiniones, estados de ánimo...) entre sus usuarios ha causado una enorme repercusión en el impacto que dicha información tiene en la vida diaria de las personas, influyendo cada vez más en su comportamiento, hábitos de vida, decisiones de compra, opiniones políticas, etc.

En este contexto, el Gobierno de Aragón, junto con el Instituto de Biocomputación y Física de Sistemas Complejos (BIFI) de la Universidad de Zaragoza, planteó la realización de una plataforma de datos sociales abiertos relacionados con la Comunidad Autónoma de Aragón. Su objetivo era poner a disposición del público una gran cantidad de datos estructurados que pueden ser de utilidad para la sociedad, tanto en términos de transparencia, como para su explotación por parte de empresas que contribuyan de este modo al crecimiento económico.

Para lograr este objetivo, en este Proyecto de Fin de Carrera (PFC) se ha desarrollado un sistema de recopilación y estructuración de información obtenida de las principales redes sociales y portales web (wikis, blogs...) junto con una API REST abierta que ofrece los datos procesados al público.

La plataforma está formada por los siguientes niveles lógicos, desarrollados por el proyectando:

- Módulo de escucha en la red de distintas fuentes sociales heterogéneas mediante sistemas *ad hoc* adaptados a cada fuente y a sus limitaciones y normativas.
- API REST que permite realizar consultas sobre los datos recogidos filtrándolos por distintos parámetros y devolviendo la información de forma estructurada codificándola en formato JSON.
- Interfaz web interna para facilitar la gestión y monitorización de los sistemas de escucha de fuentes y los contenidos servidos. Por un lado, es capaz de añadir de forma dinámica más fuentes al sistema o detenerlas y, por otro, permite tener un control sobre la información recopilada, el estado de los módulos de recolección de información y el uso de la API.

Además, la plataforma ha sido desplegada en un entorno distribuido escalable.

Agradecimientos

A los miembros del BIFI por toda la compañía y el apoyo que me han brindado durante estos tres meses, especialmente a Gonzalo por su paciencia y su inestimable ayuda que me ha hecho crecer como persona y como profesional.

A todos los compañeros de batalla que me acompañaron a lo largo de la carrera y que han compartido a mi lado esta aventura.

A todos aquellos que, a lo largo de mi vida, me ayudaron a descubrir mi vocación y consiguieron apasionarme con el mundo de la tecnología y de la informática.

Y, en definitiva, gracias a todas las personas que me han apoyado durante todos estos años, muy especialmente a mis padres, porque sin ellos no habría llegado hasta aquí.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este documento, junto con los anexos correspondientes, recoge toda la información sobre el trabajo realizado en el PFC[1] titulado "Aragón Open SocialData", desarrollado por el alumno Alberto Alcolea Ursua durante el periodo comprendido entre los meses de marzo y junio del año 2014 para la Diputación General de Aragón (DGA) en el Instituto de Biocomputación y Física de Sistemas Complejos (BIFI).

Aragón Open SocialData surge como la necesidad de poner a disposición de los ciudadanos la información relacionada con la Comunidad Autónoma de Aragón generada por la sociedad en Internet.

En este capítulo introductorio se expone brevemente qué llevó a tomar la decisión de crear esta plataforma y por qué tiene interés tanto para la administracción pública como para los ciudadanos y las empresas. También se resume el contenido y la organización del resto del documento.

1.1. Motivación

Datos abiertos (*open data* en inglés) es una iniciativa global cuyo objetivo es conseguir que determinada información, especialmente la que poseen las administraciones públicas, esté disponible para los ciudadanos. La publicación de estos datos se realiza de forma abierta y reutilizable, es decir, sin restricciones, paten-

[1]Proyecto de Fin de Carrera.

tes u otros mecanismos de control, de forma que cualquier persona o entidad que lo desee pueda utilizar esta información para su consulta, para enriquecerla con nuevos datos o para generar aplicaciones y servicios a partir de ella [1, 2].

La publicación de datos de calidad, servidos mediante formatos estructurados que pueden ser procesados fácilmente por sistemas automáticos, simplifica las tareas de transformación, mediante las cuales terceras personas pueden añadir valor adicional a la información ya disponible, por ejemplo, creando aplicaciones, resúmenes o informes.

El estudio de la enorme cantidad de información colectiva (*Big Data* [3]) que existe en Internet tiene gran utilidad para conocer mejor las opiniones que se están fraguando en la sociedad con respecto a muy diversos temas, tanto en una vertiente de corto como de largo plazo. Este tipo de análisis está directamente relacionado con la actividad social, la propagación de la información y la repercusión de determinados grupos sociales. Además, en el vertiginoso mundo de la información actual, no sólo importa qué se dice, sino también quién lo dice y cuándo se dice.

Dentro del contexto social, nos encontramos en la era del auge de las redes sociales. Vivimos continuamente conectados a Internet. Llevamos un *smartphone*[2] a todas partes y estamos, continuamente, generando información en la Red. Facebook cuenta con más de 750 millones de usuarios únicos, Twitter posee alrededor de 500 millones de cuentas generando en torno a 340 millones de *tweets* al día y realizando más de 1.600 millones de búsquedas diarias. [4, 5].

Otras comunidades autónomas españolas, como la Comunidad Foral de Navarra[3] o el País Vasco[4], han desarrollado soluciones Open Data similares, lo que refleja el interes que supone la presencia de plataformas Open Data sociales en la Red.

Debido a la existencia de esta enorme cantidad de información generada por la sociedad en la Red y al gran interés que supone su estudio se ha tomado la decisión de crear una plataforma Open Data social que facilite las tareas de recopilación y análisis de datos sociales relacionados con Aragón.

[2]Teléfono inteligente.

[3]Servicio Open Data Social de la Comunidad Foral de Navarra:
`http://www.gobiernoabierto.navarra.es/es/open-data`

[4]Servicio Open Data Social del País Vasco:
`http://opendata.euskadi.net/w79-home/es`

1.2. Entorno de aplicación

En Julio de 2013 la Diputación General de Aragón (DGA) sacó a licitación pública un proyecto sobre el desarrollo y despliegue de un servicio de datos abiertos sociales relacionados con Aragón. El proyecto fue concedido al grupo de desarrollo, innovación y transferencia de tecnología del Instituto de Biocomputación y Física de Sistemas Complejos (BIFI) cuya propuesta describía una plataforma de *escucha activa* en la Red con las siguientes funcionalidades:

- El sistema de *escucha* extraería la información vinculada de alguna forma con la Comunidad Autónoma de Aragón publicada en las redes sociales más relevantes y otros portales webs como blogs, wikis, páginas de prensa, etc.
- Los datos recopilados deberían ser tratados y analizados con el fin de agregar y estructurar la información recopilada.
- Toda la información recogida de las fuentes se almacenaría en un sistema de persistencia para poder servirla después al público mediante un servicio de datos a través de una API (*Application Programming Interface*[5]). Esta API debería ser parametrizable y permitiría realizar consultas atendiendo a distintos criterios de búsqueda.
- Los resultados de este servicio de datos se transmitirían al usuario en algún formato estructurado como JSON[6], XML[7], RDF[8] y otros similares con el fin de ser fácilmente procesables por sistemas automáticos.

La propuesta fue aceptada pero su desarrollo se aplazó durante varios meses por temas ajenos al BIFI hasta el mes de noviembre. Desde entonces se trabajó a contrarreloj para implementar y presentar públicamente en febrero de 2014 un prototipo muy básico con tan sólo un subconjunto pequeño de fuentes.

A su vez, el BIFI estaba implicado desde finales de 2012 en un proyecto INNPACTO del Ministerio de Economía y Competitividad que compartía la necesidad de extraer datos sociales de Internet: el proyecto SEPS (Sistema Experto de Probabilidad y Severidad de Incidentes en Red)[9], el cual pretende ser un sistema

[5]Una API, *Application Programming Interface* o Interfaz de Programación de Aplicaciones en castellano, es un conjunto de funciones y procedimientos que ofrece una librería para ser utilizados por otro software como una capa de abstracción.

[6]*JavaScript Object Notation.*

[7]*eXtensible Markup Language.*

[8]*Resource Description Framework.*

[9]Sitio web del proyecto SEPS: `http://www.proyecto-seps.es/`

automático para identificar la severidad reputacional y la repercusión social de los incidentes sobre una red de distribución de energía eléctrica.

Con el objetivo de reaprovechar el trabajo realizado en ambos proyectos y de facilitar su mantenimiento, se reimpulsó el desarrollo de la plataforma Aragón Open SocialData. Fue en ese momento cuando entré a formar parte del grupo de desarrollo del BIFI bajo la dirección de Gonzalo Ruiz Manzanares, director del departamento de desarrollo del BIFI. Mi trabajo ha consistido en el diseño, desarrollo y despliegue del servicio a partir del prototipo inicial y de los requisitos previos analizados y acordados conjuntamente por el BIFI y la DGA.

1.2.1. BIFI

El Instituto de Biocomputación y Física de Sistemas Complejos (BIFI) es un centro de investigación perteneciente a la Universidad de Zaragoza formado por investigadores interdisciplinares de la Universidad de Zaragoza y de otras instituciones españolas y extranjeras. Su objetivo es desarrollar investigación competitiva en las áreas de computación aplicada a la física de sistemas complejos y a los modelos biológicos y sociales.

Además de la investigación en ciencia básica, un punto fundamental del instituto es la transferencia de tecnología entre la universidad y el mundo empresarial, contexto en el que se centra este PFC.

Para llevar a cabo estos objetivos, el BIFI cuenta con el trabajo de investigadores de áreas diferentes cuya colaboración está llevando a sinergias muy significativas. En particular, expertos en supercomputación, físicos trabajando en ciencia de materiales, química cuántica o redes complejas, y biólogos trabajando en problemas estructurales como desarrollo de fármacos y plegamiento de proteínas.

1.2.2. DGA

La Diputación General de Aragón o Gobierno de Aragón (cuya abreviatura es DGA) es el órgano de gobierno de la Comunidad Autónoma de Aragón.

Desde el año 2012 el Gobierto de Aragón está tratando de realizar un conjunto de plataformas Open Data[10] con el objetivo de generar desarrollo económico y

[10]Servicio Open Data de la Diputación General de Aragón: `http://opendata.aragon.es/`

mejorar la transparencia siguiendo la política de "gobierno abierto".

La información disponible proviene de diversas fuentes institucionales, como la ofrecida por la plataforma AragoPedia[11], y sociales, como la ofrecida en la plataforma desarrollada en este PFC.

1.3. Objetivos y alcance del proyecto

Tal y como se estableció en la propuesta presentada a concurso público por la DGA, la finalidad de este proyecto es facilitar el acceso a la información social que, o bien se genera dentro de las fronteras de la Comunidad Autónoma de Aragón, o bien está relacionada con alguna de las tendencias del momento de la comunidad.

El piloto inicial presentado por el BIFI constaba sólo de cuatro módulos de escucha para las siguientes fuentes: Facebook, Twitter, Youtube e Instragram y un prototipo muy básico del módulo de la API que ofrece los resultados a terceros. Todos estos módulos han sido reescritos por el proyectando de manera parcial, o incluso total, para adaptarlos al sistema completo diseñado.

Mi trabajo ha consistido en ampliar dicho prototipo inicial cumpliendo los siguientes objetivos fijados por el BIFI:

- Realizar un estudio previo de las fuentes sociales que no se encontraban en el piloto presentado en febrero para ver qué información es la más relevante para el proyecto, analizar las mejores técnicas para su extracción, decidir qué criterios de filtrado son los más adecuados para cada fuente, etc.

- Completar el diseño de la base de datos del prototipo y adaptarlo a las nuevas fuentes añadidas, las consultas de la API y la visualización de la información.

- Diseñar e implementar un conjunto de *scrapers*[12] capaces de rastrear y extraer información pública de las fuentes seleccionadas en el primer punto.

[11] AragoPedia es una iniciativa del proyecto Open Data de la DGA para acercar la información institucional (padrón municipal de habitantes, superficie, datos de uso del suelo, etc.) a los ciudadanos mostrando los datos abiertos de Aragón a nivel regional y comarcal.

[12] Sistema software automático que permite extraer información relevante de un sitio web aplicando distintas técnicas.

- Diseñar e implementar un módulo automático (*daemon*[13]) encargado de la ejecución, detención y monitorización de los *scrapers* desarrollados en el punto anterior.
- Diseñar y desarrollar una interfaz web que permite controlar de forma manual el *daemon* y los *scrapers*. Además, esta interfaz web permite tener un control sobre la información servida censurando contenido ilegal o bloqueando usuarios que publiquen este tipo de contenidos en las fuentes o que, voluntariamente, no deseen ser escuchados. En el momento en que un contenido o un usuario es bloqueado la API deja de servir esta información y, en el caso del bloqueo de usuarios, los *scrapers* dejan de capturar el contenido publicado por éstos en las fuentes correspondientes.
- Diseñar y desarrollar una API REST[14] [6] que permite realizar consultas de acuerdo a los criterios acordados con la Diputación General de Aragón. En la sección 2.2 se explica en detalle cuáles fueron estos criterios.

Además de los puntos anteriores, durante el desarrollo del proyecto, por iniciativa del proyectando, se ampliaron los objetivos añadiendo los siguientes puntos:

- Realizar un estudio sobre métodos de monitorización de una API REST.
- Diseñar y desarrollar un sistema de monitorización de las consultas de la API a partir de los métodos estudiados en el punto anterior.
- Ampliar la interfaz web desarrollada añadiendo los sistemas de visualización siguientes:
  - Gráficos estadísticos del uso histórico de la API a partir del sistema de monitorización.
  - Gráficos estadísticos y mapas de calor de la información recopilada agregada por fuente.
  - Visualización en forma de histograma de la evolución de una palabra clave en los resultados recopilados de las fuentes.
  - Sistema de visualización del estado de ejecución de los *scrapers*. Permite visualizar en una línea de tiempo los momentos de inicio y fin de ejecución de cada *scraper* y el estado en el que se encuentra actualmente.

[13]Un *daemon*, conocido como *demonio* o *servicio* en castellano, es un tipo especial de proceso informático no interactivo, es decir, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario.

[14]Mecanismo que permite ejecutar métodos de librerías externas de forma remota mediante el envio de mensajes HTTP, tanto para realizar peticiones como para recibir respuestas.

En adición a lo anterior, el sistema ha sido desarrollado en forma de *framework*[15]. Este *framework* de *web scraping social* puede ser reutilizado y ampliado en otros proyectos que requieran realizar extracción de datos sociales de la red.

Además del prototipo inicial, ha sido responsabilidad de otros miembros del equipo de trabajo, y por tanto no ha formado parte de este PFC, el análisis de requisitos (capítulo 2) y el análisis de los aspectos jurídicos del proyecto (anexo D). En los siguientes capítulos, cuando se comenten algunos de estos aspectos que no han sido desarrollados por el proyectando, se indicará expresamente.

1.4. Contexto tecnológico

En esta sección se incluye el conjunto de tecnologías utilizadas a lo largo del proyecto. En el anexo A se explican más en detalle y se justifica el uso de cada una de ellas.

Tecnologías

Python 2.7 Lenguaje de programación principal utilizado en los módulos de extracción de datos de las fuentes (*scrapers*), la API REST, la interfaz web del lado del servidor y todos los módulos auxiliares.

Django 1.6 *Framework* escrito en Python para facilitar el desarrollo de aplicaciones web.

Celery 3.1 *Framework* escrito en Python para facilitar el desarrollo de colas de tareas asíncronas.

PosgreSQL 9.3 Sistema gestor de bases de datos.

PostGIS 2.1 Extensión de PostgreSQL para crear bases de datos espaciales[16].

JavaScript Lenguaje de programación del lado del cliente.

[15] Estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.

[16] Mecanismo de almacenamiento de información, similar a las bases de datos tradicionales, optimizado para almacenar datos geoespaciales y operar con ellos.

[17] *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML).

AJAX[17] Mecanismo mediante el cual los navegadores web realizan peticiones asíncronas a un servidor por medio de JavaScript. Se ha estudiado su funcionamiento para extraer contenido de las fuentes que hacen uso de él.

OAuth[18] Protocolo de autorización estandarizado y abierto que permite que un tercero acceda a los datos de un usuario en un servicio sin necesidad de conocer sus credenciales. Utilizado para extraer información de las fuentes.

AMQP[19] Protocolo de alto nivel orientado a la comunicación de nodos en entornos distribuidos mediante colas de mensajes [7].

Redis Sistema gestor de bases de datos en memoria. Se ha utilizado como servicio de colas del protocolo AMQP.

Apache Servidor web desde el que se sirve la API REST y la interfaz web.

HTML 5[20]**y CSS 3**[21] Lenguaje de marcado de hipertexto y lenguaje de hojas de estilo en cascada utilizados para maquetar la interfaz web.

LaTeX Lenguaje de composición de textos utilizado para la redacción de este documento.

Entorno de desarrollo

GNU/Linux (distintas distribuciones) Sistema operativo utilizado tanto en las máquinas de desarrollo como en las de producción.

Sublime Text 2/3 IDE[22] utilizado para todos los lenguajes.

vim Editor de texto utilizado para el desarrollo de scripts y para la gestión de los ficheros de configuración en los servidores de producción.

Mercurial Sistema de control de versiones.

Trello Tablero Kanban[23] colaborativo.

[18] *Open Authorization.*
[19] *Advanced Message Queuing Protocol.*
[20] *HyperText Markup Language.*
[21] *Cascading Style Sheets.*
[22] *Integrated Development Environment.*
[23] Kanban, en el contexto de desarrollo de software, hace referencia a un método de gestión de procesos visual basado en tarjetas situadas en un panel o pizarra. Estas tarjetas se mueven conforme el producto o subproducto avanza en la fase de producción. Es un método inspirado en el sistema de producción de Toyota y en el sistema de organización de la producción JIT (*Just-In-Time*, justo a tiempo) [8, 9].

Google Docs y Writelatex Redacción de la documentación.

Dia Gráficos y diagramas.

Google Chrome y Firefox Se han utilizado sus "herramientas para desarrolladores" para analizar el comportamiento de las fuentes de escucha, el contenido servidor por éstas y como *debuggers*[24] de la interfaz web desarrollada.

1.5. Estructura del documento

Este documento consta de dos partes: una memoria principal, donde se resume el trabajo realizado, el contexto en el que se ha efectuado y el modo de trabajo seguido; y una serie de anexos, donde se explica más detalladamente las decisiones de diseño y análisis tomadas, el esfuerzo dedicado al proyecto y los manuales de usuario. Por último, se incluye la bibliografía utilizada durante el desarrollo del proyecto y la redacción de esta memoria. Así pues, la estructura es la siguiente:

Parte I: Memoria

- Capítulo 1: motivación, contexto profesional y tecnológico del proyecto y objetivos y alcance del mismo.
- Capítulo 2: análisis de los requisitos del proyecto, realizado de manera conjunta entre el BIFI y la DGA antes de mi entrada al equipo de desarrollo.
- Capítulo 3: trabajo realizado por el proyectando y decisiones tomadas a lo largo del proyecto.
- Capítulo 4: metodologías utilizadas para la gestión del proyecto.
- Capítulo 5: conclusiones, líneas futuras y valoración personal.

Parte II: Anexos

- Anexo A: tecnologías utilizadas.

[24]Un debugger, depurador en castellano, es una herramienta para probar, localizar y depurar errores en una aplicación.

- Anexo B: planificación seguida a la hora de seleccionar la información más relevante de cada fuente en función de los datos suministrados y las limitaciones y normativas de cada red social y sitio web objetivo, planificación global del proyecto desglosada por objetivos y dedicación final dedicada a cada uno de ellos.
- Anexo C: diseño detallado de los módulos de la plataforma.
- Anexo D: aspectos jurídicos del proyecto estudiados por José Félix Muñoz Soro, miembro del Laboratorio Jurídico-Empresarial de la Universidad de Zaragoza.
- Anexo E: manual de usuario de la API.
- Anexo F: manual de usuario de la interfaz web.
- Anexo G: manual de instalación y despliegue de la plataforma.

En los siguientes capítulos se detallan las cuestiones más técnicas del proyecto.

Capítulo 2

Análisis previo

Antes de mi entrada al equipo de desarrollo de Aragón Open SocialData se realizó una labor de análisis de requisitos conjunta entre el BIFI y la DGA. También se seleccionaron las fuentes de interés a escuchar y se realizó un estudio sobre el estado y la disponibilidad legal de su contenido. En este capítulo quedan reflejadas todas las decisiones que fueron tomadas por el resto del equipo de desarrollo y, por tanto, no fueron responsabilidad del proyectando.

2.1. Estudio de las fuentes de datos

La conocida popularmente como *Web 2.0* ha permitido que miles de personas compartan día a día contenido en Internet. Dicho contenido tiene multiples formas (artículos, comentarios, fotografías, vídeos, etc) y está en continuo crecimiento.

La primera decisión tomada en el proyecto fue seleccionar qué fuentes sociales contenían información relevante para la plataforma (anexo B). También se estudió si, para cada una de las fuentes elegidas, se permite legalmente la extracción, almacenamiento y redistribución de la información generada en ellas. Para ello se contó con la ayuda de José Félix Muñoz Soro, miembro del Laboratorio Jurídico-Empresarial de la Universidad de Zaragoza, que aportó los conocimientos legales y jurídicos necesarios para tomar esta decisión. En el anexo D se detalla en profundidad el estudio jurídico realizado y los resultados obtenidos.

Tras este análisis se seleccionaron catorce fuentes con contenidos de interés para el proyecto: Twitter, Facebook, Google+, Youtube, Instagram, Pinterest, Fours-

quare, Flickr, Vimeo, Wikipedia, Wordpress, Blogger, Blogia y GitHub.

En el anexo B quedan también reflejados los campos recogidos para cada una de las fuentes.

Las fuentes anteriormente citadas son las únicas, de entre todas las analizadas, que permiten la reproducción y difusión, total o parcial, de su contenido. Fueron descartadas fuentes de medios de comunicación de prensa, indexadores de hoteles, portales de rutas, etc. por necesitar autorización expresa para poder usar su contenido. En cualquier caso, la responsabilidad legal del cotenido extraído, almacenado y servido por Aragón Open SocialData recae completamente sobre la DGA y no sobre el BIFI. La DGA autorizó al BIFI la realización de la plataforma tras la consulta jurídica y el equipo de desarrollo implementó el sistema solicitado.

2.2. Análisis de requisitos

Tras el acuerdo inicial con la Diputación General de Aragón (DGA) se procedió a ordenar y clasificar todos los requisitos que el proyecto Aragón Open SocialData debía cumplir.

La tabla 2.2 refleja los requisitos del proyecto acordados con la DGA clasificados en funcionales y no funcionales. Estos requisitos han sido la base sobre la que se ha realizado todo el trabajo posterior.

Además, para la realización de este proyecto, el BIFI destinó, de forma exclusiva, dos nodos (servidores). Uno de ellos está dedicado a la recogida y el análisis de datos y el otro es el responsable de procesar todas las operaciones solicitadas a través de la API. Cada uno de los nodos está configurado de acuerdo con las características enumeradas en la tabla 2.1 y ambos están conectados entre sí mediante un enlace *Gigabit Ethernet*[1].

Por último, en el acuerdo inicial se especificaron también una serie de consultas que la API debía soportar:

- Devolver las tendencias (*trendings*) del momento en Aragón.

[1] *Gigabit Ethernet*, también conocida como GigaE, es una ampliación del estándar *Ethernet* (concretamente la versión 802.3ab y 802.3z del IEEE) que consigue una capacidad de transmisión de 1 gigabit por segundo.

- Devolver contenido filtrado por fuente de datos, tipo de contenido, conversación (razón por la que fueron recogidos los datos), geoposición, periodo o palabra clave.
    - Dentro de los contenidos filtrados por geoposición se podrá filtrar por radio alrededor de un punto geográfico, por área geográfica (*bounding box*) o por radio alrededor de una localidad.
- Paginar los resultados.
- Solicitar los datos en crudo añadiendo un parámetro más a la petición.

Durante el diseño y desarrollo del sistema se tomaron como base las consultas anteriores tratando de elegir aquellas soluciones de diseño que permitían una mejor optimización de éstas.

Requisitos hardware

Servidor para rack	Intel Server System R1304GZ4GC
Procesador	2 Procesadores Intel Xeon E5-2620
Memoria	64GB de RAM (4 módulos de 16GB) ampliable
Almacenamiento	2 Discos SAS de 2TB

Tabla 2.1: Requisitos hardware para cada nodo seleccionados por el BIFI.

Requisitos funcionales

Id	Nombre	Descripción
RF-1	Escucha de fuentes	El sistema recopilará datos relacionados con Aragón de las fuentes descritas anteriormente mediante un conjunto de *scrapers*.
RF-2	API con resultados	API REST que permitirá a cualquier usuario consultar los datos recopilados.
RF-3	Control de *scrapers*	Mediante una interfaz web se tendrá un control de los sistemas de escucha de fuentes (crear, editar, detener y reanudar).
RF-4	Control de contenidos	Mediante una interfaz web se podrá bloquear contenidos indeseables o autores que no deseen ser escuchados.

Requisitos no funcionales

Id	Nombre	Descripción
RNF-1	Lenguaje utilizado	Se utilizará Python (2.6/2.7) para la realización de los *scrapers* y la interfaz web.
RNF-2	SGBD	Se utilizará PostgreSQL 9.3 como sistema gestor de bases de datos.
RNF-3	Framework web	Se utilizará algún *framework web* que facilite el desarrollo de la interfaz web.
RNF-4	Hardware	El sistema deberá funcionar correctamente sobre el hardware descrito en la tabla 2.3.

Tabla 2.2: Requisitos funcionales y no funcionales del proyecto.

Capítulo 3

Trabajo realizado

Durante el desarrollo de Aragón Open SocialData se ha realizado el diseño, la implementación y las pruebas de todos los subsistemas enumerados en el apartado 1.3. En este capítulo se detallará cuál ha sido la labor del proyectando y cuáles han sido las decisiones tomadas. El anexo C complementa a este capítulo explicando de manera más técnica y completa el diseño del sistema y la justificación de las decisiones adoptadas.

3.1. Diseño y desarrollo de la solución

Partiendo del análisis de requisitos fue necesario realizar un diseño para un sistema distribuido capaz de resolver todas las necesidades descritas anteriormente. A continuación se explica el diseño adoptado de los módulos más relevantes de la plataforma.

3.1.1. Actores del sistema

Aragón Open SocialData está formado por dos nodos conectados entre sí. Ambos nodos poseen una copia idéntica de la base de datos utilizando una arquitectura maestro-esclavo. Cada nodo tiene una responsabilidad: el nodo que posee la base de datos maestra es el encargado de recopilar toda la información de las fuentes y ofrecer la interfaz web de gestión y monitorización de los resultados y del uso del servicio. El nodo esclavo es el encargado de atender las peticiones de la API.

La figura 3.1 ilustra a muy alto nivel los principales actores del sistema distribuido anteriormente descrito. Además incluye un nuevo actor: el sistema de monitorización de la API, el cual se explica en detalle en el apartado 3.1.7.

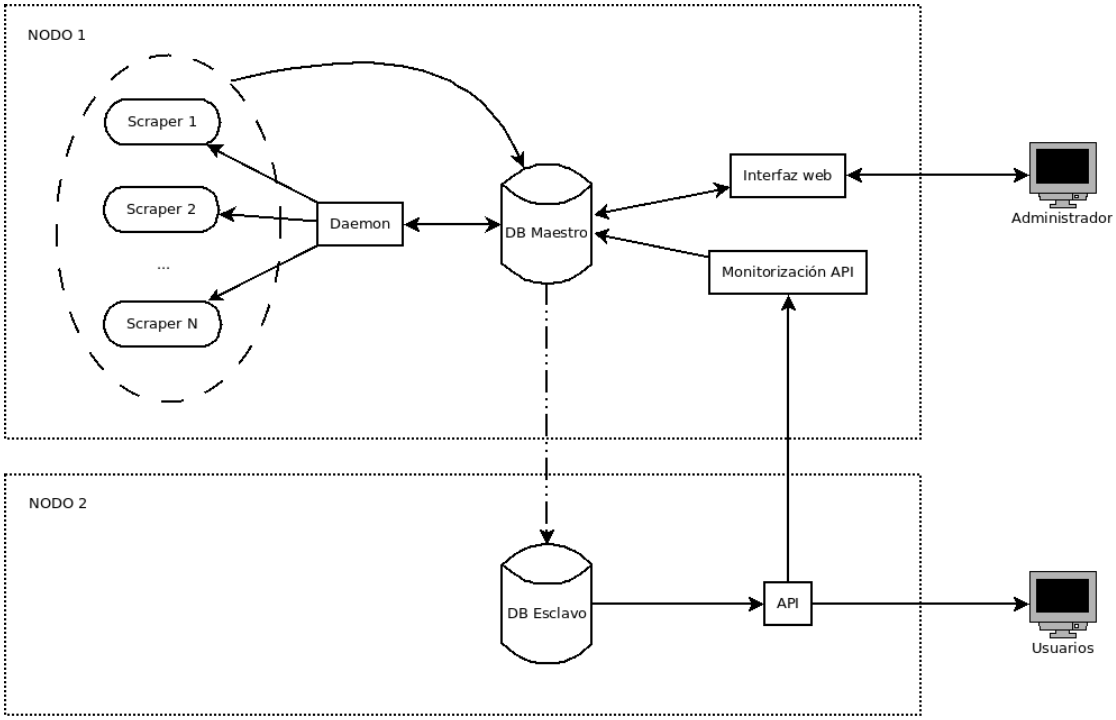


Figura 3.1: Arquitectura del sistema.

3.1.2. Módulos de ayuda en la extración de datos sociales

Se han desarrollado varios módulos que facilitan la extracción de datos de sitios web. Éstos añaden funcionalidades como la capacidad de registrar eventos mediante un sistema de *logger*[1] o monitorizar el número de elementos recogidos durante la ejecución de un *scraper*. El diseño de estos módulos se explica en profundidad en el anexo C.

[1]Sistema de registro y notificación, por ejemplo mediante correo electrónico, de eventos clasificados por severidad.

Request

Se ha diseñado un módulo llamado Request (no confundir con la popular librería Requests de Python) que permite diseñar fácilmente módulos de escucha sociales (*scrapers*) que realicen peticiones HTTP de manera tolerante a los fallos.

Los servicios web son sistemas propensos a errores (servidores caidos, congestión en la red, etc.), por ello, el módulo Request utiliza un algoritmo de *backoff*[2], similar al empleado para controlar el acceso al medio con CSMA/CD, cuyo objetivo es clasificar los errores devueltos por el servidor en distintos niveles de severidad y reintentar aquellas peticiones que, por el tipo de error devuelto, puedan responder tras un tiempo de espera (el servidor libera recursos para atender a nuevas peticiones, la red se descongestiona, etc.). La figura 3.2 refleja el funcionamiento, de manera simplificada, de este módulo.

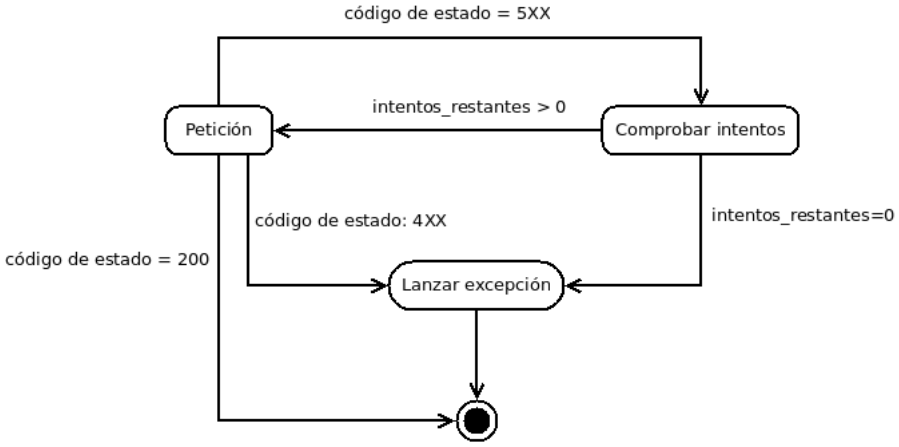


Figura 3.2: Diagrama de estados simplificado del módulo Request.

En función del código de estado HTTP devuelto por el servidor se realiza un tipo de *backoff* u otro. Se distinguen cuatro tipos de errores:

Timeout La respuesta del servidor se ha demorado demasiado en llegar y ha expirado el tiempo de espera establecido. Se reintenta el acceso utilizando un *backoff exponencial*.

Código de estado de baja severidad Códigos de estado como 102 (*Processing*) o 408 (*Request Timeout Error*). Puede que el servidor esté demasiado

[2]Algoritmo que se realimenta de los resultados obtenidos para disminuir su tasa de acceso a un recurso con el fin de encontrar eventualmente una tasa aceptable. En función de la frecuencia con la que reintenta el acceso al recurso se clasifica en *backoff lineal*, *backoff exponencial*, etc.

ocupado y no sea capaz de atender nuestra petición correctamente. Aplicamos un *backoff lineal* esperando a que el servidor pueda atendernos. Si tras un número determinado de intentos el servidor sigue sin responder registramos el error en el *logger* y lanzamos una excepción.

Código de estado de alta severidad Códigos de estado como 403 (*Forbidden*) o 420 (*Method Failure*). El servidor puede estar denegándonos el acceso por haber excedido el límite de peticiones admitidas, caso muy frecuente, o por un fallo interno temporal (500: *Internal Server Error*). Aplicamos un *backoff exponencial* esperando a que el servidor pueda procesar nuestra petición. Si tras un número determinado de intentos el servidor sigue sin responder registramos el error en el *logger* y lanzamos una excepción.

Errores irrecuperables Códigos de estado como 400 (*Bad Request*) o 405 (*Method Not Allowed*) indican que hemos realizado una petición incorrecta y, por tanto, el error recae sobre nosotros y no sobre el servidor. Registramos el error en el *logger* y lanzamos una excepción sin reintentar la petición.

HttpBot

Este módulo provee un simulador de iteracción humana en un navegador.

Algunas de las fuentes escuchadas no proveen una API para obtener resultados y, por tanto, es necesario aplicar otras técnicas de *scraping*. Estas técnicas consisten en analizar el sitio web tal y como lo veríamos en un navegador y extraer información relevante de él. Para ello, la solución más frecuente consiste en analizar el código HTML de la respuesta HTTP y filtrar el contenido de ella en base al DOM[3] del documento HTML devuelto por el servidor.

Sin embargo, existen sitios web que no devuelven los resultados de forma estática mediante HTML. Gracias a una técnica conocida como AJAX [10] (*Asynchronous JavaScript And XML*) el navegador, mediante JavaScript, puede solicitar al servidor más información de manera asíncrona ante un evento del usuario. Cuando el navegador recibe la respuesta del servidor actualiza el contenido del sitio web mostrado al usuario.

Por ejemplo, un caso muy típico donde se utiliza esta técnica es el siguiente:

[3]El *Document Object Model* o DOM (Modelo de objetos del documento) es una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos y una interfaz estándar para acceder a ellos y manipularlos.

1. Al realizar desde el navegador una petición a un servicio web éste devuelve los 20 primeros resultados.

2. Si hacemos *scroll*, es decir, si deslizamos la barra de desplazamiento hacia abajo para mostrar más contenido, un *script* de JavaScript detecta este evento y realiza una petición al servidor pidiéndole los siguiente 20 resultados mediante AJAX.

3. Tras recibir la respuesta se actualizan los resultados mostrados en el navegador añadiendo los 20 siguientes. JavaScript permite manipular el DOM de la página web mostrada en el navegador actualizándolo de forma dinámica.

4. Si volvemos a realizar *scroll* obtendremos los 20 resultados siguientes, y así sucesivamente.

La figura 3.3 ilustra este intercambio de mensajes en las cabeceras HTTP.

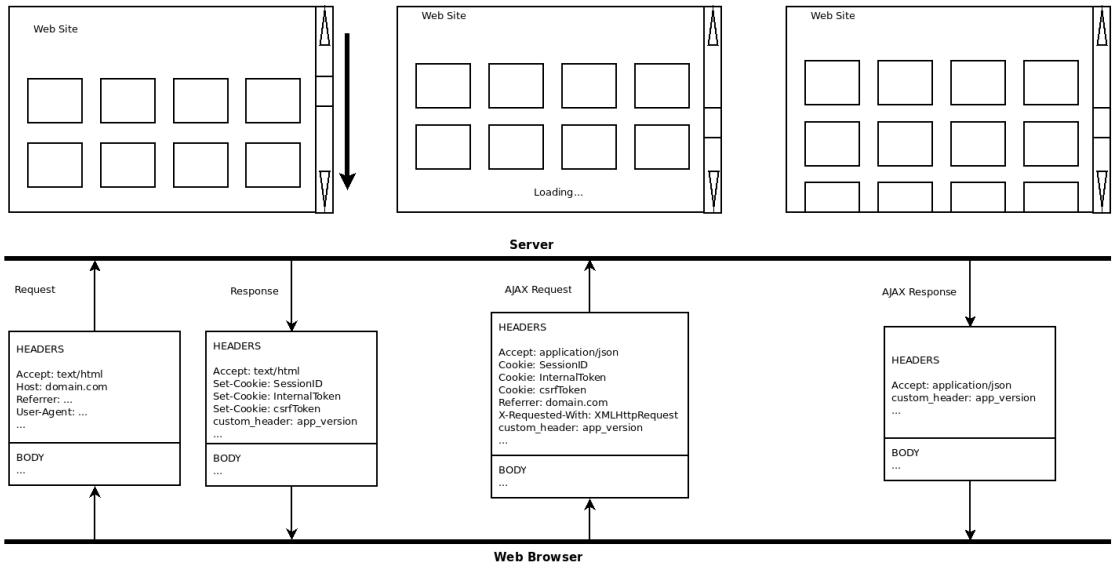


Figura 3.3: Flujo de mensajes intercambiados en las cabeceras HTTP al realizar peticiones asíncronas utilizando AJAX.

Además, como medida de seguridad, es común incluir en el sitio web *tokens* o identificadores temporales generados por el servidor: *tokens CSRF*[4], identificadores

[4] *Cross Site Request Forgery*, o falsificación de petición en sitios cruzados en castellano, es un tipo de *exploit*[5]malicioso mediante el cual el atacante intenta hacer que los usuarios de un sitio web, sin saberlo, envíen datos, como la *cookie* de sesión del usuario, la dirección IP de la víctima, etc. a un sitio web distinto al que se está accediendo.

de sesión, sitio web previo que nos ha traido aquí (*referrer*), etc. Esta información se transmite utilizando *cookies* y otros mecanismos encapsulados dentro de las cabeceras HTTP de la respuesta del servidor [11].

Al realizar una nueva petición sobre el mismo dominio, un navegador web actual incluye estas *cookies* en las cabeceras HTTP de las peticiones siguientes realizadas al sitio web. Del mismo modo, al realizar peticiones AJAX mediante JavaScript es posible modificar el contenido de las cabeceras HTTP enviadas al servidor.

Para emular este comportamiento y que el servidor remoto atienda a las peticiones correctamente es necesario imitar el flujo de mensajes intercambiados entre el navegador y el servidor web. Para ello se deben recoger los *tokens* e identificadores almacenados en las cabeceras HTTP de la primera respuesta del servidor y, a partir de esta información, se deben enviar las peticiones posteriores con las cabeceras HTTP adecuadas.

Para lograr este objetivo, en las fuentes que utilizan AJAX, se ha estudiado el flujo de peticiones y respuestas HTTP entre el servidor y el navegador web (empleando las herramientas para desarrolladores del navegador Google Chrome, en concreto la herramienta *Network*) disparadas mediante eventos del usuario (*scroll*, pulsación de un botón, etc.) y, posteriormente, se ha utilizado este módulo para simular este flujo de mensajes de acuerdo al comportamiento analizado.

3.1.3. Scrapers

Han sido desarrollados *scrapers* para todas las fuentes mencionadas anteriormente. Se han empleado distintas técnicas en función de las facilidades que ofrece cada fuente para la extracción de contenido. La tabla 3.1 resume la estrategia de *scraping* seguida para cada una de ellas.

Para las fuentes que no disponen de API y devuelven los resultados en HTML crudo se utiliza un *parser* HTML y el módulo HttpBot para simular las peticiones HTTP estándar y las peticiones AJAX que realizaría un navegador.

Un *parser* HTML es un algoritmo que analiza sintacticamente un documento HTML, convierte el texto de entrada en otras estructuras (comúnmente árboles) que son más útiles para el posterior análisis y capturan la jerarquía implícita del

[5]Fragmento de software, fragmento de datos o secuencia de comandos y/o acciones, utilizada con el fin de aprovechar una vulnerabilidad de seguridad de un sistema de información para conseguir un comportamiento no deseado del mismo.

Estrategia	Fuentes
API REST sin autenticación	YouTube, Wikipedia
API REST con OAuth	Facebook, Twitter, Instagram, Vimeo, Flickr, Foursquare, Google+
API REST con Basic OAuth	GitHub
Sindicación RSS	Blogger, Wordpress, Blogia
Parser HTML y uso de HttpBot	Pinterest

Tabla 3.1: Estrategia de *scraping* seguida para cada fuente.

documento.

OAuth es un protocolo de autenticación que permite a un usuario dar acceso a terceros a parte de sus recursos en un servico sin que estos conozcan las credenciales del usuario. El funcionamiento del protocolo de autenticación OAuth se explica en el anexo A. El sistema de autenticación *Basic OAuth*, utilizado con GitHub, consiste en incluir el *Access Token* de OAuth en la cabecera HTTP *x-oauth-basic* sin necesidad de realizar todo el proceso de certificación de credenciales previo característico de este protocolo de autorización.

Además, para aquellas fuentes que ofrecen resultados geolocalizados pero no indican explícitamente las coordenadas geográficas donde el contenido fue generado y, en su lugar, devuelven información como el municipio más cercano, una dirección física o un código postal se consulta la API de Google Maps con esta información y se extraen de ella las coordenadas donde el dato fue generado.

También se incluye un módulo auxiliar de detección de idiomas de corpus documentales utilizando la libería LangId de Python para filtrar contenido en aquellas fuentes en las que no es posible cribar los resultados de otra manera. En el anexo C se explica el funcionamiento de este módulo.

3.1.4. Daemon

El módulo Daemon convive con los distintos módulos de escucha (*scrapers*) y es el encargado de ejecutarlos periodicamente, detenerlos, reiniciarlos y monitorizar su ejecución.

Este módulo actua como *scheduler* (planificador) y permite ejecutar tareas

periódicas. Para maximizar su flexibilidad ha sido diseñado de forma que admita como tarea a ejecutar cualquier tipo de comando del sistema y pueda ser un reemplazo de la herramienta *cron*[6] de Unix para este tipo de trabajos.

Las ventajas que ofrece este módulo frente a *cron* son las siguientes:

- Ofrece la posibilidad de controlar las tareas tanto por línea de comandos como desde una interfaz web: añadir nuevas tareas, editar las existentes, detener y reanudar las tareas manualmente de forma dinámica, etc.
- Permite monitorizar el tiempo de inicio y finalización de cada tarea, así como el código de salida del comando.
- Ofrece un sistema de visualización gráfica, desde la interfaz web, del estado de la ejecución de las tareas en tiempo real.
- Implementa un sistema de notificación de eventos mediante señales u otros mecanismos.

Durante su diseño se ha puesto especial interés en desacoplar sus componentes, de forma que sea adaptable a otros proyectos similares donde se precise otros sistemas de registro de tareas o se utilicen otros sistemas de notificación, por ejemplo, mediante RPC[7].

La figura 3.4 muestra un diagrama de clases simplificado de la estructura del Daemon y de sus módulos auxiliares.

Además, los módulos auxiliares de *scraping* permiten desarrollar fácilmente *scrapers* como tareas interoperables con el Daemon, tal y como se explica en el anexo C.

3.1.5. Interfaz por línea de comandos

Se ha desarrollado una interfaz por línea de comandos para facilitar la administración de la plataforma. Esta interfaz permite ejecutar el *daemon*, detenerlo,

[6] En el sistema operativo Unix, *cron* es un administrador regular de procesos en segundo plano (*daemons*) que ejecuta procesos o scripts a intervalos regulares (por ejemplo, cada minuto, día, semana o mes).

[7] *Remote Procedure Call* (llamada a procedimiento remoto): protocolo de comunicación de alto nivel que permite ejecutar procedimientos y funciones de una máquina remota.

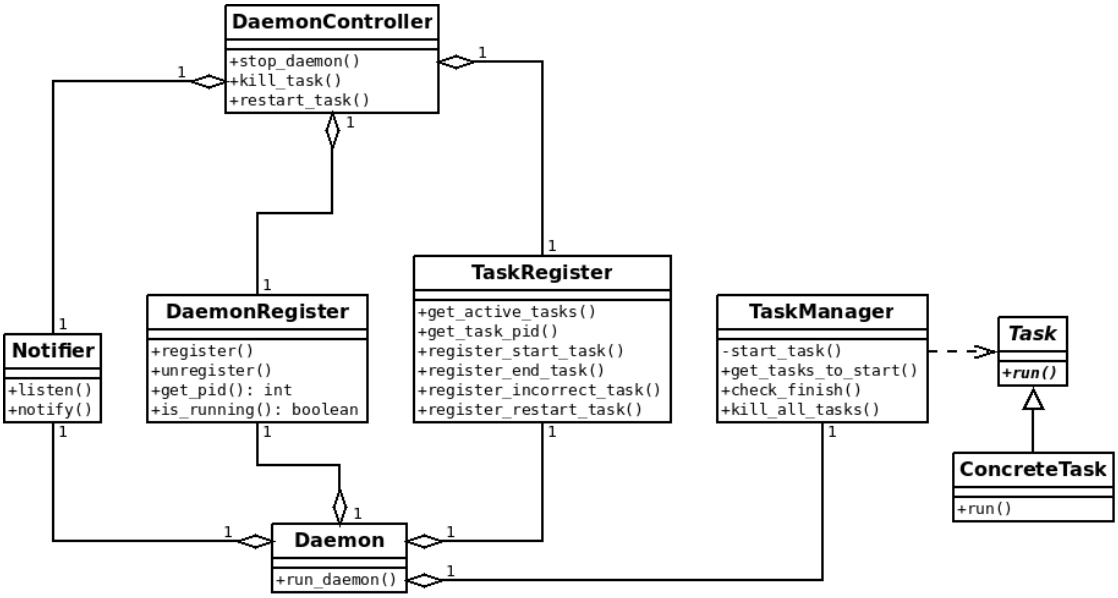


Figura 3.4: Diagrama de clases del módulo Daemon.

detener o reanudar una tarea y realizar tareas de mantenimiento relacionadas con la interfaz web: copia de ficheros estáticos al directorio correspondiente, sincronización de los campos de la base de datos, creación del usuario administrador, ejecución del servidor web de desarrollo, etc.

3.1.6. Interfaz web

De cara a que la DGA sea capaz de conocer la evolución del funcionamiento de la plataforma se ha implementado una interfaz web utilizando el *framework web* Django. Esta interfaz permite gestionar el funcionamiento de los *scrapers*, visualizar la información que éstos recogen y monitorizar el uso de la API.

El acceso a esta interfaz está reservado a los administradores de la plataforma y a usuarios a los que, expresamente, se les autorice el acceso. Por ello se han creado dos roles de acceso: *usuarios administradores* y *usuarios invitados*. Los usuarios administradores tienen control completo de toda la plataforma mientras que los usuarios invitados sólo tienen acceso en modo *"sólo lectura"*, es decir, se les permite visualizar toda la información pero no modificarla.

Las funcionalidades soportadas por la interfaz web se dividen en tres secciones: control de fuentes de escucha, control de contenidos y balance del sistema.

El control de fuentes permite interactuar con el *daemon* y los *scrapers* del siguiente modo:

- Añadiendo nuevas fuentes de escucha.
- Modificando las fuentes de escucha existentes (cambiando el comando de ejecución, la periodicidad con la que se ejecutan, etc.).
- Deteniendo o reanudando manualmente los *scrapers* de forma dinámica.
- Visualizando de forma gráfica el estado de la ejecución de los *scrapers*: duración, inicio y fin de la ejecución, código de salida, etc. (figura 3.5).

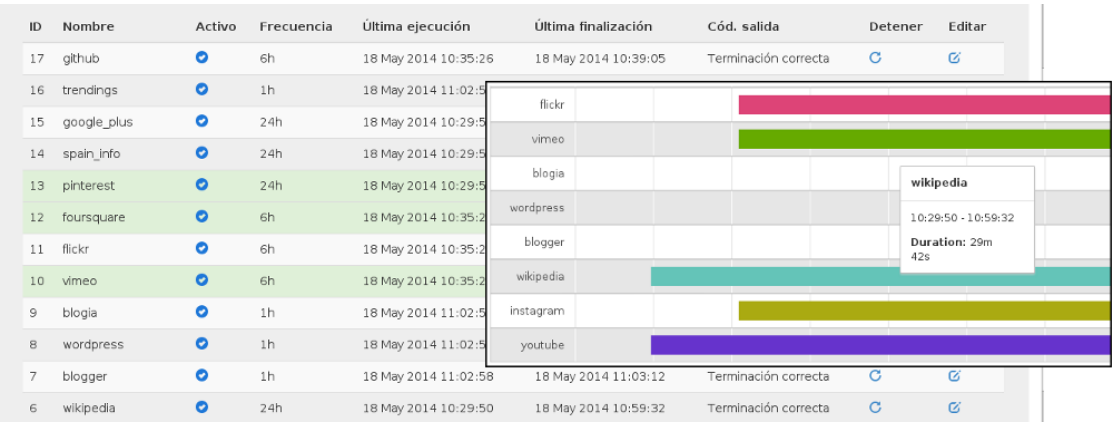


Figura 3.5: Visualización gráfica del estado de ejecución de los *scrapers*.

El control de contenidos (figura 3.6) permite censurar, eliminar y bloquear contenido indeseable o la información publicada en las fuentes por usuarios que, voluntariamente, no deseen ser escuchados. Se ha considerado necesario establecer un sistema de control de contenidos para sanear los resultados de la API evitando servir contenido ilegal o inadecuado. Si se considera conveniente, el administrador de la plataforma dará una alerta a las autoridades policiales antes de eliminar este tipo de contenido de la plataforma. De acuerdo con la LOPD (Ley Orgánica de Protección de Datos) el contenido censurado o la información publicada por aquellos usuarios que no deseen ser escuchados debe eliminarse completamente y de manera irrecuperable de Aragón Open SocialData.

Por último, la sección de balance permite visualizar de forma gráfica la evolución en tiempo real tanto del sistema de escucha como de la API (figura 3.7). Los tipos de visualización implementados son los siguientes:

Figura 3.6: Pantalla de control de contenidos.

- Visualización gráfica de los datos recopilados por fuente en forma de gráficos estadísticos y sobre un mapa de calor de la Comunidad Autónoma de Aragón.
- Visualización de las tendencias que realimentan el sistema de escucha. Además se ofrece la posibilidad a los *usuarios administradores* de añadir tendencias manualmente o eliminar algunas de ellas.
- Visualización en forma de histograma de la evolución de una palabra clave en los resultados recopilados de las fuentes. Se puede conocer cómo ha ido evolucionando a lo largo del tiempo una tendencia en las redes sociales: cuándo empezó a ser popular, cuándo dejó de serlo, qué repercursión tuvo, etc.
- Monitorización del uso de la API. Permite comprobar de forma gráfica cómo los usuarios hacen uso de la API: qué consultas son las más populares, cuál es el momento en el que se realizan más peticiones, cuál es la tasa de error del sistema, etc.

Además, el sistema de visualización permite filtrar los resultados por fecha y obtener, de este modo, los resultados históricos del sistema.

3.1.7. Monitorización de la API

Una necesidad de los administradores de Aragón Open SocialData es conocer el uso histórico de la API: ¿qué búsquedas son las más frecuentes? ¿Cuándo se realizan? ¿Qué búsquedas producen más errores? Para monitorizar el comportamiento

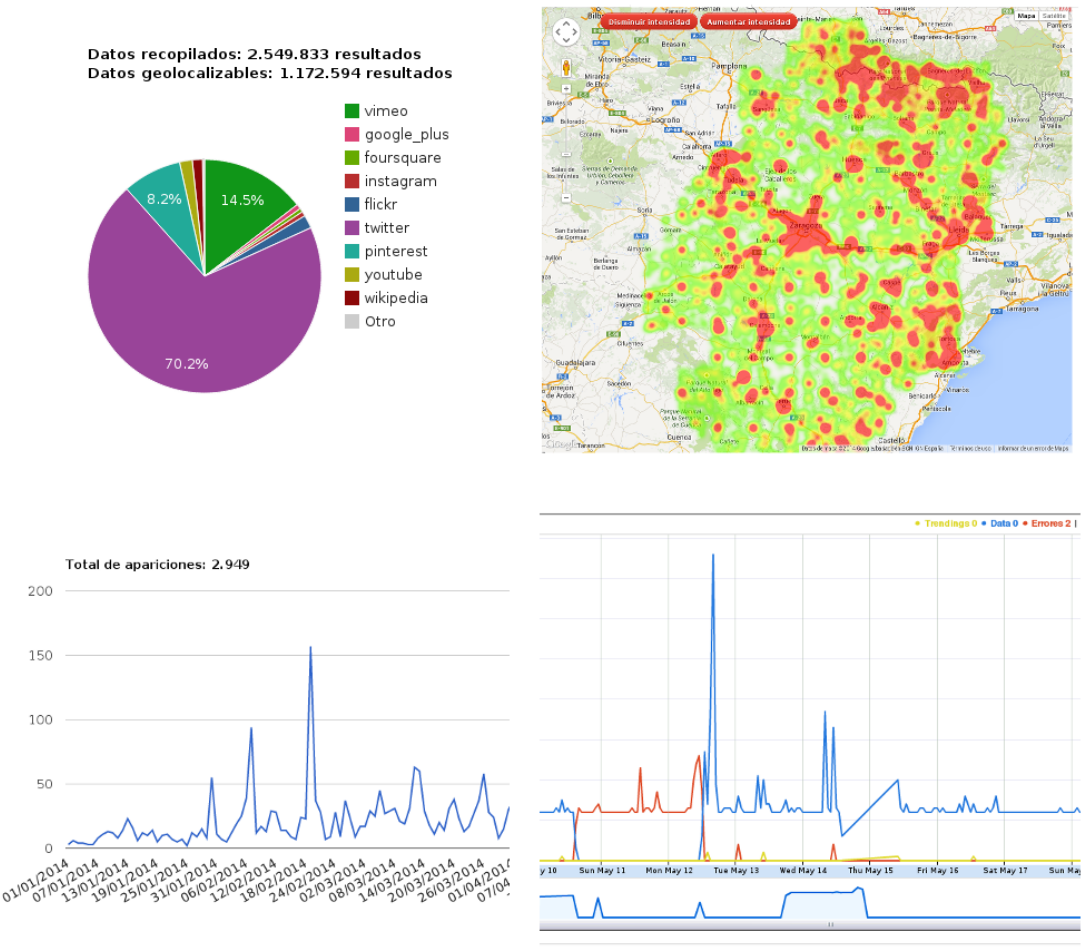


Figura 3.7: Algunos de los sistemas de visualización implementados. Arriba a la izquierda datos recopilados por fuente. Arriba a la derecha mapa de calor del contenido por fuente. Abajo a la izquierda evolución de palabras clave en los datos recogidos de las fuentes. Abajo a la derecha monitorización del uso de la API.

de los usuarios de la API se ha diseñado un sistema de tareas asíncronas utilizando el protocolo AMQP (*Advanced Message Queuing Protocol*, protocolo de colas de mensajes avanzadas en castellano). Este sistema utiliza una cola de mensajes distribuida como mecanismo de comunicación entre los dos nodos del sistema.

El sistema está compuesto por un servicio responsable de la cola de tareas al que se le conoce como *Broker*, uno o varios clientes que encargan y encolan tareas y uno o varios procesos encargados de ejecutar las tareas encoladas (*Workers*). Cuando se produce una petición en la API, el Nodo 2 encola una tarea en el *Broker* con

información de la cabecera HTTP de la petición del usuario. Cuando un proceso *Worker* del Nodo 1 esté desocupado, recogerá la primera tarea introducida en la cola y registrará en la base de datos los detalles de la petición.

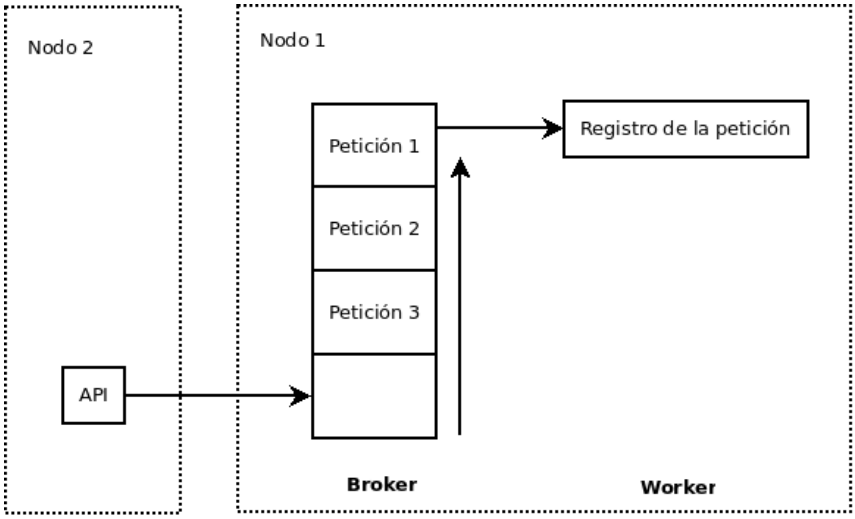


Figura 3.8: Monitorización de la API.

Este diseño permite escalar muy facilmente el sistema de monitorización. Si, conforme avance el tiempo, el servicio tiene mucho éxito y se produce un elevado número de peticiones simultáneas a la API, escalar el sistema de monitorización es tan fácil como agregar más procesos *Workers* que escuchen del mismo *Broker*. Estos nuevos procesos pueden estar situados en la misma máquina o en nuevas máquinas.

Para facilitar la implementación del módulo de monitorización se ha empleado el *framework* Celery de Python que permite trabajar con este tipo de sistemas de manera muy sencilla, haciendo transparente al desarrollador el uso de un determinado servicio como *Broker* o el uso de más o menos *Workers*.

3.2. Tests

Para comprobar el correcto funcionamiento de las distintas partes del sistema se ha seguido la metodología *Test-Driven Development* (TDD), desarrollo guiado por pruebas en castellano. Esta metodología forma parte de un conjunto de metodologías de desarrollo ágil. Su uso conlleva la realición de los tests antes de desarrollar cada una de las funcionalidades que estos deberán validar.

En total, se han generado 88 tests unitarios y de integración. Todos ellos se verifican de forma automática mediante la herramienta *nose* de Python. Esta herramienta se encarga de buscar los tests del proyecto, ejecutarlos y comprobar su correcto funcionamiento. Si se produce una excepción durante la ejecución de un test o si éste falla se notifica al usuario incluyendo información detallada del error.

Durante el desarrollo de la plataforma se ha seguido una metodología muy estricta evaluando continuamente todos los tests tras cualquier cambio en el código fuente para asegurar el correcto funcionamiento del sistema. *Nose* nos facilita esta tarea. Además, se ha creado un sistema de integración continua, rudimentario frente a otras herramientas más avanzadas pero funcional para este proyecto: un sencillo script que continuamente monitoriza los cambios en el sistema de control de versiones y ejecuta los tests alertando al desarrollador ante cualquier error.

Dadas las circunstancias singulares de este proyecto donde una gran parte de módulos están relacionados con servicios de *I/O* (entrada/salida) (*scrapers* realizando peticiones en la red, escrituras y consultas a la base de datos...) se ha emulado el comportamiento de la mayoría de estos módulos durante los tests mediante objetos simulados (*Mock objects*).

Sin embargo, los objetos más externos del sistema, los encargados de realizar realmente las tareas de I/O, no pueden ser simulados. Además, los módulos de *scraping* dependen de las API o del DOM de los servicios web de los que extraen la información. Si un servicio cambia el nombre de un atributo o deja de responder a una determinada petición, el *scraper* correspondiente a esa fuente fallará. Por ello, se ha creado un sistema robusto de monitorización y registro de errores (*logs*). Dicho sistema monitoriza y alerta detalladamente al desarrollador o desarrolladores del error encontrado por correo electrónico. Gracias a ello el responsable de mantenimiento de la aplicación será capaz de detectar, modificar y reparar el sistema fácilmente cuando uno de estos módulos falle.

3.3. Despliegue y puesta en producción

Finalmente fue necesario desplegar la plataforma en los servidores cedidos por el BIFI descritos en la sección 2.2.

Se decidió utilizar el servidor HTTP de Apache tanto para dar servicio a la API en el nodo 2 como para servir el contenido estático y dinámico de la interfaz web en el nodo 1. Este servidor cuenta con varios módulos nativos y de terceros que

permiten añadirle funcionalidades, como el módulo `mod_wsgi`, que permite lanzar un intérprete de Python con permisos determinados al recibir una petición HTTP o el módulo `mod_evasive`, que permite prevenir ataques de denegación de servicio (DDoS: *Denial-of-service attack*).

En cuanto al sistema de monitorización de la API, para minimizar el uso de red se ha instalado y configurado como *Broker* un servidor de Redis [12], el cual es un sistema muy ligero de almacenamiento en memoria de objetos clave-valor. Redis ha sido instalado en el nodo 1 y en ese mismo nodo se ejecuta un *Worker* de Celery encargado de registrar los accesos a la API. Los procesos *Worker* de Celery están consultando continuamente la cola del *Broker*. Si éste, es decir, si Redis se ejecutase en el otro nodo (nodo 2), el *Worker* del nodo 1 estaría constantemente comunicándose con la otra máquina para consultar el estado de la cola de tareas, haciendo un uso innecesario del enlace que conecta ambos nodos.

En el anexo G se detallan todos los pasos seguidos para desplegar la plataforma en los dos nodos.

Capítulo 4

Gestión del proyecto

En este capítulo se presentan las metodologías que han servido de base para la gestión de este proyecto y la planficiación realizada a partir de ellas, gracias a la cual ha sido posible entregarlo en el plazo previsto.

4.1. Metodología

La existencia de plazos de entrega cortos, determinados tras el acuerdo inicial entre el BIFI y la DGA, y el riesgo de que el trabajo realizado no sea el esperado por el cliente hacen de este proyecto un ejemplo perfecto para el uso de metodologías ágiles.

Sin embargo, por las restricciones temporales y de recursos que impone el desarrollo de un proyecto de fin de carrera, no se ha seguido ninguna metodología de una manera estricta y completa, en su lugar, se han adoptado varios aspectos de metodologías ágiles como Scrum, más cerca del Scrumban [13] (fusión de las metodologías Scrum y Kanban) que del Scrum estricto y tradicional, y TDD (*Test-Driven Development*, desarrollo guiado por pruebas).

Scrum es una metodología ágil para la gestión de proyectos que permite validar el progreso de un proyecto mediante una serie de iteraciones que reciben el nombre de *sprints*. Cada *sprint* suele tener una duracción entre 2 y 4 semanas.

En este proyecto, debido a las restricciones de tiempo, se han realizado 4 iteraciones, completando uno de los hitos del proyecto en cada una de ellas. Tras

cada *sprint* el objetivo de la siguiente iteración ha sido completar otro de los hitos pendientes. La tabla 4.1 presenta de forma resumida la duración y objetivos alcanzados en cada *sprint*.

Sprints de Scrum.

Sprint	Duración	Objetivo
1	3 semanas	Desarrollo de *scrapers* y del *daemon* que los controla.
2	2 semanas	Desarrollo de la interfaz web.
3	3 semanas	Desarrollo de la API y puesta en producción.
4	4 semanas	Elaboración de la memoria y la documentación del proyecto.

Tabla 4.1: Duración e hitos alcanzados en cada iteración de Scrum.

Además, cada vez que una iteración finalizaba se organizaba una reunión con el *Product Owner* (cliente) para validar que el trabajo realizado hasta la fecha era realmente lo que él necesitaba. Estas reuniones han servido como mecanismo de realimentación del proyecto. Una de las tareas de cada *sprint* ha sido mejorar la aplicación existente hasta el momento a partir de las necesidades descritas por el cliente en cada reunión.

Sin embargo, debido a las tareas multidiscipilnares en las que me he visto involucrado en este proyecto (desarrollo, soporte técnico, administración de sistemas, etc.) ha sido muy frecuente la entrada de tareas imprevistas y de alta prioridad durante el desarrollo de un *sprint*. Por ello se ha adoptado un sistema de tarjetas con prioridades, similar al utilizado por Kanban, utilizando la herramienta online *Trello*.

No se ha trabajado con dos tableros separados, uno para Scrum y otro para Kanban, como se recomienda en aquellos proyectos que adopten el *Scrumban*. En su lugar se ha utilizado un tablero común flexibilizando los *sprints* y dejando un tiempo extra de margen en cada iteración para este tipo de tareas imprevistas.

Además, como se ha mencionado en el apartado 3.2, se ha realizado un desarrollo guiado por pruebas (TDD) en la mayoría de módulos implementados. Este mecanismo se ha empleado conjuntamente con un sistema, muy sencillo pero funcional, de integración continua. La aplicación de estas técnicas me ha permitido

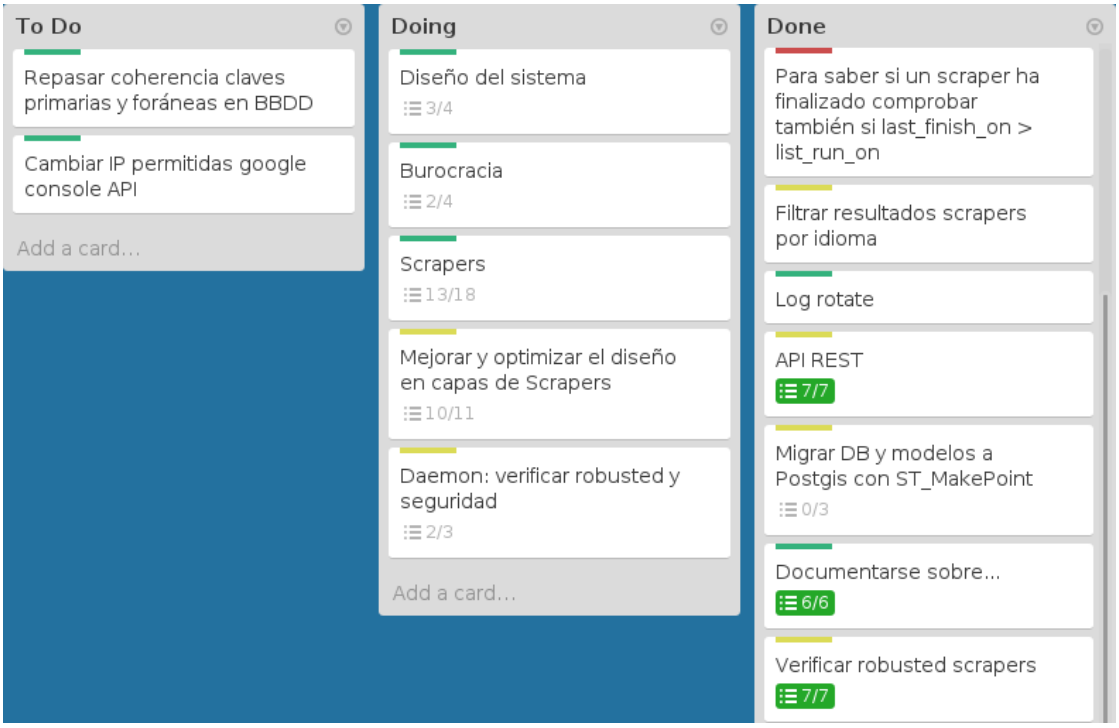


Figura 4.1: Kanban de tareas con prioridades de Trello.

anticiparme a los errores e indentificarlos y solventarlos rápidamente cuando estos aparecían.

El resultado ha sido bastante satisfactorio. Se han podido cumplir perfectamente los plazos previstos y el producto desarrollado cumple con la calidad y los requisistos esperados.

4.2. Planificación y dedicación

La figura 4.2 muestra las principales fechas límite que se han seguido a lo largo del proyecto. Cada una de ellas corresponde con una iteración de Scrum y han sido fijadas de acuerdo a las fechas de entrega y reuniones con el cliente. El último hito de la figura (Formación y difusión) es responsabilidad de la Diputación General de Aragón y no del proyectando.

Para cada objetivo se ha realizado una estimación del esfuerzo necesario para alcanzarlo en número de horas y, tras la finalización de cada *sprint*, se ha compara-

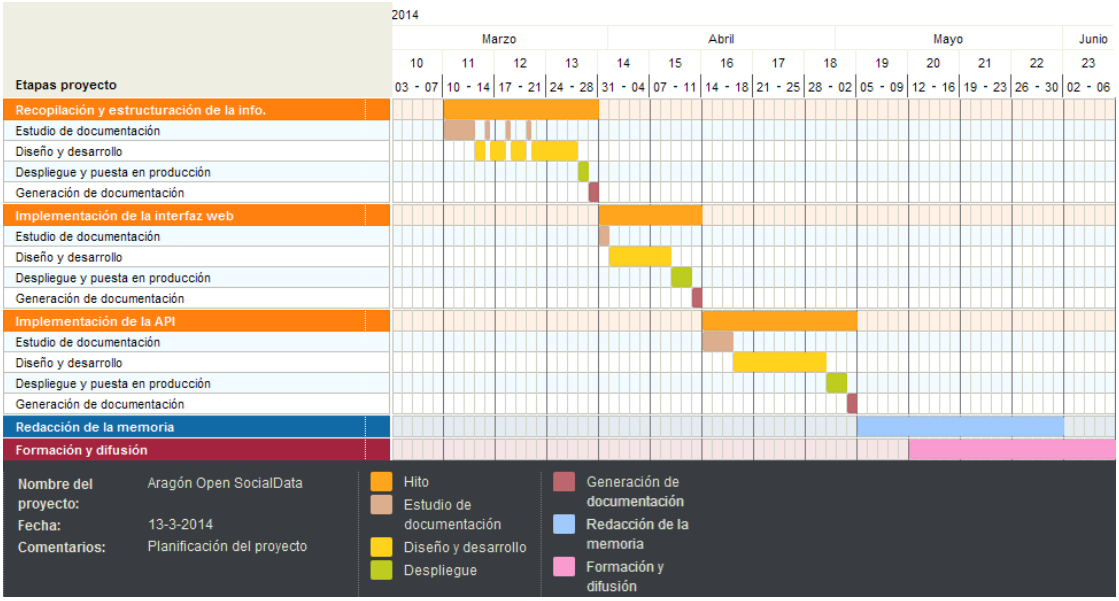


Figura 4.2: Diagrama de Gantt.

do el número de horas reales dedicadas con el número de horas estimadas. De esta forma ha sido posible comprobar si las estimaciones realizadas han sido correctas o no y utilizar este resultado como realimentación (*feedback*) para ajustar las estimaciones del siguiente *sprint*. En el anexo B se detalla la planificación realizada, desglosada por objetivos, y el número de horas finales dedicadas en cada *sprint*.

El número total de horas invertidas en el proyecto Aragón Open SocialData desde la entrada del proyectando a él hasta la fecha de entrega de la memoria es de **493 horas**.

Capítulo 5

Conclusiones y trabajo futuro

En este último capítulo se analizan los resultados obtenidos, se indican posibles líneas de trabajo futuras y se expone una valoración personal del PFC desarrollado.

5.1. Líneas futuras

Al tratarse este proyecto de un servicio Open Data abierto al público, cualquier persona que lo desee puede reutilizar la información recopilada y almacenada en la plataforma para desarrollar sus propias aplicaciones, informes, estudios estadísticos, etc.

No obstante, durante el desarrollo de este PFC han ido surgiendo nuevas ideas en las que poder seguir trabajando para mejorar y ampliar las funcionalidades del sistema de *web scraping social* realizado. Algunas de estas funcionalidades son las siguientes:

- Ampliar el número de fuentes sociales escuchadas incluyendo sitios web de prensa (El Periódico de Aragón, El Heraldo de Aragón...), rutas (OpenPaths, Wikiloc, Runastic...) y servicios de alquiler de hoteles y otros inmuebles (Tripadvisor, Airbnb...).

- Mejorar la eficacia del sistema de análisis de tendencias excluyendo aquellas que no tienen relación aparente con Aragón o que son tendencias a nivel nacional y no sólo de la comunidad autónoma.

- Incluir más formatos de respuesta en la API como, por ejemplo, XML, RDF, CSV...

Para dar a conocer la existencia de esta plataforma y tratar de fomentar su uso se realizarán varios cursos de formación. Uno de ellos será un curso de 4 horas orientado a responsables de la administración de la infraestructura y del servicio Aragón Open SocialData, a los que se otorgarán acreditaciones para el uso del sistema. El otro curso irá destinado a usuarios de la plataforma.

Además, con el mismo propósito, se realizará un *hackathon*[1] los días 26 y 27 de Septiembre de 2014 en Jaca. Con este evento se pretende dar a conocer el proyecto, conseguir que el público se familiarice con la plataforma y lograr desarrollar alguna aplicación que haga uso de toda la información recopilada hasta ese momento y sirva como ejemplo para futuros proyectos.

5.2. Conclusiones

El proyecto ha sido desarrollado a tiempo cumpliendo los plazos previstos. El cliente, es decir, la DGA, se encuentra satisfecho con los resultados obtenidos, siendo éstos los esperados.

Los objetivos planteados al inicio del proyecto se han alcazado con éxito. La plataforma Aragón Open SocialData se encuentra completamente operativa y accesible para todo aquel que desee utilizarla. Además se han añadido varias mejoras relacionadas con la visualización de la información recopilada y el uso de la API: resultados estadísticos y mapas de calor por fuente, gráficos de evolución de palabras clave por fecha, gráficos de evolución histórica de las consultas realizadas por los usuarios a la API, etc.

Se ha intentado ser proactivo en todo momento creando una solución software en forma de *framework*, de modo que pueda ser reutilizable en futuros proyectos. De hecho, en el BIFI se está haciendo uso de algunas funcionalidades de dicho *framework* aplicadas al proyecto SEPS.

[1]Encuentro de entusiastas de la tecnología cuyo objetivo es el desarrollo colaborativo de software y/o hardware durante varios días como si de un maratón se tratase.

5.3. Valoración personal del trabajo realizado

Todo este trabajo ha supuesto la culminación de varios años de formación universitaria, poniendo de manifiesto todo lo aprendido en un proyecto real para un cliente de gran envergadura como es la Diputación General de Aragón (DGA). Se espera que el producto creado sea utilizado por cientos de usuarios y permita el desarrollo de varias aplicaciones interesantes que utilicen como base la plataforma desarrollada en este PFC.

Gracias a trabajar en este proyecto he descubierto la gran importancia que posee la información social generada por los usuarios en Internet y todas las posibilidades que ofrece su estudio.

También me ha permitido estudiar y ampliar mi formación en temas como las bases de datos geoespaciales y los GIS, los protocolos de autorización de acceso en Internet como OAuth, las colas de tareas asíncronas, la complejidad que entrañan los sistemas distribuidos y la necesidad de aplicar metodologías de desarrollo de software durante todo el ciclo de vida de un producto para poder lograr los objetivos planteados y afrontar los problemas que puedan ir apareciendo.

Además, este proyecto ha supuesto mi primer contacto con el mundo laboral. He trabajado durante tres meses en el Instituto de Biocomputación y Física de Sistemas Complejos (BIFI) junto con personas excelentes de las que he aprendido grandes cosas y que me han guiado y ayudado en los momentos en los que he necesitado su apoyo.

En definitiva el desarrollo de este PFC ha sido muy gratificante y de él me llevo un muy buen sabor de boca y una experiencia que espero poder aplicar en futuros proyectos de mi vida laboral.

Anexos

Apéndice A

Tecnologías utilizadas

A lo largo de este anexo se enumerarán y explicarán las tecnologías utilizadas durante el proyecto, justificando por qué se han elegido éstas y no otras y analizando las ventajas y desventajas de cada una de ellas.

A.1. Python

Python es el lenguaje de programación principal empleado para implementar el proyecto. Se ha utilizado la versión 2.7 del mismo, versión algo antigua pero estable y compatible con multitud de paquetes de terceros, especialmente con los del popular repositorio conocido como PyPI (*Python Package Index*).

Python es un lenguaje de programación de alto nivel e interpretado cuya filosofía hace incapié en escribir código con una sintaxis muy limpia que facilite su legibilidad[1]: "el código se lee más veces de las que se escribe". Además, la sintaxis de Python obliga al desarrollador a escribir código siguiendo unas reglas de estilo comunes: sangrado, saltos de línea, ausencia de llaves y *puntos y coma* finales que dificulten la lectura del código fuente, etc. Gracias a esto, una persona que domine el lenguaje puede entender de un primer vistazo el código escrito por otros fácilmente.

Las fuentes de escucha, desde las que se extrae la información que nutre esta

[1]La filosofía de Python, conocida como el Zen de Python, se puede encontrar en el PEP (*Python Enhancement Proposal*) número 20: `http://legacy.python.org/dev/peps/pep-0020`

plataforma, poseen API y sitios web que pueden sufrir modificaciones en cualquier momento y de forma imprevista. Cuando uno de estos cambios suceda, será necesario adaptar los módulos de escucha (*scrapers*) a las modificaciones realizadas. La filosofía de Python encaja en este proyecto: será necesario leer y entender dentro de un tiempo indefinido el código desarrollado para realizar las tareas de mantenimiento, y Python, por su sintaxis, facilita este trabajo.

Además, Python provee un amplio repertorio de librerías software orientadas a interactuar con protocolos de Red de alto nivel como HTTP, SMTP, OAuth, etc. y posee varios *frameworks* que pueden ser utilizados como base para desarrollar nuevas aplicaciones software.

En este proyecto se han utilizado dos *frameworks*: Django y Celery. Django [14] es un *framework web*, publicado en el año 2005, que facilita el desarrollo de aplicaciones web poniendo especial énfasis en la extensibilidad de sus componentes, en el desarrollo ágil y en cumplir el principio DRY[2] [15] (*Don't Repeat Yourself*). Gracias a esto existen un gran número de extensiones, aplicaciones y *middlewares*[3] creados por la comunidad que pueden ser reutilizados en cualquier nuevo proyecto software. Celery [16], el otro *framework* utilizado, permite gestionar, de forma sencilla, tareas asíncronas y distribuidas mediante colas de tareas y el protocolo AMQP (*Advanced Message Queuing Protocol*).

A.2. PostgreSQL

PostgreSQL, también conocido popularmente como *postgres*, es una sistema gestor de bases de datos relacionales de código abierto y ampliable con extensiones como PostGIS que, por su diseño, facilita la alta concurrencia mediante un sistema denominado MVCC (*Multi-Version Concurrency Control* , acceso concurrente multiversión en castellano). Este sistema permite que, mientras se consulta una base de datos, cada transacción ve una imagen de los datos (una versión de la base de datos) sin tener en cuenta el estado actual de la información que realmente está escrita en el disco. Esto evita que las transacciones vean contenido

[2]DRY: *Don't Repeat Yourself*, no lo repitas tú mismo. Principio según el cual toda pieza de información nunca debería ser duplicada. La duplicación incrementa la dificultad en los cambios y la mantenibilidad posterior, puede perjudicar a la claridad de la información y generar inconsistencias. Por "pieza de información"podemos entender, en un sentido amplio, datos almacenados en una base de datos, el código fuente de un programa software, información textual o documentación, etc.

[3]Software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos.

inconsistente como el causado por la actualización de otra transacción concurrente en el mismo registro de datos. De esta forma, PostgreSQL proporciona aislamiento transaccional para cada sesión de la base de datos.

Además, PostgreSQL permite gestionar bases de datos distribuidas siguiendo distintos esquemas. El más común, y el utilizado en este proyecto, es el sistema de réplica maestro-esclavo. Una base de datos (maestro), en un nodo, actua como una base de datos estándar mientras que otra (esclavo), en otro nodo, posee una réplica exacta de la información almacenada en la base de datos maestra y sólo admite consultas de lectura.

También, debido al gran número de fuentes que generan información geolocalizada y a que varias consultas de la API necesitan filtrar el contenido en función de su posición geográfica, se ha decidido crear una base de datos espacial que almacena toda la información recopilada por las fuentes utilizando la extensión PostGIS.

PostGIS es una extensión que aporta a PostgreSQL un conjunto de tipos de datos y funciones geoespaciales que facilitan y optimizan las consultas que involucran datos geolocalizados. Para ello, PostgresSQL con esta extensión hace uso de un GIS (*Geographic Information System* o SIG: Sistema de Información Geográfica en castellano). Un GIS es un sistema de hardware, software y procedimientos elaborados para facilitar la obtención, manipulación, análisis, modelado, representación y salida de datos geográficamente referenciados.

Para evaluar si era necesario utilizar PostGIS o no se realizó un pequeño estudio midiendo el tiempo de ejecución que requerían las consultas geoespaciales de la API de Aragón Open SocialData. En el análisis se evaluaron estas consultas utilizando los tipos de datos estándar de PostgreSQL y los tipos y funciones geoespaciales de PostGIS. El estudio se realizó sobre un conjunto de 22.802 datos recogidos de varias fuentes. Más adelante, cuando el sistema se llevó a producción, se volvió a realizar el mismo estudio con un conjunto de datos más grande (2.038.535) para comprobar si, ante un aumento de registros en la base de datos, el comportamiento de ésta se veía afectado.

Los resultados de este estudio se muestran en la tabla A.1 y reflejan que, para ambos casos, el uso de los tipos de datos y funciones geoespaciales de PostGIS ofrecen una mejora en el tiempo de ejecución de las consultas de aproximadamente el 82 % frente a utilizar los tipos de datos nativos de PostgreSQL.

Tipo de dato	**Tiempo de ejecución**
Estudio con 22.802 datos.	
Tipo double de PostgreSQL	203,9704 ms
Tipo GeometryColumn (Point) de Postgis	35,5194 ms
Estudio con 2.038.535 datos.	
Tipo double de PostgreSQL	20.885,2612 ms
Tipo GeometryColumn (Point) de Postgis	3.635,7042 ms

Tabla A.1: Evaluación de los tipos de datos nativos de PostgreSQL frente a los tipos geoespaciales de Postgis.

A.3. API REST

Una API, *Application programming interface* o interfaz de programación de aplicaciones en castellano, es un conjunto de métodos que ofrece una librería para ser utilizados por otro software como una capa de abstracción.

REST son las siglas de *Representational State Transfer* y especifican una arquitectura software cuyo objetivo es realizar llamadas a procedimientos remotos utilizando una interfaz simple mediante el protocolo HTTP [17].

Por lo tanto, una API REST es un mecanismo que permite ejecutar métodos de librerías externas de forma remota mediante el envio de mensajes HTTP, tanto para realizar peticiones como para recibir respuestas. Estas API REST frecuentemente forman parte de los conocidos SaaS (*Software as a Service*), donde una compañia ofrece sus servicios bajo demanda a sus clientes de forma distribuida [18].

Muchas de las redes sociales actuales poseen este tipo de API, gracias a las cuales ofrecen información de su red a terceros con ciertas limitaciones.

La plataforma Aragón Open SocialData también permite a sus usarios interactuar con toda la información recopilada mediante una API REST utilizando el método GET de HTTP. Es un mecanismo fácil de usar, muy estandarizado ya que hace uso del protocolo HTTP y eficiente al devolver los resultados de las peticiones en formatos ligeros como JSON (*JavaScript Object Notation*).

A.4. Web scraping

Web scraping es una técnica utilizada para extraer información de sitios web de manera automática. Un *scraper* es la entidad software encargada de acceder, rastrear y extraer la información relevante de un sitio web de manera automática.

Existen muchas técnicas de *scraping*: acceder a los recursos mediante las API de los sitios web objetivo, analizar las respuestas de las peticiones HTTP utilizando *parsers*, simular el comportamiento de un humano en sitios web complejos que utilicen peticiones asíncronas con AJAX [10] (*Asynchronous JavaScript And XML*), etc.

Python posee varias herramientas que facilitan las tareas de *scraping*: las librerías urllib, urllibr2 y requests que permiten realizar peticiones HTTP de manera sencilla; las librerías BeautifulSoup, lxml y json que permiten *parsear* las respuestas HTTP, filtrar el contenido (por ejemplom utilizando el lenguaje xpath[4] en lxml) y convertirlo en tipos de datos nativos de Python; y herramientas como Selenium, que aunque originalmente fueron diseñadas para otros propósitos, pueden utilizarse, en casos excepcionales, como herramientas de *web scraping*.

Además, Python posee varios *frameworks* de *web crawling* como Scrapy. No obstante, en este proyecto no se hace uso de este *framework*. Un *web crawler* o *araña web* es una herramienta que rastrea un sitio web, o varios de ellos, mediante un sistema de seguimiento de enlaces. Cuando una *araña* visita una página web la indexa y extrae sus hiperenlaces de forma metódica y automatizada. A continuación, visita cada uno de los hiperenlaces extraídos y vuelve a realizar el mismo proceso para cada uno ellos. Este proceso se repite de forma indefinida. Sin embargo, las necesidades de este proyecto son distintas: en la mayoría de las ocasiones sólo es necesario acceder a una página o servicio incluyendo un parámetro concreto en la URI [19] de dicho recurso; por lo tanto, se ha descartado el uso de este *framework* o soluciones similares que no aportan ninguna funcionalidad necesaria al proyecto.

[4]XPath (*XML Path Language*) es un lenguaje que permite construir expresiones que recorren y procesan un documento XML.

A.5. OAuth y OAuth2

OAuth [20] (*Open Authorization*) es un protocolo de autorización estandarizado y abierto que permite que un tercero acceda a los datos de un usuario en un servicio sin necesidad de conocer sus credenciales. Los usuarios no tienen que compartir sus contraseñas directamente con una aplicación ajena al servicio, OAuth actua como *llave* y permite a las aplicaciones acceder a los datos de un usuario y actuar en su nombre siempre que el usuario lo autorice.

La figura A.1 muestra un diagrama de secuencia de UML[5] (*Unified Modeling Language*) simplificado del funcionamiento de este protocolo.

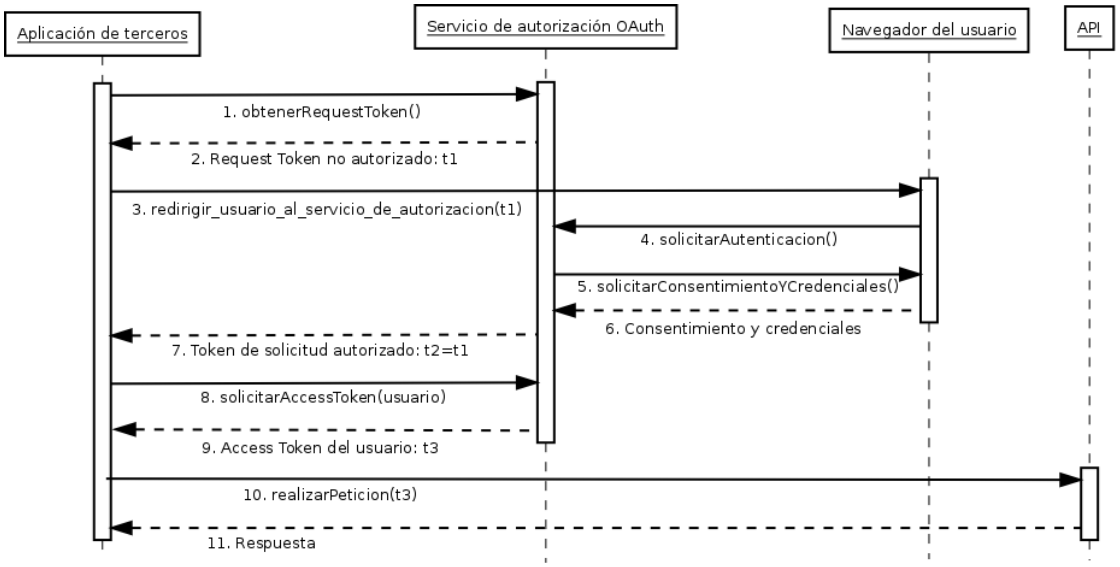


Figura A.1: Diagrama de secuencia del protocolo OAuth.

En algunos casos es posible acceder a servicios que no requieran la autorización de un usuario concreto simplemente obteniendo un *token* de aplicación y realizando las peticiones con él, es decir, omitiendo los pasos del 3 al 9 de la figura A.1.

OAuth2 [21], la segunda versión de este protocolo, incorpora algunas mejoras frente a la primera: un mejor soporte para aplicaciones no basadas en un navegador

[5]UML (*Unified Modeling Language*): lenguaje de modelado de sistemas de software - `http://www.uml.org/`

web, mejor soporte para SSL/TLS[6], *tokens* de acceso de rápida expiración (*short-lived access tokens*), una separación de roles más adecuada, etc.

Servicios como Twitter, Google, Vimeo o GitHub hacen un uso extensivo de este protocolo para autorizar el acceso a sus API.

A.6. AMQP

AMQP (*Advanced Message Queuing Protocol*) es un protocolo abierto de capa de aplicaciones (de nivel 7 según el modelo OSI[7]) orientado a la comunicación de nodos en entornos distribuidos mediante colas de mensajes.

AMQP define los siguientes roles:

- *Broker*: servidor al que los clientes AMQP se conectan para intercambiar mensajes.
- Cliente: entidad que, mediante la presentación de credenciales como una contraseña, puede ser autorizado (o no) a conectarse a un *broker* y a enviar o recibir mensajes de las colas que éste gestione.
- Colas: entidades que reciben los mensajes. Los clientes pueden escribir y recibir mensajes que se almacenarán en las colas. Para ello existen dos mecanismos: subscribirse a las colas, de manera que el *broker* les notifique y entregue los mensajes publicados en aquellas a las que estén suscritos; o consultar activamente los mensajes de ellas. La cola garantiza que los mensajes son entregados en el mismo orden en que llegaron a la cola. Esta ordenación se conoce habitualmente como FIFO (*First Input, First Output*).
- Mensaje: unidad básica de comunicación entre los clientes. Son publicados en las colas y están compuestos por una cabecera y un cuerpo.

[6]SSL (*Secure Sockets Layer*, capa de conexión segura en castellano) y su sucesor TLS (*Transport Layer Security*, seguridad de la capa de transporte en castellano) son protocolos de cifrado (de la capa de transporte según el modelo OSI) que proporcionan autenticación y privacidad al protocolo de red subyacente.

[7]El modelo OSI (*Open System Interconnection*), que fue creado por la Organización Internacional para la Estandarización (ISO) en el año 1980, es un modelo de red descriptivo y un marco de referencia para la definición de las arquitecturas de red utilizadas por distintos sistemas de comunicaciones. `http://es.wikipedia.org/wiki/Modelo_OSI`

Existen otros protocolos de comunicación de alto nivel como JMS (*Java Message Service*) para el lenguaje Java; MQTT (*Message Queue Telemetry Transport*), un protocolo binario desarrollado por IBM basado en un modelo publicador-suscriptor y STOMP (*Streaming Text Oriented Messaging Protocol*), un protocolo de texto plano similar a HTTP pero más ligero que éste.

Se ha elegido AMQP por ser un protocolo maduro, con una buena documentación y una gran fiabilidad. Es usado en Nova, componente de Open Stack, desarrollado originalmente por la NASA para el proyecto Nebula Cloud Computing [22] (uno de los primeros sistemas de computación "en la nube"), también es usado por Instagram para gestionar las imágenes de los *followers* que cada usuario visualiza [23] e incluso por Google en el proyecto Rocksteady utilizando RabbitMQ como *broker* [24].

Además, existe un *framework* escrito en Python que implementa la mayoría de funcionalidades de este protocolo: Celery, que, además, soporta distintos servicios como *brokers*: RabbitMQ, Redis, MongoDB, Amazon SQS, etc.

En este proyecto se utiliza este protocolo como mecanismo para comunicar los nodos del sistema cuando un usuario realiza una petición a la API. En el momento en que ésta es recibida, un nodo se encarga de procesar la petición y encola una tarea asíncrona en el *broker*. El otro nodo, cuando tenga recursos disponibles, recogerá la tarea de la cola y registrará el acceso del usuario a la API en la base de datos relacional.

A.7. Redis

Como *broker* encargado de gestionar las colas del protocolo AMQP se ha utilizado Redis. Redis es una sistema de almacenamiento en memoria de objetos clave-valor. Es un servicio muy ligero y con una nivel de fiabilidad aceptable para el uso que se le da en este proyecto.

Otros servicios analizados para ser usados como *brokers* fueron RabbitMQ, MongoDB y la base de datos relacional de PostgreSQL que se utiliza en todo el sistema.

RabbitMQ es un sistema de mensajería que implementa el estándar AMQP almacenando las colas de mensajes en memoria. Es un servicio dedicado a este tipo de aplicaciones, robusto y altamente configurable. Por otro lado, PostgreSQL

y MongoDB son bases de datos (PostgreSQL relacional y MongoDB no relacional y orientada a documentos), que en ningún caso están optimizadas para este trabajo.

El requisito de mayor prioridad ha sido minimizar el tiempo de respuesta que requiere encolar una tarea. Cuando el nodo encargado de atender las consultas de la API recibe una petición es prioritario enviar una respuesta al usuario en el menor tiempo posible, minimizando así el tiempo de espera que sufre el usuario entre el envio de la petición y la recepción de la respuesta.

Por ello, en primer lugar se descartaron MongoDB, PostgreSQL y otras bases de datos de almacenamiento en disco a favor de los sistemas de almacenamiento en memoria. Sin enbargo, a pesar de que RabbitMQ parecía la opción ideal, finalmente se eligió Redis como *broker*.

Redis presenta dos ventajas sobre RabbitMQ: es un servicio más fácil de configurar y consume menos recursos. En cualquier caso, el sistema ha sido diseñado para que si en algún momento se decide utilizar otro *broker* sólo sea necesario modificar una línea en un fichero de configuración.

Apéndice B

Planificación y dedicación

El siguiente anexo muestra la planificación seguida a la hora de seleccionar la información más relevante de cada fuente, la planificación global del proyecto desglosada por objetivos y la dedicación final dedicada a cada uno de ellos.

B.1. Información recopilada por fuente

La siguiente sección muestra la información recopilada por fuente de acuerdo con los datos suministrados por cada una de ellas y conforme a sus limitaciones y normativas.

La información que se extrae de cada fuente es la siguiente tras aplicar el criterio de búsqueda seguido (información publicada desde Aragón o relacionada con las tendencias de la comunidad):

Facebook Metadatos y contenido de páginas públicas y eventos.

Twitter Metadatos y contenido de tweets públicos.

Google+ Metadatos y contenido de actividades[1] públicas.

Youtube y Vimeo Metadatos de vídeos públicos.

Instagram Metadatos de fotos y vídeos públicos.

[1]Una actividad es una nota que un usuario publica en sus Novedades en Google+.

Flickr y Pinterest Metadatos de fotos públicas.

Wikipedia Metadatos y contenido de las entradas publicadas.

Blogger, Wordpress y Blogia Metadatos de las entradas publicadas.

Foursqare Metadatos de negocios y lugares de interés.

GitHub Metadatos de repositorios públicos.

En el anexo D se valida que la extracción de la información cumple con las limitaciones y normativas existentes en cada una de las fuentes.

Los metadatos extraídos corresponden a los campos que se enumeran a continuación. No todos ellos están disponibles en todas las fuentes.

Para todas las fuentes se recogen los siguientes datos comunes:

- source_id: id interna de la fuente.
- type: tipo de información servida.
- author: nombre de usuario que posee el autor del contenido en la fuente de escucha.
- url
- captured_on: fecha en el que el dato fue recogido.
- published_on: fecha en la que el dato fue publicado (en algunas fuentes no es del todo fiable).
- raw_data: datos en crudo tal y como se recogieron.
- reason: razón por la cual se recogió el dato: información geolocalizada dentro de Aragón ('G') o información relacionada con temas relevantes para Aragón ('R').

El campo *author* se recoge de todas las fuentes excepto de:

- Foursquare: ofrece información de negocios y lugares de interés. La API no da información sobre el autor que la publica.

- Blogia: no da información sobre el autor en los métodos de sindicación que soporta (RSS).

Además de los campos anteriores, de algunas fuentes se recogen otros campos específicos como la fecha de comienzo y finalización de eventos, la posición geográfica desde la que se publicó el dato, la url del *thumbnail* (imagen en miniatura), etc. La tabla B.1 refleja qué datos adicionales se recogen para cada fuente.

Fuente	type	author	title	description	thumbnail	starts_on	ends_on	geom[2]	accuracy
Facebook Events	event	Sí	Sí	Sí	No	Sí	Sí	Sí	Sí
Facebook	text	Sí	No	Sí	No	No	No	No	No
Twitter	text	Sí	No	Sí	No	No	No	Sí	Sí
Youtube	video	Sí	Sí	Sí	Sí	No	No	Sí	Sí
Instagram	picture	Sí	No	Sí	Sí	No	No	Sí	Sí
Wikipedia	text/x-wiki	Sí	Sí	Sí	No	No	No	Sí	Sí
Blogger	text/html	Sí	Sí	Sí	No	No	No	No	No
Wordpress	text/html	Sí	Sí	Sí	No	No	No	No	No
Blogia	text/html	No	Sí	Sí	No	No	No	No	No
Vimeo	video	Sí	Sí	Sí	No	No	No	No	No
Flickr	picture	Sí	Sí	Sí	Sí	No	No	Sí	Sí
Foursquare	venue	No	Sí	No	No	No	No	Sí	Sí
Pinterest	image	Sí	Sí	Sí[3]	Sí	No	No	No	No
Google+	text	Sí	Sí	Sí	No	No	No	No	No
GitHub	code	Sí	Sí	Sí	No	No	No	No	No

Tabla B.1: Datos recopilados específicos de cada fuente.

[2]Latitud y longitud del ítem almacenado.
[3]No todas las fotos poseen descripción.

B.2. Desglose de objetivos y dedigación

La tabla B.2 refleja la planificación detallada, ordenada por hitos y objetivos, y el número de horas que se dedicó finalmente a cada uno de ellos.

Objetivos y las horas dedicas a cada tarea.

Id	Descripción	Sprints			
		1	2	3	4
Recopilación y estructuración de información.					
1	Análisis de API de redes sociales.	12	1	2	
2	Análisis del estudio del arte en técnicas de *web scraping* y *web crawling*.	10			
3	Estudio y análisis de Python y sus librerías nativas y de terceros para realizar *scraping*.	19			
4	Diseño del esquema de BD.	1	2	1	
5	Desarrollo de *scrapers*.	28	6	5	
6	Test unitarios de módulos de *scrapers* no relacionados con I/O.	3	1	1	
7	Refactorización de *scrapers* con módulos auxiliares de *scraping*.	2	4	4	1
8	Desarrollo del módulo request con el algoritmo de *backoff* a partir de los resultados observados.			16	
9	Análisis de librerías de Python y patrones de diseño para realizar un módulo *daemon*.	4			
10	Diseño y desarrollo del *daemon* y del sistema de control de *scrapers*.	12	3	2	
11	Test unitarios del *daemon*.	4	2	1	
12	Refactorización de *daemon* y *scrapers* añadiendo funcionalidad de *logger*.	6			
13	Análisis de ORMs y sistemas de acceso a la BD.	7	2		

14	Refactorización del módulo de acceso a BD de los *scrapers* adaptado al ORM elegido.		4	1	
15	Documentar información recogida por los *scrapers*.		1	2	
16	Análisis de estrategias y módulos de detección de lenguaje.				5
17	Desarrollo del módulo de detección de lenguaje a partir del análisis realizado.				1
Interfaz web.					
18	Estudio de la documentación de Django.	4	11	5	
19	Mostrar fuentes y controlar su ejecución.		2	2	
20	Acceso autenticado a la interfaz web.		1		2
21	Refactorización con una lógica de negocio común.		2	6	
22	Test unitarios de la lógica de negocio.		5	4	
23	Estudiar la API de Google Maps.	2	2		
24	Visualización de resultados.		4		2
25	Implementación del sistema de visualización de resultados agregados de las fuentes.		5	1	1
26	Control de contenidos.		2	2	
27	Visualización de uso de la API.			2	
API.					
28	Análisis de los sistemas de *accounting* y monitorización de servicios web.			8	
29	Implementar API.			8	2
30	Test unitarios de la API.			5	1
31	Estudiar las colas de tareas asíncronas, el protocolo AMQP y Celery.			16	
32	Análisis de distintos *brokers* para trabajar como *backends* con Celery: RabbitMQ, Redis...			6	
33	Monitorizar uso de la API.			3	
34	Documentar API.			3	
Despliegue y puesta en producción.					

35	Desplegar sistema de escucha.	4	2	2	
36	Desplegar interfaz web.		16	6	
37	Desplegar API.			3	
Memoria.					
38	Estudiar lenguaje de composición de textos Latex.				10
39	Lectura de recomendaciones y planificación de la documentación.				4
40	Redacción de la memoria.				89
41	Redacción de los anexos.				45
Reuniones.					
42	Reuniones semanales de seguimiento.	3	2	3	5
43	Reuniones con el cliente (*Product Owner*).	1	1	1	1
	Horas dedicadas por sprint:	122	81	121	169
	Total de horas dedicadas:	**493**			

Tabla B.2: Objetivos y horas dedicadas a cada tarea.

Apéndice C

Diseño detallado y documentación del sistema de web scraping social desarrollado

El siguiente anexo ofrece un diseño detallado del sistema de web *scraping* social desarrollado con una doble función: exponer de manera formal y detallada el diseño de los componentes del sistema y explicar su modo de uso y qué opciones se ofrecen para ampliarlo y adaptarlo a otros proyectos. Este anexo se divide en cinco secciones: introducción, funcionamiento del módulo Daemon, funcionamiento del módulo Request, desarrollo de nuevos *scrapers* y funcionamiento de la interfaz web.

C.1. Introducción

El sistema de web *scraping* social desarrollado está basado en cuatro componentes:

- Daemon.
- Herramientas auxiliares de *scraping*.
- Interfaz por línea de comandos.
- Interfaz web.

La figura C.1 muestra un diagrama de componentes de los módulos del sistema de web *scraping* social (nodo 1) y su interacción con el resto del sistema.

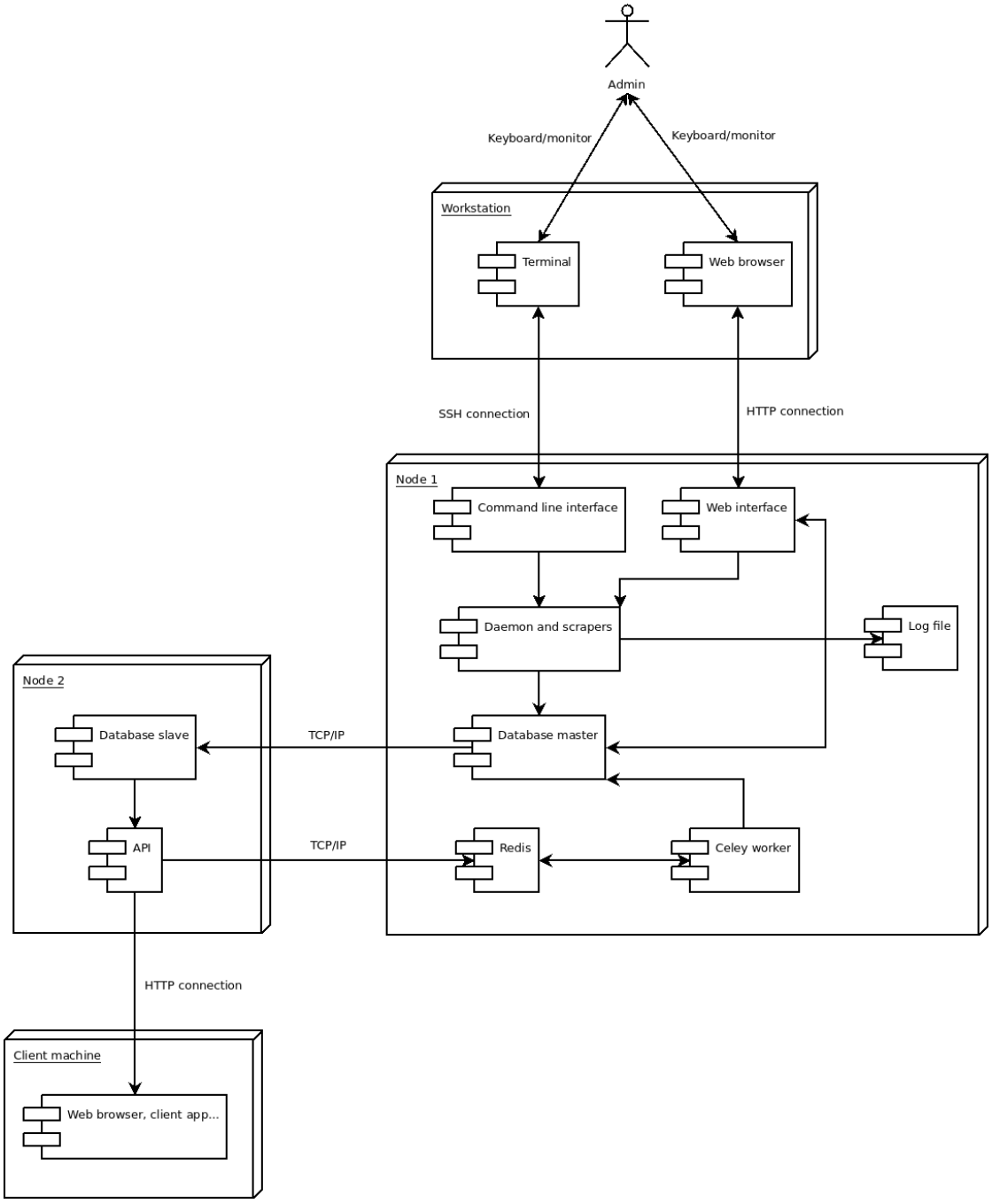


Figura C.1: Diagrama de componentes de Aragón Open SocialData.

La interfaz por línea de comandos, como se explica en la memoria principal, es tan sólo un script de Python que interactúa con el resto de los módulos del sistema de web *scraping* y permite realizar las siguientes tareas:

- Ejecutar el *daemon* (opción *rundaemon [-v]*).
- Detener el *daemon* (opción *stopdaemon*).
- Detener una tarea (opción *kill task_id*).
- Reanudar una tarea (opción *kill task_id*).
- Interactuar con los comandos de Django (opción *web [django-manage.py commands]*).

Esta interfaz sólo debería ser usada para lanzar el *daemon*, para detenerlo en caso de error o para trabajar en el entorno de desarrollo.

C.2. Daemon

El módulo Daemon es el encargado de ejecutar periódicamente las tareas programadas (*scrapers*). Tiene una doble función, por un lado realiza la labor de *scheduler* y lanzador de tareas, y, por otro, monitoriza el estado de las tareas en ejecución.

El diagrama de clases de la figura C.2 ilustra todos los componentes que forman parte de este módulo:

Daemon Clase principal del módulo. Es la encargada de ejecutar el bucle principal del *daemon* y detener correctamente el sistema de escucha ante interrupciones o señales enviadas por el sistema operativo.

DaemonController Clase que interactúa con la instancia en ejecución del *daemon*. Permite realizar las operaciones de detención del Daemon y detención o reanudación de una tarea.

DaemonRegister Interfaz cuya responsabilidad es registrar el *pid* (*Process Identifier*) del proceso que ejecuta el *daemon* en el sistema. Pueden utilizarse varias técnicas: escribir el *pid* en un fichero, registrarlo en una tabla de la

base de datos, almacenarlo en una máquina o servicio remoto, etc. (En el diagrama se representa con este nombre una clase que implementa esta interfaz registrando el *pid* en un fichero de texto del sistema).

Notifier Interfaz cuya responsabilidad es servir como mecanismo de comunicación entre el proceso que ejecuta el *daemon* y el resto de procesos que interactúan con él (interfaz por línea de comandos, interfaz web, *scrapers*...). (En el diagrama se representa con este nombre una clase que implementa esta interfaz).

TaskManager Clase encargada de controlar qué tareas están pendientes de ejecución (*scheduler*), qué tareas han finalizado ya, y qué tareas están en ejecución.

TaskRegister Interfaz cuya responsabilidad es registrar el estado de la ejecución de una tarea (*pid* del proceso en el que se ejecuta, momento en el que inicia su ejecución, momento en el que finaliza, código de salida, etc.). (En el diagrama se representa con este nombre una clase que implementa esta interfaz).

Worker Clase encargada de lanzar el subproceso encargado de ejecutar una tarea y monitorizar su ejecución.

Logger Clase encargada de registrar eventos en un fichero (*log*).

La implementación de las interfaces descritas anteriormente realizada en este proyecto es la siguiente:

DaemonRegister Registra el *pid* del *daemon* en un fichero de texto del sistema.

Notifier Notifica eventos mediante la señal *SIGUSR2* de POSIX.

TaskRegister Registra el estado de la ejecución de una tarea en una tabla de la base de datos.

En otros proyectos puede realizarse una implementación distinta de estas interfaces ampliando las funcionalidades del sistema desarrollado.

La figura C.3 ilustra como interactúan estas clases entre sí. Asimismo, la figura C.4 refleja el funcionamiento del bucle principal del *daemon*.

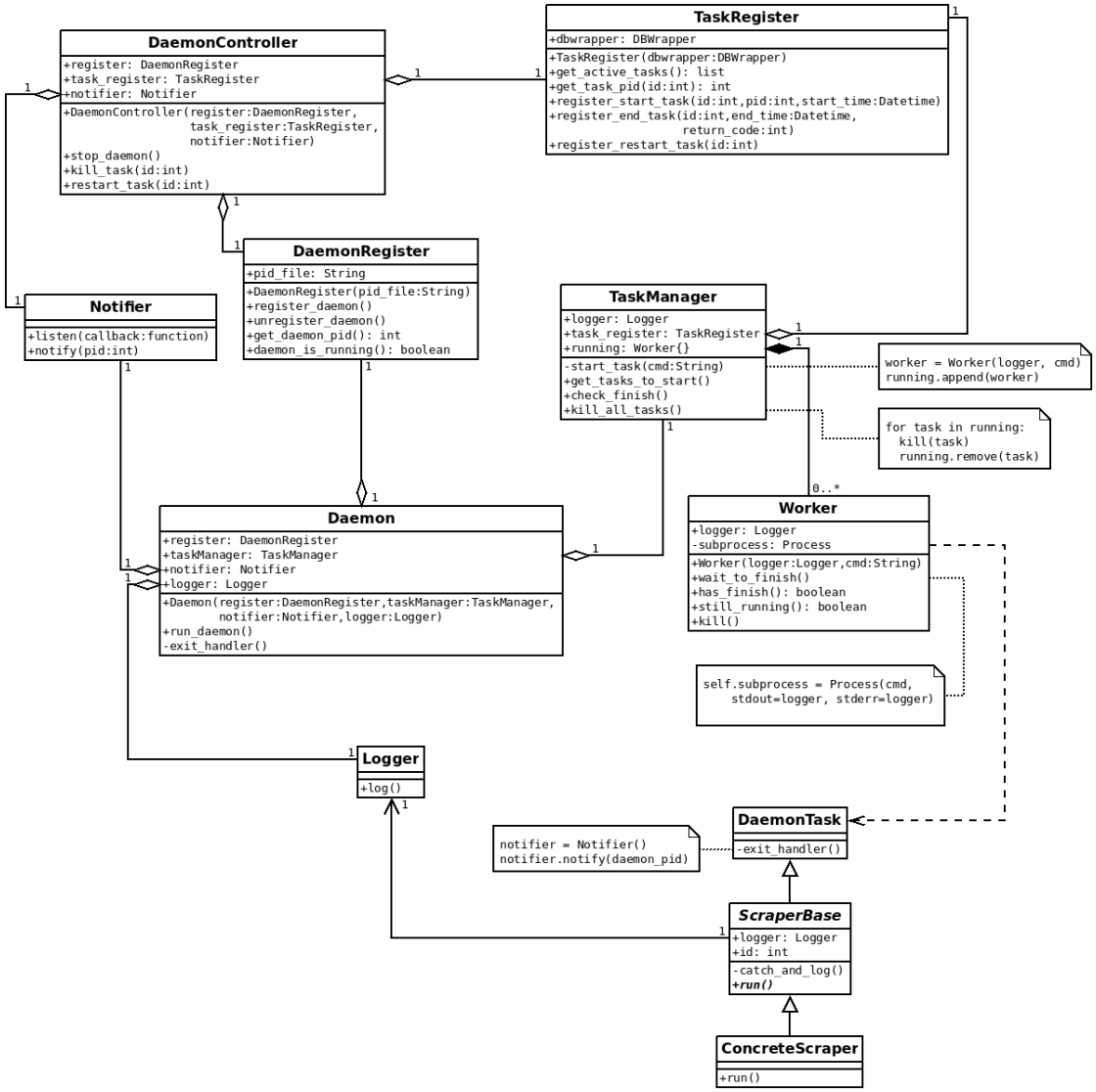


Figura C.2: Diagrama de clases del módulo Daemon.

C.3. Request

El módulo Request permite realizar peticiones HTTP utilizando diferentes mecanismos de manera tolerante a fallos. Para ello se hace uso de un algoritmo de *backoff* similar al utilizado para controlar el acceso al medio con CSMA/CA.

Ha sido diseñado con el objetivo de ser fácilmente extensible a distintas librerías y mecanismos internos (*backends*) encargados de realizar la petición real (urllib,

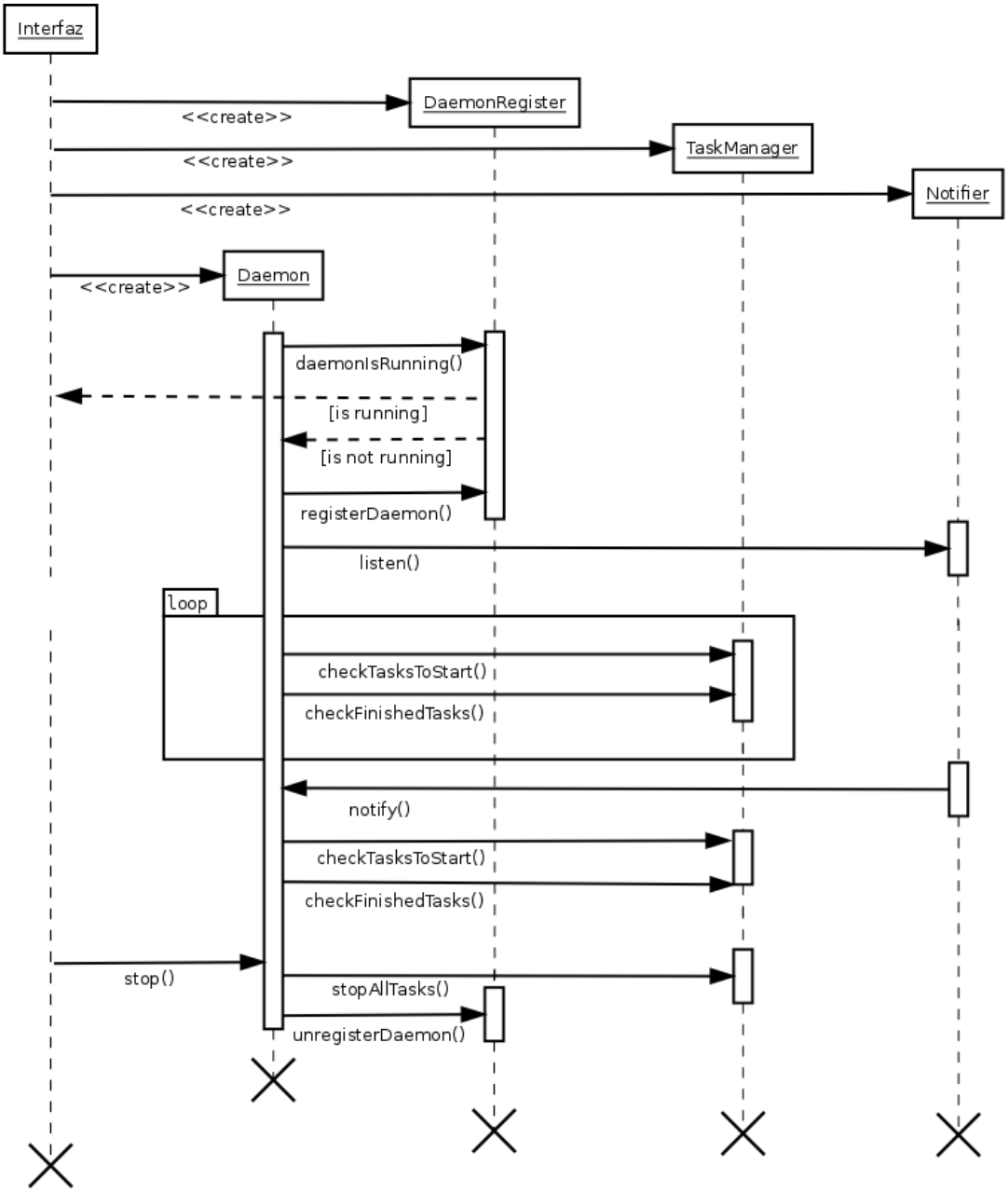


Figura C.3: Diagrama de secuencia del módulo Daemon y sus componentes.

oauth2, requests, feedparser, etc.). El método principal del módulo encapsula la petición mediante un *wrapper* y controla las excepciones que se producen en él de manera unificada para todos los *backends*.

Ofrece las siguientes características:

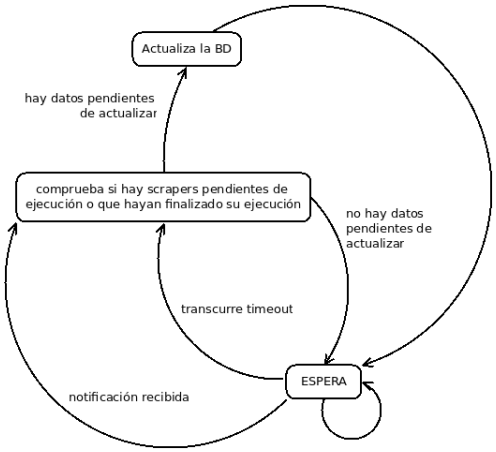


Figura C.4: Diagrama de estados del bucle principal del Daemon.

- Método unificado para obtener la respuesta, código de estado y excepciones en los *scrapers* independientemente del *backend* utilizado. Gracias a esto se simplifica el desarrollo de *scrapers* y otros módulos que realicen peticiones remotas.

- Sistema de control del tiempo de espera máximo permitido hasta recibir la respuesta del servidor (*timeout*).

- Sistema de distinción y clasificación de errores HTTP. Si recibimos una respuesta con un error HTTP 408 (*Request Timeout*), es posible que el servidor esté demasiado ocupado en este momento y que, si realizamos de nuevo la petición un poco más tarde, pueda atendernos correctamente. En cambio, si recibimos un error HTTP 400 (*Bad Request*), el error lo hemos cometido nosotros y aunque reintemos realizar la petición el resultado devuelto por el servidor será el mismo.

- Algoritmo de *backoff* encargado de reintentar las peticiones erróneas en función del código de error HTTP devuelto por el servidor.

El siguiente código ilustra de manera simplificada el funcionamiento de este algoritmo de backoff:

```
def get(url):
    count_timeouts = 0
    attempts_linear = MAX_ATTEMPS
    attempts_exponential = MAX_ATTEMPS
    backoff_linear = MIN_BACKOFF_LINEAR
    backoff_exponential = MIN_BACKOFF_EXPONENTIAL
    backoff_timeout = MIN_BACKOFF_EXPONENTIAL
```

```
while True:
  try:
    response, status_code = do_request(url)

  # Timeout. Retry until count_timeouts = max_timeouts with backoff algorithm.
  except TimeoutError:
    count_timeouts += 1
    if count_timeouts == MAX_TIMEOUTS:
      raise ServerDoesNotRespondError()
    else:
      time.sleep(backoff_timeout)
      backoff_timeout *= STEP_EXPONENTIAL
  else:
    # Correct request.
    if status_code in CODES_OK:
      return response

    # Retry max_attempts tries with linear backoff algorithm.
    elif status_code in CODES_RETRY_LINEAR:
      attempts_linear -= 1
      if attempts_linear > 0:
        time.sleep(backoff_linear)
        backoff_linear += STEP_LINEAR
      else:
        raise RequestError()

    # Retry max_attempts tries with exponential backoff algorithm.
    elif self.status_code in CODES_RETRY_EXPONENTIAL:
      attempts_exponential -= 1
      if attempts_exponential > 0:
        time.sleep(backoff_exponential)
        backoff_exponential *= STEP_EXPONENTIAL
      else:
        raise ServerDoesNotRespondError()

    elif self.status_code in CODES_EXIT:
      raise ServerDoesNotRespondError()
```

Como indica la figura C.5, se han implementado varias clases para interactuar con diversos *backends*: Requests básico, Requests con OAuth Basic, Requests con OAuth1, Requests con OAuth2, Feedparser y GdataYoutube.

Para dar soporte a más *backends* tan sólo es necesario crear una nueva clase que herede de la clase abstracta *RequestBase* e implemente el método *do_request* devolviendo la respuesta en el atributo *response*, el código de estado en el atributo *status_code*, las cabeceras (si las hay) en el atributo *headers* y, si se vence el *tiemout* establecido, lance la excepción *TimeoutError*.

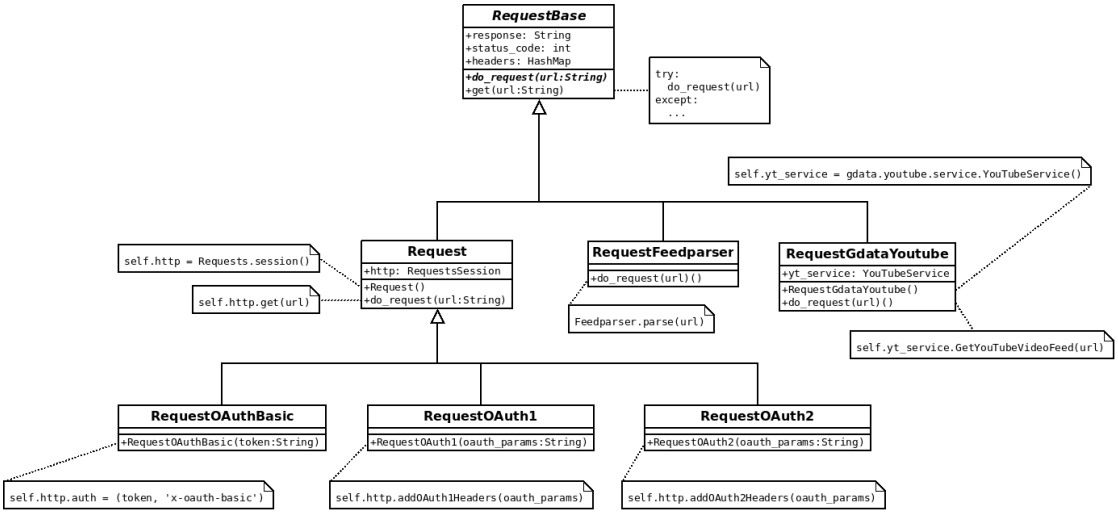


Figura C.5: Diagrama de clases del módulo Request con las clases implementadas para dar soporte a los siguientes *backends*: Requests (con soporte para varios mecanismos de OAuth), Feedparser y GdataYoutube.

C.4. Desarrollo de nuevos scrapers

El *daemon* ha sido desarrollado de manera que se pueda programar en él la ejecución de cualquier tarea. Las tareas se definen por el comando necesario para ejecutarlas, de modo que es posible ejecutar cualquier cosa como una tarea: un *scraper* en Python, otro en Ruby, un script en Bash que realice una copia de seguridad de la base de datos, otro tipo de software existente en la máquina, etc.

No obstante, para facilitar el desarrollo de nuevos *scrapers*, se ha diseñado un conjunto de métodos auxiliares que facilitan esta labor. La figura C.6 ilustra el diagrama de clases de los principales componentes de ayuda a los *scrapers* desarrollados.

Se distinguen dos tipos de *scrapers*: *scrapers* que insertan contenido en la base de datos y *scrapers* que no lo hacen. Las ventajas de utilizar estos módulos auxiliares son las siguientes:

- Notificación al *daemon* de la finalización de una tarea. El *daemon* atiende y registra la finalización de la tarea sin necesidad de esperar a que éste despierte tras transcurrir el *timeout* interno del bucle principal del *daemon*.

- Registro de excepciones no capturadas en el sistema de *logger*. Si una excep-

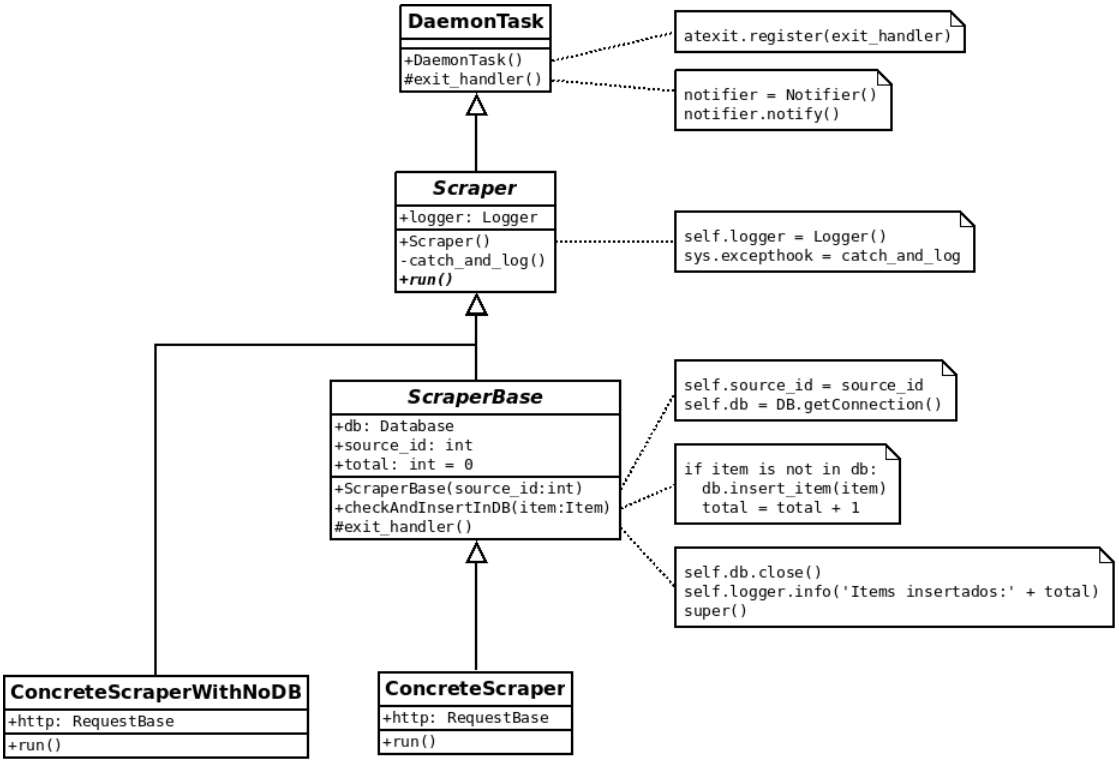


Figura C.6: Diagrama de clases de los principales componentes de ayuda a los *scrapers* desarrollados.

ción imprevista sucede, se registra en el *log* indicando el momento en el que ha sucedido, el mensaje de error y el *traceback* de la excepción. A continuación se detiene la ejecución de la tarea con el código de salida correspondiente. Dependiendo de la severidad del error el sistema de *logger* notifica la excepción al desarrollador o desarrolladores mediante correo eléctronico.

- Monitorización automática en el *log* del número de elementos insertados por un *scraper*. Herramienta útil para realizar *debug* durante la implementación de un *scraper*.

Para desarrollar un nuevo *scraper* utilizando estos módulos auxiliares se debe implementar una clase que herede de *ScraperBase* o de *Scraper*, en función de si almacena la información recopilada en una base de datos o no, e implemente el método *run*. Dicho método será el punto de entrada al *scraper*.

Por ejemplo, el *scraper* de GitHub que utiliza estos módulos auxiliares corresponde al siguiente código:

```
DELAY = 3

class Github(ScraperBase):
    http = request.RequestOAuthBasic(API_KEY)

    def process_item(self, repo):
        try:
            item = {
                'source_id': self.source_id,
                'reason': constants.REASON_REL,
                'type': constants.TYPE_CODE,
                'url': repo['html_url'],
                'author': repo['owner']['login'],
                'title': repo['name'],
                'description': repo['description'],
                'raw_data': json.dumps(repo),
                'published_on': datetime.datetime.now(),
            }
            self.check_and_insert_in_db(item)
        except Exception, e:
            self.logger.critical('Unexpected error. (...)')

    def run(self):
        keywords = self.db.get_trendings()
        for keyword in keywords:
            url = BASE_URL.format(keyword)
            try:
                response = self.http.get(url)
                data = response.json()
            except Exception, exc:
                utils.error_handler(exc, self.id, self.logger, url)
            else:
                try:
                    for repo in data['items']:
                        self.process_item(repo)
                except Exception, e:
                    self.logger.critical('Unexpected error. (...)')
            time.sleep(DELAY)
```

También se ha realizado un módulo auxiliar de detección de idiomas. Este módulo puede ser usado para filtrar y descartar corpus documentales de acuerdo a un idioma concreto en función de los campos más relevantes del contenido (*title* y *description*). Para ello se ha utilizado la librería LangId de Python, que emplea un algoritmo de aprendizaje automático mediante redes neuronales.

Otra estrategia estudiada para la detección de idiomas fue el análisis de *stopwords*, palabras carentes de significado semántico de un lenguaje como conjunciones, determinantes, etc., mediante la librería de procesamiento de lenguaje natural NLTK (*Natural Language Toolkit*) de Python. Se sometieron ambas estrategias de detección de idiomas a un conjunto de 100 pruebas con corpus documentales no triviales (mezcla de idiomas en la misma frase, faltas de ortografía, etc.) y los resultados obtenidos fueron los siguientes: un 85 % de fiabilidad utilizando la libería LangId y un 78 % empleando la estrategía de las *stopwords*.

C.5. Interfaz web

Para el desarrollo de la interfaz web se hace uso del *framework* web Django, aprovechando todas sus ventajas como el ORM (*Object-relational mapping*) incorporado, el sistema de gestión de cuentas, el panel de administración, etc.

En general, todos los ficheros relacionados con la interfaz web se encuentran en el directorio `escucha/web/`, pero se ha modificado la estructura por defecto de un proyecto Django para adaptarla a la de los demás componentes de Aragón Open SocialData.

Además, a pesar de que Django ofrece un potente ORM, el sistema ha sido diseñado utilizando *wrappers* de forma que, en el futuro, pueda emplear otros sitemas de interacción con la base de datos.

C.5.1. Configuración

El fichero de configuración se ha desglosado en varios ficheros en función de los distintos entornos de ejecución: *developtment*, *staging* y *production*. Todos ellos heredan de un mismo fichero de configuración base. Estos ficheros se encuentran en el directorio: `escucha/conf/django/`.

El uso de un fichero de configuración u otro se realiza de forma automática en función del *hostname* de la máquina que lo ejecute. Los *hostnames* están especificados en el fichero `escucha/conf/__init__.py`.

C.5.2. WSGI

Es el punto de entrada del servidor web Apache. Se ha modificado el fichero por defecto generado por Django para que cargue correctamente la configuración en función del *hostname* de la máquina que ejecute el servidor. El fichero se encuentra en `escucha/web/wsgi.py`.

Además, se ha desarrollado un *middleware* para registrar las excepciones producidas durante la ejecución del sitio web en el *log* de errores de Apache. Este *middleware* se encuentra en `escucha/web/utils/logs.py` y es necesario añadirlo al fichero de configuración de Django de los entornos que utilicen Apache como servidor web (*staging* y *production*).

Apéndice D

Aspectos jurídicos del proyecto

El siguiente anexo es un resumen de la consulta jurídica realizada a José Félix Muñoz Soro, miembro del Laboratorio Jurídico-Empresarial de la Universidad de Zaragoza, para evaluar la viabilidad ofrecida por las fuentes en cuanto a extracción, almacenamiento y redistribución de sus contenidos para su uso en el proyecto Aragón Open SocialData, en adelante AOSD[1].

La responsabilidad final del contenido extraído, almacenado y servidor por la plataforma recae sobre la Diputación General de Aragón. Ésta contrató los servicios de José Félix Muñoz y autorizó al equipo de desarrollo del BIFI la implementación de los módulos de escucha de las fuentes validadas. Este anexo se incluye únicamente como contenido orientativo de la consulta jurídica realizada.

D.1. Naturaleza del servicio, normativa y condiciones aplicables

La naturaleza del servicio Aragón Open SocialData posee unas características que requieren un minucioso análisis para poder enmarcarlo dentro de la normativa jurídica actual.

[1]La versión completa de este documento puede encontrarse en la siguiente dirección: `http://opendata.aragon.es/portal/documentacion`.

D.1.1. Ley de servicios de la sociedad de la información (LSSI)

En cuanto a la naturaleza jurídica, consideramos que Aragón Open Social Data es un PSSI y, por tanto su actividad estará regulada por la Ley 34/2002, de 11 de julio, de Servicios de la Sociedad de la Información (LSSI).

En consecuencia, la responsabilidad sobre los contenidos que publique AOSD vendrá dada por lo dispuesto en dicha norma. Consideramos que no sería de aplicación el artículo 16, que se refiere a PSSI cuyo servicio consiste en albergar datos proporcionados por el destinatario y a la información almacenada a petición del destinatario, sino el artículo 17 que se refiere a webs de enlaces y buscadores.

> **LSSI, artículo 17**: Responsabilidad de los prestadores de servicios que faciliten enlaces a contenidos o instrumentos de búsqueda.
> 1. Los prestadores de servicios de la sociedad de la información que faciliten enlaces a otros contenidos o incluyan en los suyos directorios o instrumentos de búsqueda de contenidos no serán responsables por la información a la que dirijan a los destinatarios de sus servicios, siempre que:
> a) No tengan conocimiento efectivo de que la actividad o la información a la que remiten o recomiendan es ilícita o de que lesiona bienes o derechos de un tercero susceptibles de indemnización, o
> b) Si lo tienen, actúen con diligencia para suprimir o inutilizar el enlace correspondiente. Se entenderá que el prestador de servicios tiene el conocimiento efectivo a que se refiere el párrafo a) cuando un órgano competente haya declarado la ilicitud de los datos, ordenado su retirada o que se imposibilite el acceso a los mismos, o se hubiera declarado la existencia de la lesión, y el prestador conociera la correspondiente resolución, sin perjuicio de los procedimientos de detección y retirada de contenidos que los prestadores apliquen en virtud de acuerdos voluntarios y de otros medios de conocimiento efectivo que pudieran establecerse.

D.1.2. Ley sobre reutilización de la información del sector público (LRISP)

Dentro de los objetivos de AOSD cabría distinguir dos niveles, uno en el que la información se publica para que el usuario pueda conocer los contenidos publicados

en y sobre Aragón, haciendo exclusivamente un uso personal de los mismos, y otro en el que además se permita la reutilización de los contenidos por parte de los usuarios. Respecto a esta segunda posibilidad no sería de aplicación, al menos a una parte importante de los contenidos, la Ley 37/2007, de 16 de noviembre, sobre reutilización de la información del sector público (LRISP), ya que el ámbito de la misma se circunscribe a la información del sector público.

Por tanto, entendemos que la posible reutilización de los contenidos debería regirse por las normas de derecho privado y, en particular, las referentes a la propiedad intelectual. Ello implica que en algunos casos, como por ejemplo en los medios de comunicación, la reutilización por terceros podría no ser posible, salvo que se contara con una autorización expresa de las fuentes de las que se haya obtenido la información.

Además, de forma general, debe tenerse en cuenta que en las condiciones establecidas por una parte importante de las fuentes se prohíbe expresamente la utilización de los contenidos con fines comerciales. En este informe se considera que esta objeción no afecta a AOSD, dado que sus fines no son lucrativos. Sin embargo, si se permite que terceros reutilicen con fines lucrativos los contenidos accedidos en AOSD se estará violando esta condición. Por ello se recomienda no autorizar la reutilización con fines comerciales o lucrativos.

D.1.3. Condiciones

Además del marco legal, en el desarrollo y gestión de Aragón Open SocialData será preciso tener muy en cuenta las condiciones particulares establecidas por los PSSI de los que se va a tomar la información agregada. También, en algunos casos, habrá de estarse a las condiciones puestas directamente por los autores para la explotación de los contenidos.

La práctica totalidad de las condiciones particulares contemplan la posibilidad de reforma unilateral por parte del PSSI, sin que se admita obligación alguna derivada de estas modificaciones ni posibilidad de oponerse por parte de los usuarios. El resultado es que las opciones sobre los contenidos han de tomarse como están en cada momento, partiendo únicamente de esta situación para estudiar el contexto en el que actúa Aragón Open SocialData. Pero, éste puede cambiar en aspectos relevante y sin previo aviso, lo que crea cierta indeterminación sobre el servicio que, desde un punto de vista jurídico, no puede evitarse. Sin embargo, las prácticas seguidas por los PSSI hacen que sea previsible una estabilidad suficiente en el marco definido por las mismas, como para mantener en su conjunto el servicio

previsto, aunque puedan producirse en el mismo pequeñas modificaciones.

D.2. Datos de carácter personal

El principio general establecido por la Ley Orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal (LOPD) es que todo tratamiento de datos de carácter personal, así como la comunicación de los datos a terceros, precisan del consentimiento del titular de los datos.

D.2.1. Carácter de fuentes accesibles al público de los orígenes de información

Dentro de la información manejada y publicada por AOSD podrían incluirse datos personales -es decir, correspondientes a una persona física. Identificada o identificable-, por lo que es preciso determinar si AOSD debe obtener el consentimiento de los titulares de los datos.

Consideramos que puede recurrirse a la excepción que para la necesidad de obtener el consentimiento se establece en el artículo 6.2o de la LOPD, aplicable a aquellos datos que figuren en fuentes accesibles al público, las cuales se definen en la misma ley del siguiente modo:

> **LOPD, artículo 3**
> j) Fuentes accesibles al público: aquellos ficheros cuya consulta puede ser realizada, por cualquier persona, no impedida por una norma limitativa o sin más exigencia que, en su caso, el abono de una contraprestación. Tienen la consideración de fuentes de acceso público, exclusivamente, el censo promocional, los repertorios telefónicos en los términos previstos por su normativa específica y las listas de personas pertenecientes a grupos de profesionales que contengan únicamente los datos de nombre, título, profesión, actividad, grado académico, dirección e indicación de su pertenencia al grupo. Asimismo, tienen el carácter de fuentes de acceso público los diarios y boletines oficiales y los medios de comunicación.

Entendemos que la práctica totalidad de las fuentes de información previstas para el proyecto se ajustan a la primera condición exigida por esta definición, ya que pueden ser accedidas por cualquier persona. Analizamos a continuación, de

forma más pormenorizada, cada uno de los grupos en que las hemos dividido.

Medios de comunicación

En este grupo incluimos a las páginas web institucionales, páginas de edición colaborativa (wikis), agregadores, blogs, mediablogging, medios de comunicación y foros. Algunas de las fuentes anteriores forman parte claramente de la categoría tradicional de medios de comunicación, como es el caso de los propios medios de comunicación y paginas institucionales. En cuanto a las demás fuentes creemos que también pueden considerase medios de comunicación, al menos en los aspectos que interesan a la hora de valorar su carácter de fuentes accesibles al público.

Redes sociales

Dentro de este apartado incluimos combinaciones sociales y microblogging. En el caso de las redes sociales, la situación respecto a la protección de datos es más compleja, ya que en las mismas no todos los contenidos se incluyen con la finalidad de ser publicados de forma que cualquiera pueda tener acceso a los mismos. Es el propio usuario quien puede configurar su perfil de privacidad, normalmente con las siguientes opciones:

1. La información sólo puede ser vista por sus amigos o contactos.
2. La información puede ser vista por sus amigos o contactos y por los amigos o contactos de estos.
3. La información puede ser vista por todo el mundo.

Entendemos que podrían considerarse como fuente accesible al público las publicaciones de los usuarios que tengan activa la tercera opción por lo que, siendo estos los únicos contenidos de las redes a los que accederá AOSD, no sería preciso el consentimiento del interesado.

D.2.2. Necesidad de finalidad legitima

La LOPD establece dos condiciones que deben cumplirse en los tratamientos de datos procedentes de fuentes accesibles al público para que actúe la excepción a la

necesidad de obtener el consentimiento de los titulares que acabamos de exponer. La primera es que el tratamiento sea necesario para la satisfacción del interés legítimo perseguido por el responsable el fichero y, la segunda, que no se vulneren los derechos y libertades fundamentales del interesado.

La finalidad de AOSD es legítima, ya que se trata de un servicio para la ciudadanía que facilita una observación activa y en tiempo real de toda la información relacionada con Aragón que se incorpora a Internet, y funciona bajo los principios de objetividad y neutralidad.

Pese a ello, se habilita un mecanismo mediante el cual es posible para los ciudadanos origen de la información excluir sus publicaciones de la agregación por parte de AOSD (en adelante op-out). En algunos supuestos éste puede ser un derecho derivado de la condición de autor de la información y, por tanto, será obligatorio habilitar esta opción op-out; en otros, su objetivo es evitar que un ciudadano sienta que sus publicaciones son agregadas contra su voluntad.

D.2.3. Posible publicación de datos de terceros

Las fuentes de información a utilizar en el proyecto AOSD son gestionadas por entidades ajenas y están bajo la responsabilidad de estas. En muchos casos se trata de PSSI que, a su vez, integran información procedente de los ciudadanos. Por ello, puede plantearse el problema de que, aunque es una práctica prohibida en las condiciones de uso de todas las fuentes de información, los ciudadanos incluyan en la información que publican datos de carácter personal de otras personas y que, finalmente, estos acaben siendo hechos públicos en el sitio web de AOSD.

En este caso la responsabilidad viene determinada por el artículo 17 de la LSSI y, de acuerdo con el mismo, la obligación de AOSD es actuar diligentemente para la retirada de dicha información en cuanto tenga noticia de su existencia.

D.3. Propiedad intelectual

La Propiedad intelectual está regulada en nuestro país por el Real Decreto Legislativo 1/1996, de 12 de abril, que aprueba el Texto Refundido de la Ley de propiedad intelectual (en adelante LPI). De los cuatro actos de explotación previstos en la misma (copia, distribución, difusión y transformación), AOSD realizará

en todo caso una reproducción (copia) de los contenidos, aunque sea parcial, y un acto de difusión.

La copia y conservación de los contenidos agregados durante un largo periodo de tiempo no es en sí misma un acto de explotación diferenciado de la simple reproducción, pero desde el punto de vista práctico, sí que tiene en muchas ocasiones características diferenciadas. Es el caso, por ejemplo, de los medios de comunicación en los que es frecuente que los contenidos del día sean difundidos de forma gratuita mientras que el acceso al archivo (hemeroteca) constituye un servicio de pago. Por ello será preciso establecer para cada uno de los orígenes de información en qué condiciones es posible realizar la copia y conservación de los contenidos.

En el caso de AOSD, como se utilizan contenidos puestos a disposición del público por distintas fuentes, deberán tenerse en cuenta las condiciones establecidas respecto a los derechos de propiedad intelectual de cada una de ellas. A continuación se resume el contenido de estas con la finalidad de determinar las consecuencias que cabe extraer de las mismas para AOSD. Además, se distingue la normativa aplicable para los dos tipos de utilización del contenido de los que AOSD puede hacer uso: **visualización en el origen** y **visualización en AOSD**.

Twitter **Visualización en AOSD**. Los contenidos pueden reutilizarse sin limitaciones específicas, siempre que se acceda a los mismos a través del API que proporciona la empresa.

Facebook **Visualización en AOSD**. Se permite la copia y publicación de todos los contenidos que el usuario haya calificado como *Public*.

Google+ **Visualización en AOSD**. Se permite la copia y publicación de todos los contenidos que el usuario haya compartido con la opción *público*.

Youtube **Visualización en el origen**. Utilizando obligatoriamente el reproductor de YouTube.

Instagram **Visualización en el origen**. No se autoriza expresamente la copia del contenido por parte de Instagram, lo que sería necesario según la LPI.

Vimeo **Visualización en AOSD**. Los videos pueden reproducirse con el reproductor de Vimeo, pero no se exige que sea así en todo caso.

Flickr **Visualización en el origen**. No se prohíbe expresamente la copia del contenido, pero este no puede ser publicado en AOSD.

Foursquare **Visualización en AOSD**. Se permite la copia y publicación de todas las publicaciones del usuario, que este haya compartido de forma pública.

Pinterest **Visualización en el origen**. No se prohíbe expresamente la copia del contenido, pero esta no queda amparada por la licencia que el usuario otorga sobre los contenidos.

Wikipedia **Visualización en AOSD**. Se permite la copia y publicación siempre que se cite a la fuente y se licencie el contenido en una modalidad equivalente.

Blogger **Visualización en AOSD**. Se permite la copia y publicación de todos los blogs.

Wordpress **Visualización en AOSD**. Se permite la copia y publicación de todos los blogs.

Blogia **Visualización en AOSD**. Se permite la copia y publicación de todos los blogs.

GitHub **Visualización en AOSD**. No hay inconveniente en reutilizar la información del sitio.

Apéndice E

Manual de usuario de la API

Para acceder a los servicios ofrecidos por la API de Aragón Open SocialData es necesario acceder a una URL distinta en función del servicio solicitado. Todas las URL están compuestas por dos partes, la base y los *endpoints* de cada servicio.

La URL base es la siguiente:

`http://analyzer2.bifi.unizar.es:8080/socialdata`

Un *endpoint*, dentro del contexto de una API REST, es un mecanismo que permite ejecutar metodos remotos y obtener los resultados de ellos, de esta manera los usuarios de la API pueden realizar consultas sobre los datos. La API de Aragón Open SocialData provee dos endpoints, uno para recoger información relacionada con las tendencias del momento en Aragón y otra para acceder al contenido sin agregar.

E.1. Trending topics

Para obtener los temas más relevantes en Aragón en tiempo real utilizaremos la llamada:

`http://analyzer2.bifi.unizar.es:8080/socialdata/trendings`

Esta llamada nos dará como resultado los trending topics de Aragón con el siguiente formato de respuesta:

```
{
  "status": "OK",
  "results": [result_1, result_2, result_3...]
}
```

status Indica el estado de la llamada. "OK" indica éxito, "NOK" indica error.

results Es un vector con los resultados. Cada uno de ellos es de nuevo un objeto JSON con la forma:

```
{
  "name": name,
  "url": url
}
```

name Texto del trending topic.

url Enlace directo al trending topic.

E.2. Contenido

Para obtener el contenido sin agregar hay que realizar una llamada a la API de la siguiente manera:

`http://analyzer2.bifi.unizar.es:8080/socialdata/data?parameters`

Los parámetros que permiten filtrar los resultados y hacer búsquedas más avanzadas son los siguientes y pueden combinarse como el usuario lo desee.

E.2.1. Filtrado por tipo de contenido

A través de este sistema se escuchan varios tipos de contenido que se irán ampliando conforme la plataforma vaya avanzando y mejorando.

`/socialdata/data?type=content_type`

Donde *content_type* puede tomar uno de los siguientes valores (entre paréntesis las fuentes de las que se poseen datos de cada tipo de contenido):

- event (Facebook Events).
- text (Twitter, Facebook, Google+, Blogger, Blogia, Wordpress).
- wiki (Wikipedia).
- video (Youtube, Vimeo).
- picture (Instagram, Flickr, Pinterest).
- venue (Foursquare).
- code (Github).

En caso de no indicar este parámetro se devolverán los resultados de todos los tipos de contenido.

E.2.2. Filtrado por fuente de los datos

Se puede filtrar por las fuentes de datos escuchadas por el sistema.

`/socialdata/data?source=source_name`

Donde *source_name* puede ser:

- facebook_events
- twitter
- youtube
- instagram
- facebook
- wikipedia
- blogger
- wordpress
- blogia

- vimeo
- flickr
- foursquare
- pinterest
- google_plus
- github

En caso de no indicar ninguna se buscará contenido en todas ellas.

E.2.3. Filtrado por conversación

Se capturan datos en base a dos criterios. Por un lado, se capturan datos respecto a su geoposición, que corresponden a conversaciones sobre qué se está hablando dentro de Aragón. Por otro, se capturan contenidos en base a temas y usuarios que son de especial relevancia para Aragón.

Para seleccionar los datos de una de las dos conversaciones hay que añadir el parámetro *conversation_ type*. Si este parámetro no se añade se devolverá contenido de ambas conversaciones.

```
/socialdata/data?source=twitter&conversation=conversation_type
```

Donde *conversation_ type* puede tomar los valores:

- geo: conversación sobre lo que se habla dentro de Aragón.
- rel: conversación de temas y usuarios relevantes para Aragón.

La tabla E.1 indica el tipo de conversación disponible para cada fuente.

Tipo de conversación por fuente.

Fuente	Geo	Rel
facebook_events	Sí	No
twitter	Sí	Sí
youtube	Sí	No
instagram	Sí	No
facebook	Sí	No
wikipedia	Sí	No
blogger	No	Sí
wordpress	No	Sí
blogia	No	Sí
vimeo	No	Sí
flickr	Sí	No
foursquare	Sí	No
pinterest	No	Sí
google_plus	No	Sí
github	No	Sí

Tabla E.1: Tipo de conversación escuchada por fuente.

E.2.4. Filtrado por geoposición

Se podrá filtrar contenido por la posición desde la que fue publicado de tres formas distintas: dando un centro y un radio en kilómetros, dando un cuadrado geográfico (*bounding box*), o dando el nombre de un municipio y un radio en kilómetros.

En caso de no indicar este filtro, se mostrarán todos los resultados. Las tres formas de filtrado son excluyentes, es decir, no se pueden utilizar a la vez, ya que sólo se atenderá a una de ellas.

- Filtrado por centro y radio

  `/socialdata/data?center=coordinates&distance=distance_in_km`

  coordinates Tupla de números reales con formato latitud, longitud.

distance Entero que indica el radio, en kilómetros, alrededor del centro donde buscar.

Ejemplo: `/socialdata/data?center=41.35678,-0.8148576&distance=5`

Esta llamada buscará contenido alrededor del punto con latitud 41.35678 y longitud -0.8148576 en un radio de 5 Km.

- Filtrado por cuadrado geográfico

  `/socialdata/data?bbox=min_lat,max_lat,min_lng,max_lng`

  min_lat, max_lat, min_lng, max_lng Reales con formato 41.35678. Deben ir estrictamente en ese orden.

  Ejemplo: `/socialdata/data?bbox=-0.8148576,41.35678,-0.667584,41.78553`

- Filtrado por municipio y radio

  `/socialdata/data?locality=locality&distance=distance_in_km`

  locality Cadena de texto con el nombre del municipio.

  distance Entero que indica el radio, en kilómetros, alrededor del centro del municipio donde buscar.

  Ejemplo: `/socialdata/data?locality=Zaragoza&distance=5`

  Esta llamada buscará contenido publicado desde el centro de Zaragoza en 5 Km a la redonda.

En las llamadas en las que hay que indicar distancia, si ésta no se especifica, se toman 5 Km como distancia por defecto.

E.2.5. Filtrado por periodo

Se puede filtrar contenido publicado en un intervalo de tiempo determinado. Para ello se debe indicar la fecha de inicio del mismo y, opcionalmente, la fecha de fin. En caso de no indicar una fecha de fin se toma como ésta la última fecha de la que se poseen datos:

`/socialdata/data?start_date=start_date&end_date=end_date`

start_date y end_date Cadenas de texto con el siguiente formato: `dd/mm/yyyy`.

En caso de no indicar un intervalo temporal se mostrarán todos los resultados.

E.2.6. Filtrado por palabra clave

Se puede filtrar contenido en función de palabras clave que su autor haya publicado en alguno de los campos del mismo.

```
/socialdata/data?query=query
```

query Cadena de texto que contiene la frase de búsqueda (admite espacios).

Ejemplo: `/socialdata/data?query=pilares2013`

En caso de incluir más de una palabra clave, la consulta se realizará utilizando el operador lógico **AND** entre ellas.

E.2.7. Datos geolocalizados

Se puede especificar que la API devuelva unicamente datos geolocalizados con el parámetro *geolocated*:

```
/socialdata/data?source=twitter&geolocated=true
```

E.2.8. Paginación

Con el fin de optimizar el flujo de información, los resultados se devuelven paginados con un límite de 20 por página, excepto en las búsquedas por periodo donde se devuelven 1000 resultados por página.

Para cambiar de página sólo es necesario incluir un nuevo parámetro a la URL:

```
/socialdata/data?page=page_number
```

Si no se añade este parámetro se devuelve siempre la primera página.

Al obtener un resultado de la API, en él se indica el número de página actual en el que nos encontramos y el número de páginas totales para esa búsqueda.

E.2.9. Resultados

Los resultados, por el momento, se ofrecen en formato JSON y son devueltos ordenados cronológicamente de más recientes a más antiguos.

Si se detecta un error durante el procesamiento de la petición el resultado devuelto tendrá la siguiente forma:

```
{
  "status": "NOK",
  "error_message": message
}
```

status Indica el estado de la llamada. "NOK" indica error.

error_message Cadena de texto con los detalles del error.

En caso de que el resultado de la llamada sea correcto tendrá el formato:

```
{
  "status": "OK",
  "page": page,
  "total_pages": total_pages,
  "results": [result_1, result_2, result_3...]
}
```

status Indica el estado de la llamada. "OK" indica éxito.

page Indica el número de página de resultados actual.

total_pages Entero que indica el número de páginas con resultados que cumplen las condiciones de la llamada.

results Vector con los resultados. Cada uno de ellos es de nuevo un objeto JSON con la forma:

```
{
  "type": type,
  "source": source,
  "title": title,
  "description": description,
  "url": url,
  "author": author,
  "starts_on": starts_on,
  "ends_on": ends_on,
  "published_on": published_on,
  "lat": latitude,
  "lng": longitude,
  "thumbnail": thumbnail
}
```

type Cadena que indica el tipo de contenido (valores `content_type` listados anteriormente).

source Cadena de texto que indica a qué fuente pertenece el resultado.

title Cadena con el título del contenido en el caso de que lo tenga. description: contenido del elemento (texto del tweet, descripción del evento, descripción del vídeo, etc.).

url Dirección url directa al contenido.

author Cadena de texto con el nombre del autor en caso de que lo haya.

starts_on Para los eventos, si la tuvieran, fecha de comienzo.

ends_on Para los eventos, si la tuvieran, fecha de final.

published_on Fecha de publicación del contenido.

latitude Latitud del contenido en formato real.

longitude Longitud del contenido en formato real.

thumbnail Dirección url del thumbnail asociado si lo hubiera.

Todas las fechas tienen el formato: 16/01/14 13:03.

E.2.10. Datos en crudo

Se puede especificar que la API devuelva los datos en crudo, tal y como fueron recogidos de la fuente, añadiendo el parámetro *raw_mode* a la petición:

```
/socialdata/data?source=twitter&raw_mode=true
```

De esta manera se añade un nuevo campo a los resultados llamado *raw_ data* que contiene los datos tal cual fueron extraídos de la fuente (JSON, HTML, etc.)

Apéndice F

Manual de usuario de la interfaz web

El presente manual está dirigido a los usuarios de la interfaz web de administración y gestión del proyecto Aragón Open SocialData.

El acceso a esta interfaz está restringido. Sólo está autorizado el acceso a ella a los usuarios que hayan sido dados de alta en la plataforma. Por ello, se dividen los usuarios con acceso a este servicio en dos roles:

Usuarios *administradores* Tienen acceso completo a todas las funcionalidades de la intefaz. Además pueden gestionar a los usuarios que tienen acceso a ella como se explica en la sección F.5.

Usuarios *invitados* Tienen acceso restringido a ciertas funciones. Pueden ver todos los resultados mostrados en la interfaz, pero no pueden modificar nada salvo su contraseña.

Este manual es válido para ambos tipos de usuario y en cada sección se explicará qué funcionalidades están restringidas para los usuarios *invitados*.

F.1. Acceso y cambio de contraseña

La interfaz web de administración y gestión del proyecto Aragón Open SocialData está disponible en la siguiente URL:

`http://analyzer1.bifi.unizar.es:8080`

F.1.1. Acceso a la plataforma

Tras acceder a la dirección anterior aparece la pantalla de bienvenida, la cual le pedirá unas credenciales (usuario y contraseña) para poder continuar (figura F.1).

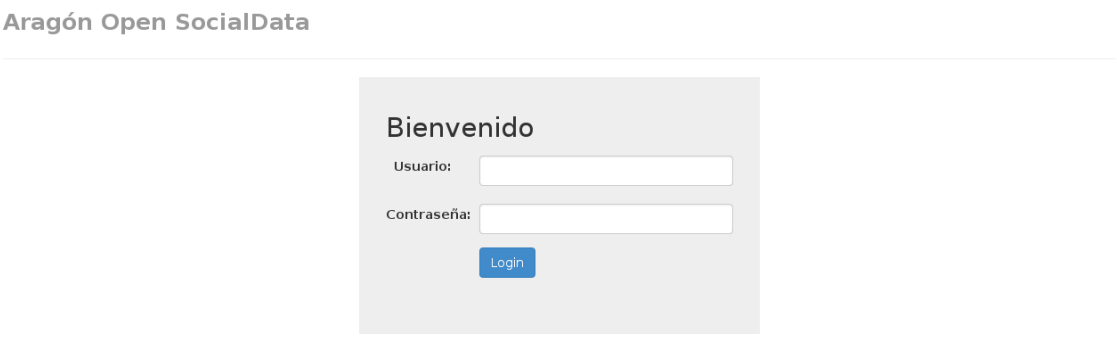


Figura F.1: Pantalla de acceso.

Si no dispone de estas credenciales contacte con un administrador de la plataforma para que autorice su acceso.

Tras introducir sus datos y pulsar el botón *login* habrá ingresado correctamente en la interfaz. En caso de que el usuario no exista o los datos intrucidos sean incorrectos verá un mensaje de error como el mostrado en la figura F.2. Si ha olvidado su contraseña contacte con un administrador de la plataforma para que le genere una nueva.

Para cerrar la sesión pulse el icono con la rueda dentada de la parte superior derecha y, a continuación, pulse en la opción *Logout* del menú desplegable que aparece (figura F.3).

Se recomienda cerrar la sesión una vez finalizado el uso de la plataforma, especialmente si la máqina desde la que se accede es compartida y otras personas, a parte de usted, tienen acceso a ella.

Figura F.2: Mensaje de error en la pantalla de acceso.

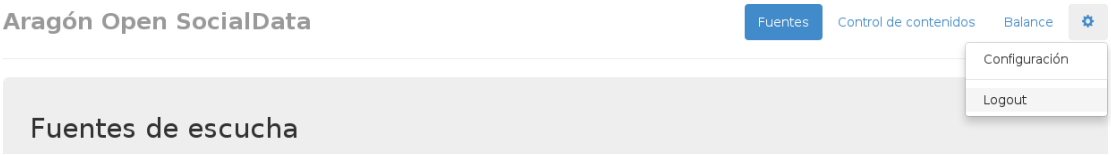


Figura F.3: Cerrar la sesión actual.

F.1.2. Cambio de contraseña

Para mejorar la seguridad de la plataforma es aconsejable que las cuentas de usuario sean individuales y no se compartan. Por ello, una vez que su cuenta esté creada, debe modificar la contraseña que el administrador le ha asignado por defecto.

Para cambiar la contraseña debe pulsar el icono con la rueda dentada de la parte superior derecha y, a continuación, pulse en la opción *Configuración* del menú desplegable que aparece como se muestra en la figura F.4.

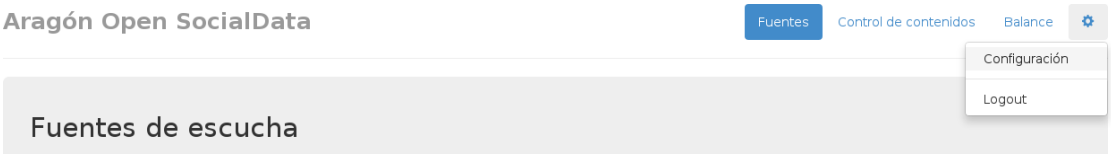


Figura F.4: Acceso a la pantalla de cambio de contraseña.

A continuación verá una pantalla como la de la figura F.5. En ella debe introducir su antigua contraseña y la nueva. Si todo ha ido bien volverá a la pantalla inicial. Si se ha producido algún error verá un mensaje con información sobre el fallo. Siga las instrucciones de dicho mensaje para actualizar correctamente su contraseña.

Figura F.5: Pantalla de cambio de contraseña.

F.2. Gestión y monitorización de las fuentes de escucha

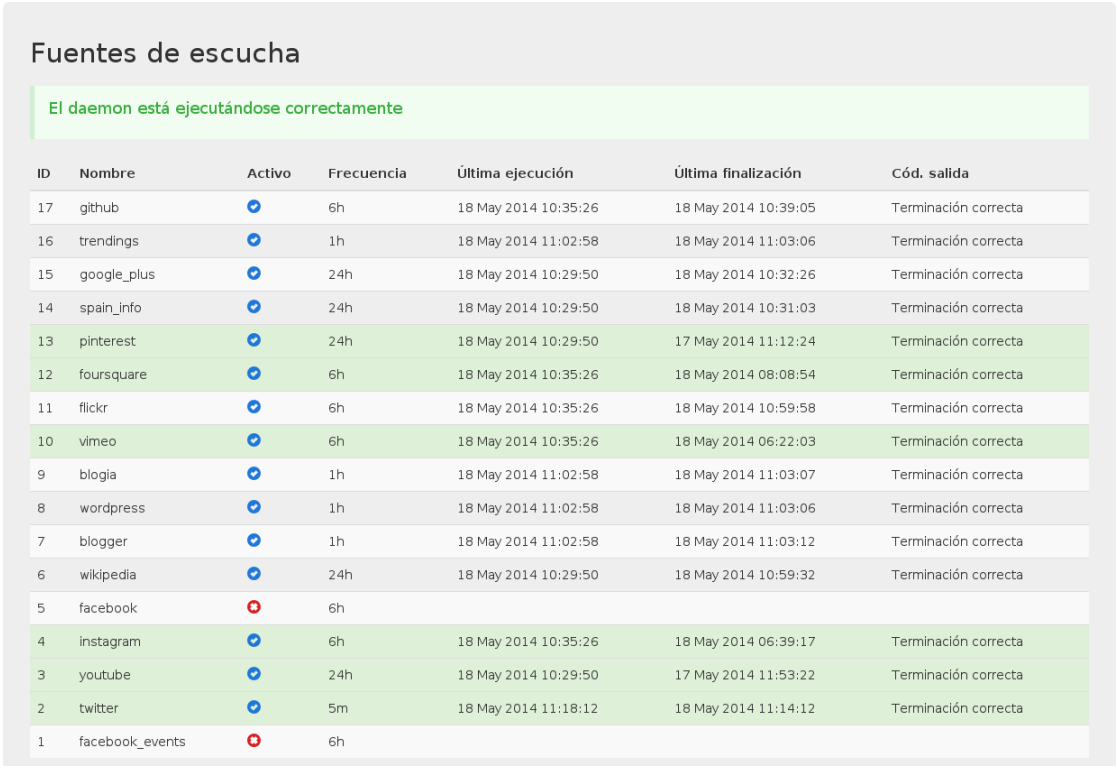
Desde la interfaz web puede monitorizar el estado de la ejecución de los módulos de extracción de datos de las fuentes y gestionar estos módulos. Para acceder a esta sección debe pulsar sobre la pestaña *Fuentes* del menú de navegación superior (figura F.6).

Figura F.6: Acceso a la sección de gestión y monitorización de las fuentes de escucha.

F.2.1. Monitorización del estado de las fuentes de escucha

El acceso a esta zona está permitido tanto para usuarios *administradores* como para usuarios *invitados*.

La pantalla de gestión y monitorización de fuentes se divide en dos partes. La zona superior muestra una tabla detallada (figura F.7) con la información de los módulos de extracción de datos de las fuentes (*scrapers*). También se muestra el estado general del sistema de escucha (*Daemon*).



ID	Nombre	Activo	Frecuencia	Última ejecución	Última finalización	Cód. salida
17	github	✓	6h	18 May 2014 10:35:26	18 May 2014 10:39:05	Terminación correcta
16	trendings	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:06	Terminación correcta
15	google_plus	✓	24h	18 May 2014 10:29:50	18 May 2014 10:32:26	Terminación correcta
14	spain_info	✓	24h	18 May 2014 10:29:50	18 May 2014 10:31:03	Terminación correcta
13	pinterest	✓	24h	18 May 2014 10:29:50	17 May 2014 11:12:24	Terminación correcta
12	foursquare	✓	6h	18 May 2014 10:35:26	18 May 2014 08:08:54	Terminación correcta
11	flickr	✓	6h	18 May 2014 10:35:26	18 May 2014 10:59:58	Terminación correcta
10	vimeo	✓	6h	18 May 2014 10:35:26	18 May 2014 06:22:03	Terminación correcta
9	blogia	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:07	Terminación correcta
8	wordpress	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:06	Terminación correcta
7	blogger	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:12	Terminación correcta
6	wikipedia	✓	24h	18 May 2014 10:29:50	18 May 2014 10:59:32	Terminación correcta
5	facebook	✗	6h			
4	instagram	✓	6h	18 May 2014 10:35:26	18 May 2014 06:39:17	Terminación correcta
3	youtube	✓	24h	18 May 2014 10:29:50	17 May 2014 11:53:22	Terminación correcta
2	twitter	✓	5m	18 May 2014 11:18:12	18 May 2014 11:14:12	Terminación correcta
1	facebook_events	✗	6h			

Figura F.7: Tabla con la información de los módulos de extracción de datos de las fuentes (*scrapers*).

La información disponible es la siguiente:

ID Identificador interno de la fuente de escucha. Si sucede un error crítico y la interfaz web no responde, el administrador del sistema puede detener o re-

iniciar el *scraper* manualmente desde la línea de comandos conociendo este identificador.

Nombre Nombre de la fuente de escucha.

Activo Indica si el *scraper* está activo o no. Si está inactivo nunca recogerá datos de esa fuente. Si está activo, cuando el tiempo indicado en la columna frecuencia haya finalizado, empezará a escuchar a la fuente.

Frecuencia Frecuencia con la que la fuente es escuchada. Indica el tiempo necesario que ha de pasar desde la última ejecución del *scraper* hasta que el *scraper* vuelva a ser ejecutado (siempre que haya finalizado su ejecución anterior).

Última ejecución Momento en el que el *scraper* comenzó su ejecución por última vez.

Última finalización Momento en el que el *scraper* finalizó su ejecución por última vez. Si el *scraper* está ejecutándose mostrará la fecha del instante en el que el *scraper* finalizó en su ejecución anterior.

Código de salida Estado de la última ejecución finalizada del *scraper*.

La fila de los *scrapers* que están actualmente en ejecución tiene un fondo de color verde.

La zona inferior muestra una gráfica (líneas de tiempo) mediante la cual puede observar los momentos de inicio y fin de cada *scraper* y la duración de estos (figura F.8). Si coloca el cursor del ratón sobre la línea de tiempo de un *scraper* obtendrá información adicional sobre su ejecución (figura F.9).

F.2.2. Control de las fuentes de escucha

Las operaciones descritas en esta sección sólo son accesibles para usuarios *administradores*.

Al acceder como usuario *administrador* a la sección *Fuentes* tal y como indica la figura F.6 vera una pantalla algo distinta a la que ve un usuario *invitado* (figura F.10).

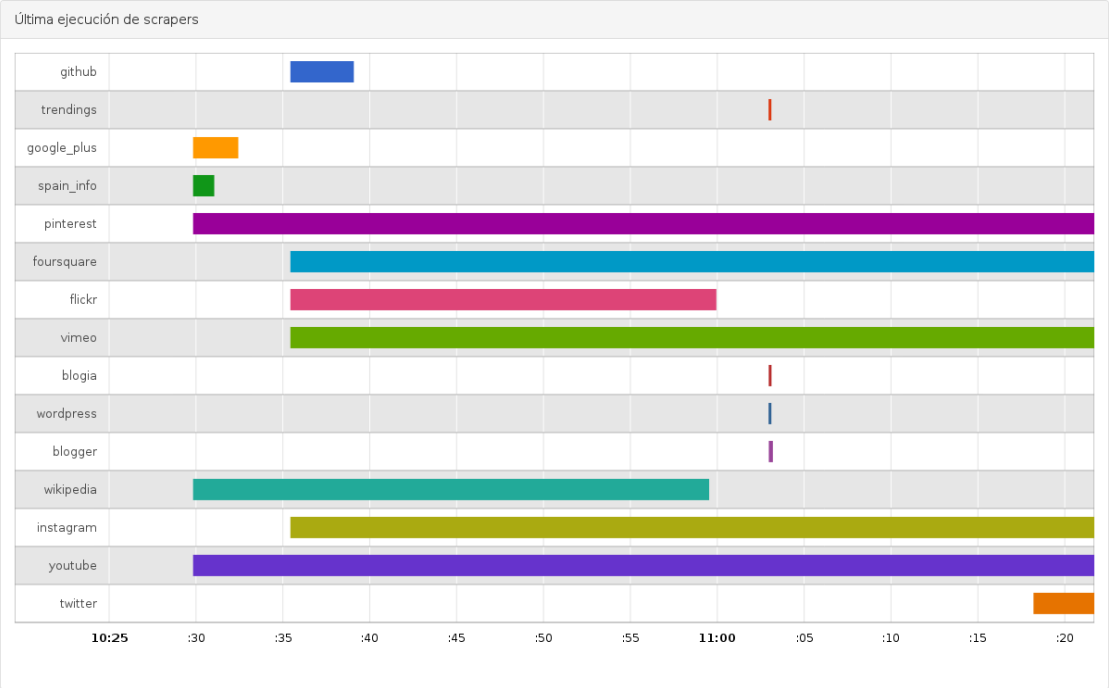


Figura F.8: Gráfico con el *timeline* de la última ejecución de cada *scraper*.

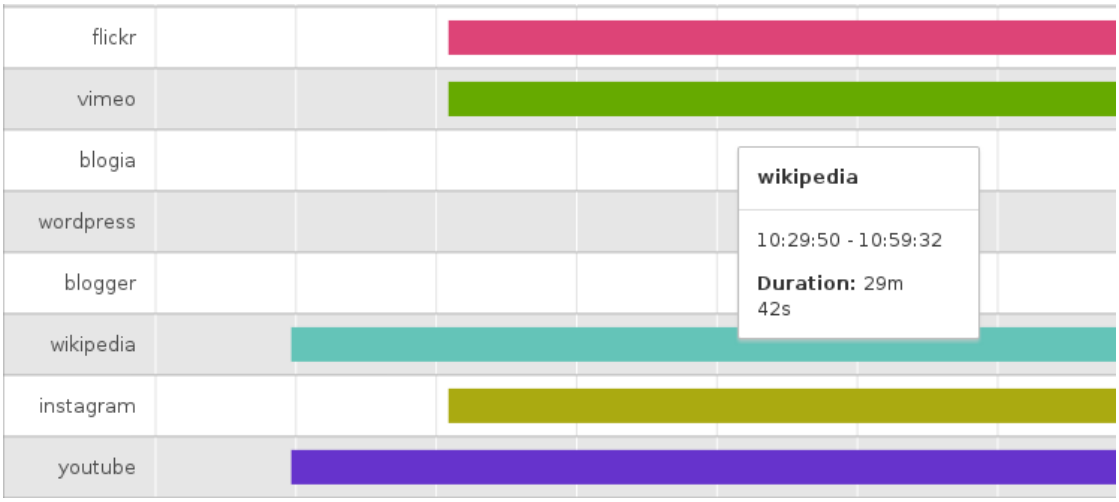
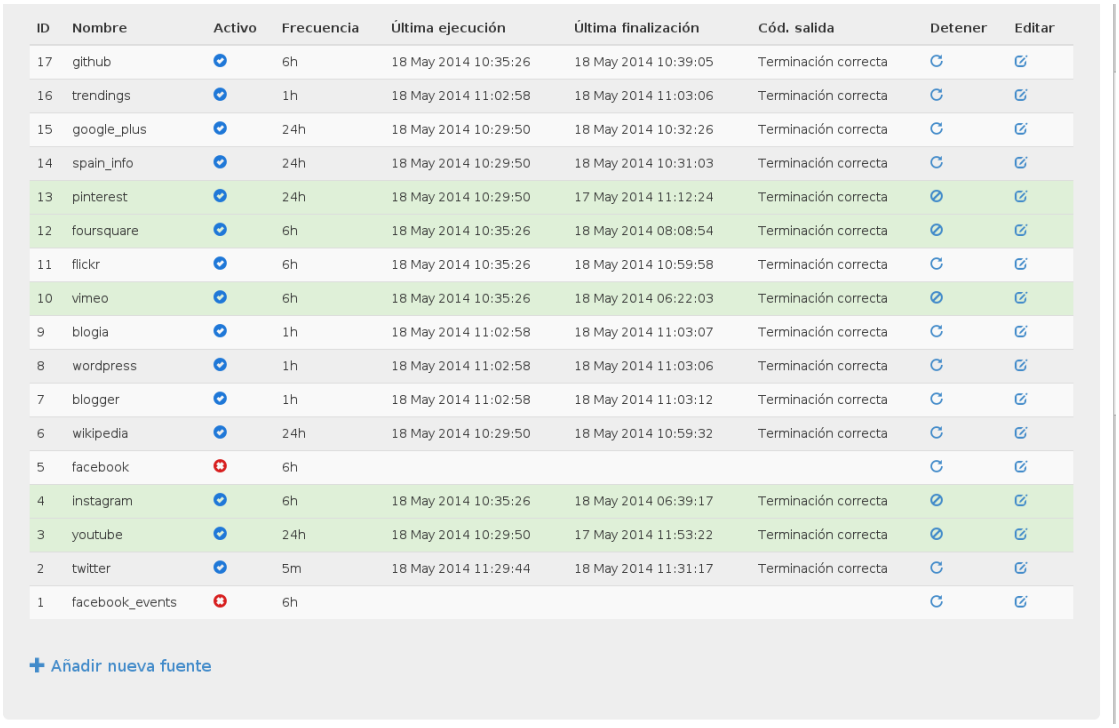


Figura F.9: Al situar el cursor sobre la línea de tiempo de la fuente *Wikipedia* vemos información adicional sobre su ejecución.

Un usuario *administrador* podrá realizar las siguientes operaciones con los *scra-*



ID	Nombre	Activo	Frecuencia	Última ejecución	Última finalización	Cód. salida	Detener	Editar
17	github	✓	6h	18 May 2014 10:35:26	18 May 2014 10:39:05	Terminación correcta	↻	✎
16	trendings	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:06	Terminación correcta	↻	✎
15	google_plus	✓	24h	18 May 2014 10:29:50	18 May 2014 10:32:26	Terminación correcta	↻	✎
14	spain_info	✓	24h	18 May 2014 10:29:50	18 May 2014 10:31:03	Terminación correcta	↻	✎
13	pinterest	✓	24h	18 May 2014 10:29:50	17 May 2014 11:12:24	Terminación correcta	⊘	✎
12	foursquare	✓	6h	18 May 2014 10:35:26	18 May 2014 08:08:54	Terminación correcta	⊘	✎
11	flickr	✓	6h	18 May 2014 10:35:26	18 May 2014 10:59:58	Terminación correcta	↻	✎
10	vimeo	✓	6h	18 May 2014 10:35:26	18 May 2014 06:22:03	Terminación correcta	⊘	✎
9	blogia	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:07	Terminación correcta	↻	✎
8	wordpress	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:06	Terminación correcta	↻	✎
7	blogger	✓	1h	18 May 2014 11:02:58	18 May 2014 11:03:12	Terminación correcta	↻	✎
6	wikipedia	✓	24h	18 May 2014 10:29:50	18 May 2014 10:59:32	Terminación correcta	↻	✎
5	facebook	✗	6h				↻	✎
4	instagram	✓	6h	18 May 2014 10:35:26	18 May 2014 06:39:17	Terminación correcta	⊘	✎
3	youtube	✓	24h	18 May 2014 10:29:50	17 May 2014 11:53:22	Terminación correcta	⊘	✎
2	twitter	✓	5m	18 May 2014 11:29:44	18 May 2014 11:31:17	Terminación correcta	↻	✎
1	facebook_events	✗	6h				↻	✎

+ Añadir nueva fuente

Figura F.10: Tabla con la información de los módulos de extracción de datos de las fuentes (*scrapers*) tal y como la ve un usuario *administrador*.

pers: añadir nuevos *scrapers*, editarlos, cambiar su estado de activación, deterlos y reiniciarlos.

Para modificar el estado de activación sólo debe pulsar sobre el icono correspondiente de la columna *Activo* (✓ o ✗).

Si un *scraper* está ejecutandose y, por algún motivo, desea detenerlo, debe pulsar sobre el icono ⊘ de la columna *Detener*.

Del mismo modo, si un *scraper* está detenido esperando su próxima ejecución (de acuerdo a la frecuencia con la que fue programado) y desea ejecutarlo saltándose esa espera debe pulsar sobre el icono ↻ de la columna *Detener*.

Para añadir un *scraper* debe pulsar el enlace *Añadir nueva fuente* de la parte inferior izquierda. Tras esto, se mostrará la pantalla de la figura F.11.

Se recomienda que esta acción sólo la realice el administrador encargado de dar soporte a la plataforma.

Figura F.11: Pantalla desde la que insertar un nuevo *scraper*.

Los datos a rellenar para una fuente son los siguientes:

Nombre Nombre de la fuente.

URL URL de la fuente.

Comando Comando a ejecutar para extraer datos de la fuente.

Tipo Tipo de *scraper*. Se distinguen dos tipos de *scrapers*:

Periódico El *scraper* se ejecutará periodicamente una vez que pase el tiempo indicado en *Frecuencia*.

Stream El *scraper* está continuamente escuchando datos de la fuente y, eventualmente, tras el tiempo indicado en *Frecuencia* los datos son volcados a la plataforma.

Frecuencia Frecuencia (en minutos) con la que el *scraper* se ejecuta.

Geolocalizable Indica si el contenido de la fuente aporta información geolocalizable o no.

Para editar una fuente agregada anteriormente, por ejemplo para modificar su frecuencia, se debe pulsar sobre el icono de la columna *Editar*. Tras ello se

mostrará una pantalla similar a la de la figura F.11 con los campos rellenados a partir de la información almacenada de la fuente.

F.3. Control de contenidos

Desde esta sección puede censurar, eliminar y bloquear contenido indeseable o la información publicada en las fuentes por usuarios que, voluntariamente, no deseen ser escuchados.

Para acceder a esta sección pulse sobre la pestaña *Control de contenidos* del menú de navegación superior (figura F.12).

Figura F.12: Acceso a la sección de control de contenidos.

El acceso a esta zona está restringido exclusivamente a usuarios *administradores*. Si un usuario *invitado* intenta acceder verá un mensaje como el de la figura F.13.

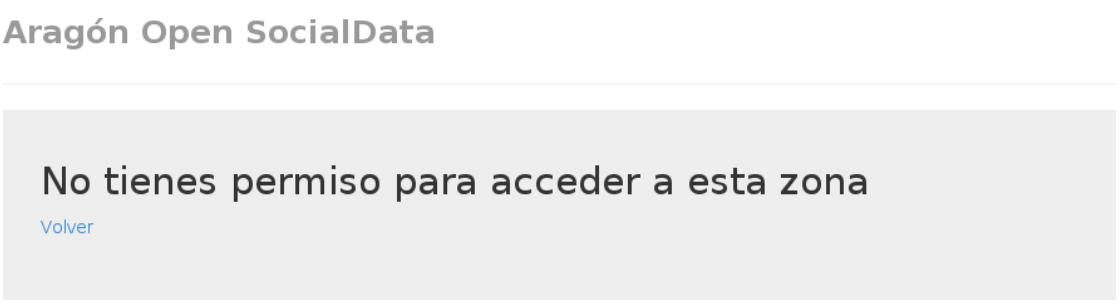


Figura F.13: Mensaje de error que obtiene un usuario *invitado* al intentar acceder a una zona de acceso restringido.

F.3.1. Control de contenidos por entrada

Puede eliminar contenido de dos maneras: eliminando entradas individuales conociendo la URL o eliminando toda la información recopilada de un usuario en una fuente de escucha.

En esta sección se explican los pasos a seguir para eliminar entradas conociendo su URL.

En primer lugar debe acceder a la zona de *Control de contenidos* como se indica en la figura F.12 y, a continuación, debe pulsar sobre la pestaña *Control de contenidos por entrada* del menú de navegación inferior (figura F.14).

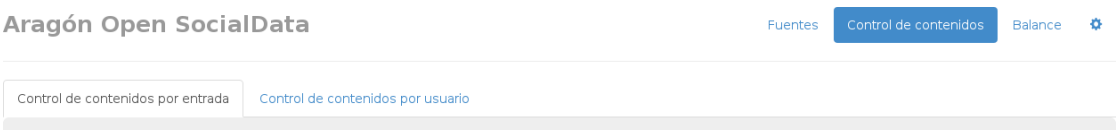


Figura F.14: Acceso a la sección de control de contenidos por entrada.

Esta sección se divide en dos zonas: la zona superior (figura F.15) le permite eliminar una entrada conociendo su URL y la zona inferor muestra de forma paginada las entradas que han sido eliminadas anteriormente (figura F.16).

Figura F.15: Eliminar entrada conociendo su URL.

La URL utilizada para eliminar una entrada es la que se obtiene al realizar una consulta a la API en el campo *url*.

La información mostrada de las entradas eliminadas es la siguiente:

ID Identificador interno de la entrada eliminada.

Figura F.16: Entradas eliminadas.

URL URL eliminada.

Fecha de bloqueo Fecha en la que la entrada fue eliminada.

Volver a escuchar Volver a escuchar la entrada pulsando sobre el icono .

¡ATENCIÓN! Si elimina una entrada desaparecerá por completo de los resultados de la API y no podrá recuperarse. Si una vez eliminado el contenido cancela el bloqueo y vuelve a escuchar el contenido, no se garantiza que vuelva a aparecer en los resultados de la API. Sólo volverá a mostrarse si algún *scraper* captura nuevamente el dato.

F.3.2. Control de contenidos por usuario

También puede eliminar contenido de usuarios indeseables o que, por voluntad propia, no deseen ser escuchados.

En primer lugar debe acceder a la zona de *Control de contenidos* como se indica en la figura F.12 y, a continuación, debe pulsar sobre la pestaña *Control de contenidos por entrada* del menú de navegación inferior (figura F.17).

Esta sección se divide en dos zonas: la zona superior (figura F.18) le permite eliminar las entradas de un usuario en una fuente y la zona inferor muestra de forma paginada los usuarios que han sido eliminados anteriormente (figura F.19).

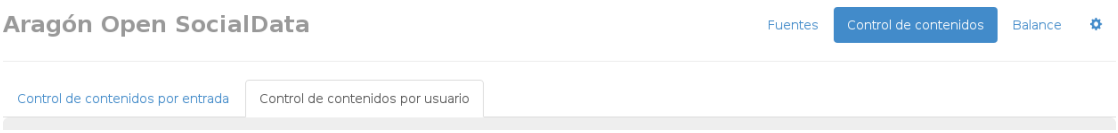


Figura F.17: Acceso a la sección de control de contenidos por usuario.

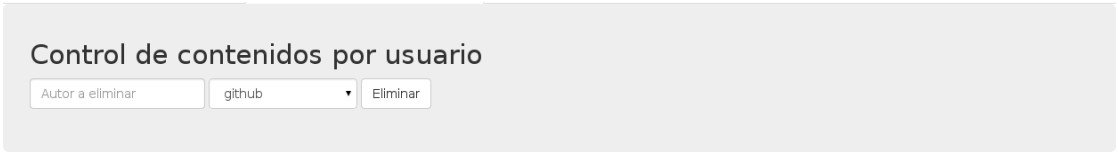


Figura F.18: Eliminar entradas de un usuario.

El nombre de usuario (autor) utilizado es el que se obteine al realizar una consulta a la API en el campo *author*.

Debido a que un mismo nombre de usuario puede existir en distintas fuentes de escucha, y puede estar asociado a individuos distintos, es necesario inidicar tanto el nombre del usuario como la fuente de la que se desea eliminar. Si se quiere eliminar un nombre de usuario de más de una fuente es necesario eliminarlo manualmente de todas las que se deseen.

Figura F.19: Usuarios eliminados.

La información mostrada de las entradas eliminadas es la siguiente:

ID Identificador interno del usuario eliminado.

Nombre Nombre del usuario eliminado.

Fuente Fuente de la que el usuario fue eliminado.

Fecha de eliminación Fecha en la que el usuario fue eliminado.

Volver a escuchar Volver a escuchar al usuario en esa fuente pulsando sobre el icono .

¡ATENCIÓN! Si elimina un usuario en una fuente desaparecerán por completo de los resultados de la API y no podrá recuperarse. Si una vez eliminado el contenido cancela el bloqueo y vuelve a escuchar el contenido, no se garantiza que vuelva a aparecer en los resultados de la API. Sólo volverá a mostrarse si algún *scraper* captura nuevamente el dato.

F.4. Balance y visualización de resultados

Desde esta sección puede visualizar de forma gráfica los resultados recogidos por los módulos de escucha de las fuentes, en función de distintos parámetros, y la evolución histórica del uso de la API.

Para acceder a ella pulse sobre la pestaña *Balance* del menú de navegación superior (figura F.20).

Figura F.20: Acceso a la sección de balance de la plataforma.

Existen cuatro modalidades de visualización disponibles:

- Visualizar los datos recopilados por fuente en forma de gráficos.
- Visualizar los datos recopilados por fuente en un mapa de calor.
- Visualizar la evolución de una palabra clave en los resultados de las fuentes.

- Monitorizar el uso del servicio (API).

Todas estas modalidades admiten filtrar los resultados en un rango de fechas como se ve en la figura F.21.

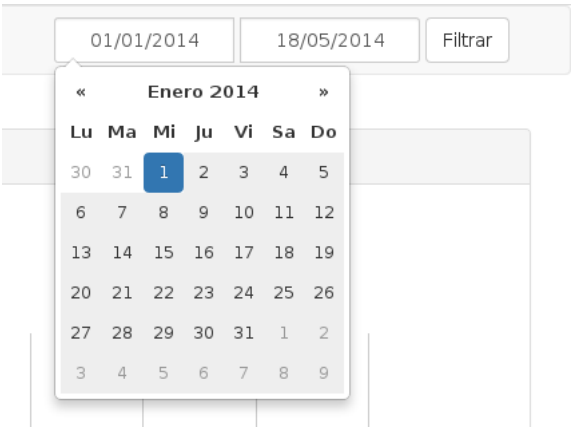


Figura F.21: Acceso a la sección de balance de la plataforma.

F.4.1. Visualización del estado de los datos recopilados

Esta sección muestra visualmente los datos recopilados por fuentes mediante gráficas y tablas.

Para acceder a ella, en primer lugar debe acceder a la zona de *Balance* como se indica en la figura F.20 y, a continuación, debe pulsar sobre la pestaña *Datos recopilados* del menú de navegación inferior (figura F.22).

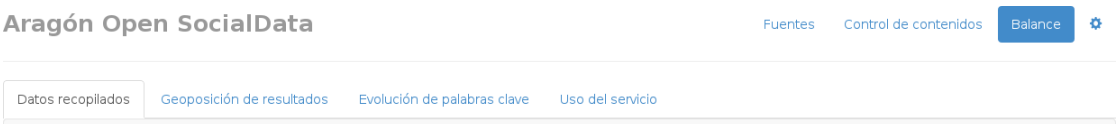


Figura F.22: Acceso a la sección de visualización de los datos recopilados en forma de gráficos.

Se puede visualizar la información de cuatro formas:

- Datos recopilados por fuente visualizados mediante un grafico circular tal y como se muestra en la figura F.23 arriba a la izquierda.
- Datos recopilados por fuente visualizados mediante un gráfico de barras tal y como se se muestra en la figura F.23 arriba a la derecha.
- Tendencias del momento en Aragón tal y como se muestra en la figura F.23 abajo a la izquierda.
- Resumen tabular de la información recopilada de las fuentes tal y como se muestra en la figura F.23 abajo a la derecha.

Las tendencias de Aragón se recogen periodicamente y de ellas se muestran los siguientes campos:

Trending Tendencia relevante dentro de la Comunidad Autónoma de Aragón.

Fecha Fecha en la que fue recogida esta tendencia por última vez.

Manual Indica si la tendencia fue añadida manualmente por un usuario *administrador* o fue agregada de forma orgánica por el sistema de escucha.

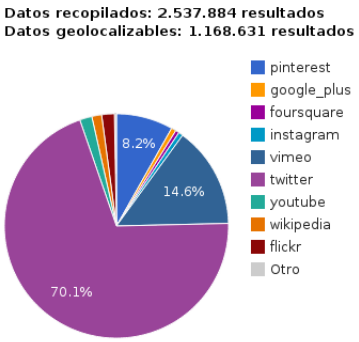
Si un usuario *administrador* accede a esta sección, además de todo lo anterior, en la zona de *Trendings* tiene la posibilidad de agregar nuevas tendencias manualmente y eliminar las ya existentes (figura F.24).

Las tendencias son una parte importante del sistema de escucha. Muchos *scrapers* se realimentan de ellas para extraer nueva información de las fuentes. Por este motivo puede ser interesante que, en algún momento, se desee recoger información relacionada con un tema especialmente relevante para la administración.

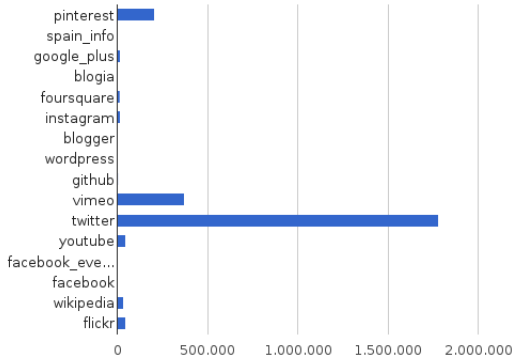
Una tendencia añadida manualmente permanecerá inalterable en el sistema hasta que un usuario *administrador* la elimine.

Las tendencias recogidas de forma orgánica se refrescan periódicamente (por defecto cada hora) eliminando las tendencias anteriores y agregando las nuevas. Una tendencia orgánica eliminada puede volver a ser añadida cuando el sistema de escucha vuelva a recoger las tendencias.

Datos recopilados por fuente



Total por fuente



Trendings

Trending	Fecha	Manual
Neptuno	18 May 2014 12:03:56	No
Vamos Zaragoza	18 May 2014 12:03:56	No
Sabadell	18 May 2014 12:03:56	No
Pinto	18 May 2014 12:03:56	No
Tata	18 May 2014 12:03:56	No
Diego Costa	18 May 2014 12:03:56	No
Tierno	18 May 2014 12:03:56	No
Romareda	18 May 2014 12:03:56	No
España	18 May 2014 12:03:56	No
Maccabi	18 May 2014 12:03:56	No

Resumen

Fuente	Entradas totales	Geolocalizable
twitter	1.778.384	Sí
vimeo	370.352	No
pinterest	207.842	No
flickr	46.447	Sí
youtube	44.748	Sí
wikipedia	35.935	Sí
google_plus	17.337	No
instagram	16.332	Sí
foursquare	13.279	Sí
github	3.517	No
facebook_events	1.595	Sí
facebook	1.154	No
wordpress	378	No
blogia	307	No
blogger	243	No
spain_info	34	Sí
Total geolocalizado	1.168.631	
Total	2.537.884	

Figura F.23: Datos recopilados por fuente y tendencias del momento en Aragón.

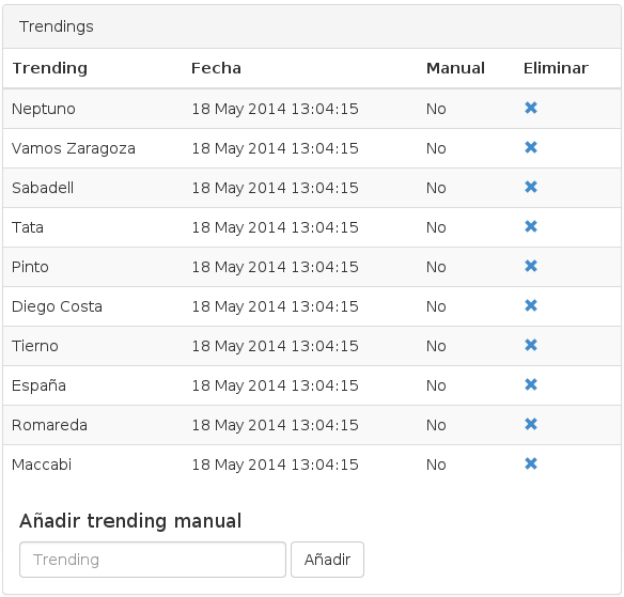


Figura F.24: Tendencias del momento en Aragón vistas por un usuario *administrador*.

F.4.2. Visualización geoposicionada de los datos recopilados

Esta sección muestra visualmente los datos recopilados por fuentes mediante mapas de calor.

Para acceder a ella, en primer lugar debe acceder a la zona de *Balance* como se indica en la figura F.20 y, a continuación, debe pulsar sobre la pestaña *Geoposición de resultados* del menú de navegación inferior (figura F.25).

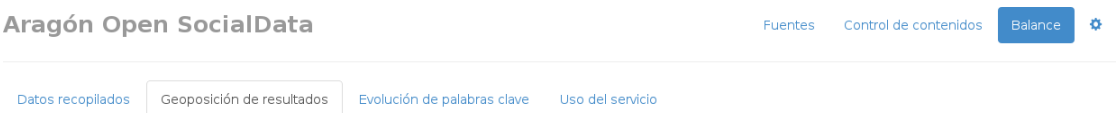


Figura F.25: Acceso a la sección de visualización de los datos recopilados en forma de mapas de calor.

Al acceder a esta sección verá una pantalla como la de la figura F.26. A mano izquierda encontrará el mapa de calor con el que puede interactuar y, a la derecha, un formulario mediante el cual puede realizar consultas.

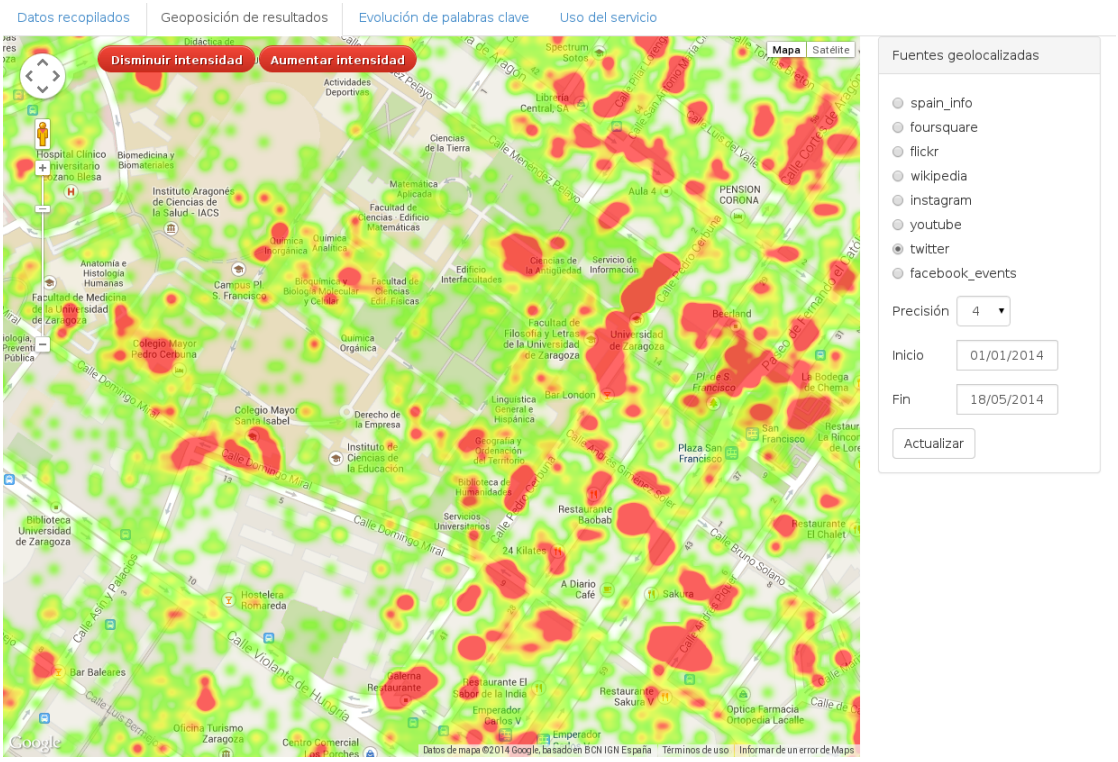


Figura F.26: Mapa de calor de los resultados de Twitter con nivel de precisión 4.

Puede realizar consultas filtrando los resultados por fuente, precisión y fecha. Se admiten consultas con cinco niveles de precisión ([0-4]), cada uno de ellos indica cuantos decimales son tomados en las coordenadas (latitud y longitud). Cuanto más alto sea este nivel más tiempo requerirá el sistema para procesar la petición.

Además, el mapa es completamente interactuable y posee dos botones en la parte superior izquierda que permiten variar el nivel de intensidad máxima del mapa de calor para refinar el resultado obtenido.

F.4.3. Evolución de palabras clave

Esta sección le permite observar la evolución de palabras clave contenidas en los datos recopilados de las fuentes de forma gráfica.

Para acceder a ella, en primer lugar debe acceder a la zona de *Balance* como se

indica en la figura F.20 y, a continuación, debe pulsar sobre la pestaña *Evolución de palabras clave* del menú de navegación inferior (figura F.27).

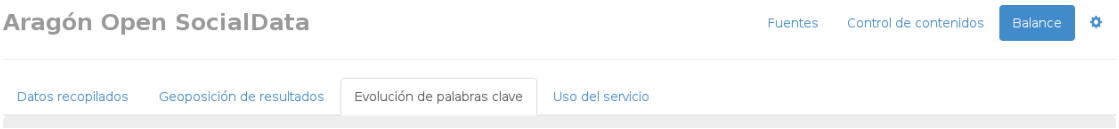


Figura F.27: Acceso a la sección de visualización de la evolución de palabras clave.

Desde aquí puede consultar qué repercusión ha tenido una palabra clave (o conjunto de ellas) en las redes sociales y los portales web escuchados. Los resultados se muestran en un gráfico como el de la figura F.28.

Figura F.28: Evolución de la palabra clave *elecciones* en las fuentes escuchadas entre el 1 de Enero de 2014 y el 18 de Mayo de 2014.

En caso de incluir más de una palabra clave la consulta se realizará utilizando

el operador lógico **AND** entre ellas.

F.4.4. Monitorización del uso del servicio (API)

Por último, esta sección le permite monitorizar el uso del servicio y visualizar la evolución histórica del mismo.

Para acceder a ella, en primer lugar debe acceder a la zona de *Balance* como se indica en la figura F.20 y, a continuación, debe pulsar sobre la pestaña *Uso del servicio* del menú de navegación inferior (figura F.29).

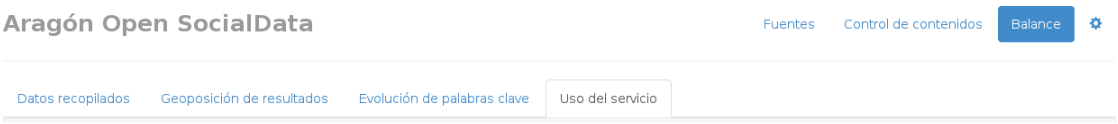


Figura F.29: Acceso a la sección de monitorización del uso del servicio.

Esta sección le permite monitorizar el uso de la API en tiempo real (más o menos aproximado en función del número de usuarios simultáneos que estén haciendo uso del servicio).

En la parte superior se muestra gráficamente el uso del servicio en función de tres parámetros (figura F.30): peticiones al *endpoint* de datos (azul), peticiones al *endpoint* de trendings (amarillo), peticiones errorenas (rojo). Se puede alejar y acercar el zoom del gráfico interactuando con el minigráfico situado debajo de él.

En la parte inferior se muestra información adicional sobre las peticiones: qué parámetros son los más frecuentes, cuántas peticiones han sido procesadas correctamente, cuál es el porcentaje de errores de la API, etc. (figura F.31).

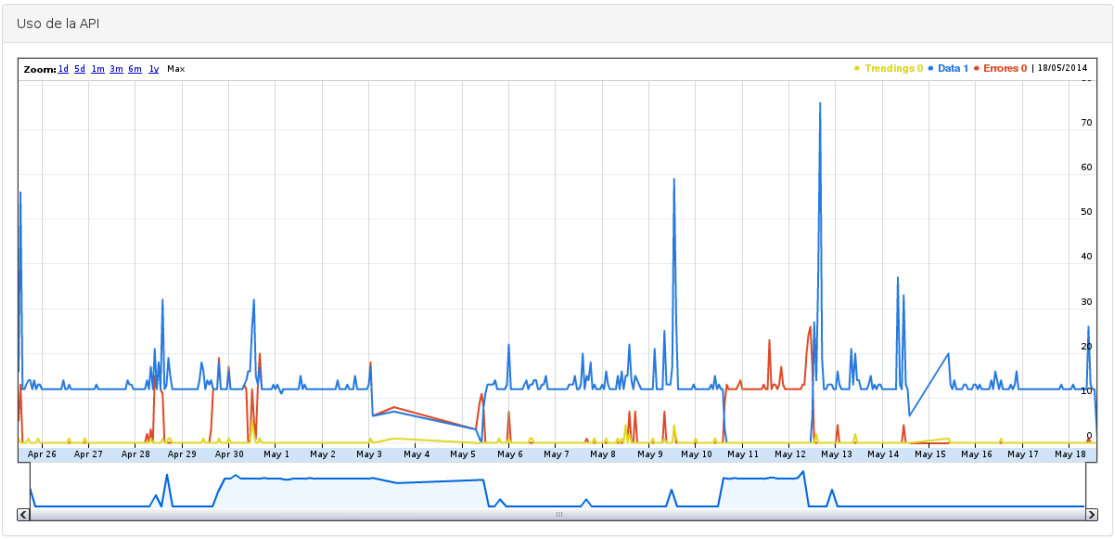
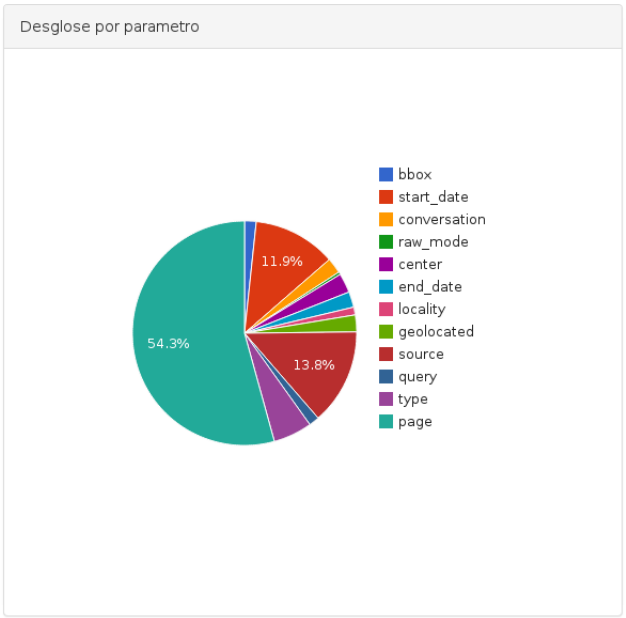


Figura F.30: Gráfica de uso del servicio.



Resumen

Tipo	Peticiones
page	291
source	74
start_date	64
type	30
center	15
geolocated	13
conversation	12
end_date	12
bbox	9
query	8
locality	6
raw_mode	2
Trendings	51
Data	5.855
Errores	1.786
Correctas	4.120
Porcentaje de errores	30,2404334575%

Figura F.31: Desglose del uso del servicio por parámetros.

F.5. Gestión de usuarios

Los usuarios *administradores* pueden gestionar qué usuarios acceden al sistema y qué rol ocupan (*invitados* o *administradores*).

Para acceder a la sección de gestión de usuarios debe dirigirse a la siguiente url:

`http://analyzer1.bifi.unizar.es:8080/admin/`

Una vez ingresadas sus credenciales accederá a una pantalla como la de la figura F.32.

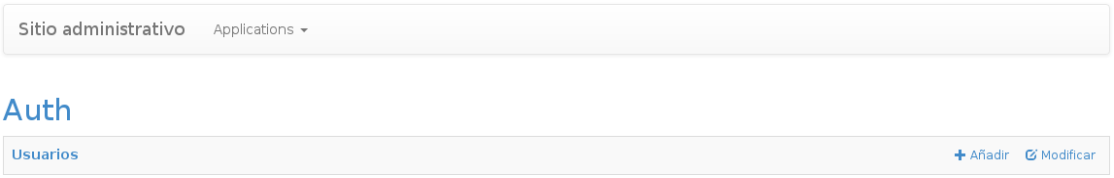


Figura F.32: Pantalla principal del sistema de gestión de usuarios.

Pulsando sobre el enlace *Usuarios* se accede a la lista de usuarios con acceso al sistema (figura F.33).

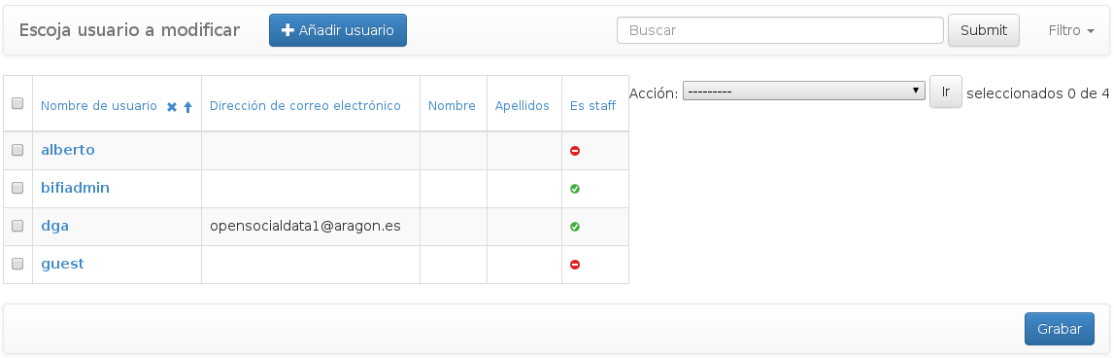


Figura F.33: Pantalla con la lista de usuarios autorizados para acceder a la interfaz de gestión y monitorización de Aragón Open SocialData.

Si se desea añadir un nuevo usuario se debe pulsar sobre el botón *Añadir* de la

derecha.

Para editar los permisos de un usuario, una vez que esté dado de alta, debe pulsar sobre el enlace *Usuarios* para listar todos los usuarios, pulsar sobre el nombre del usuario al cual desea modificarle los permisos y seleccionar los permisos adecuados como se ve en la figura F.34.

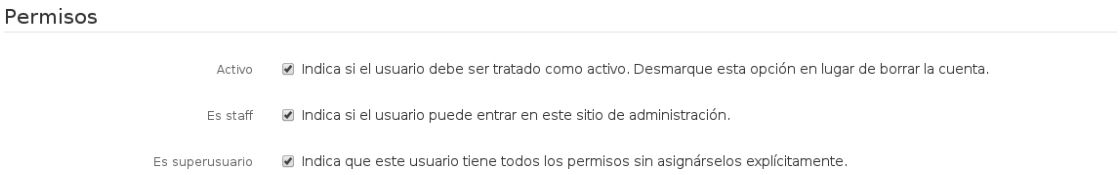


Figura F.34: Permisos disponibles.

Los permisos disponibles son los siguientes:

Activo Indica si el usuario debe ser tratado como activo. Desmarque esta opción en lugar de borrar la cuenta. Si está opción está desmarcada el usuario permanecerá almacenado en la base de datos pero no tendrá acceso a la interfaz.

Es staff Indica si el usuario puede entrar en este sitio de administración. Al marcar esta opción estamos indicando que el usuario tiene el rol de *administrador*. En caso contrario tendrá el rol de *invitado*.

Es superusuario Indica que este usuario tiene todos los permisos sin asignárselos explícitamente. Tiene el mismo efecto que la opción anterior, pero se aconseja marcar con *Es staff* a los usuarios con el rol de *administrador*.

Si desea modificar la contraseña de un usuario, por ejemplo para volverle a dar acceso al sistema si éste ha olvidado la suya, debe dirigirse a la ventana de edición de usuarios, como en el caso de anterior, y pulsar en el enlace de la figura F.35.

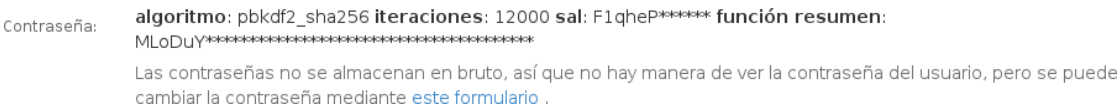


Figura F.35: Cambiar contraseña manualmente a un usuario.

Una vez realizados los cambios debe pulsar el botón *Grabar* de la parte inferior derecha de la página (el resto de opciones no se utilizan en esta plataforma).

Apéndice G

Manual de instalación y despligue

A lo largo de este manual se explican todos los pasos necesarios para instalar y desplegar la plataforma Aragón Open SocialData en un entorno de producción con dos nodos, uno dedicado a la escucha de las fuentes y otro dedicado a atender las peticiones de la API.

G.1. Prerrequisitos

Es necesario que estén instalados los siguientes paquetes en el sistema operativo de ambos nodos:

- python2.7
- python-pip
- python-virtualenv
- libxml2-dev
- libxslt1-dev
- libssl-dev
- libffi-dev
- libpq-dev

- python-dev
- apache2 con los módulos mod_wsgi y mod_evasive
- postgresql9.3
- postgis2.1

También es necesario que esté instalado Redis en el nodo 1 (maestro) y que dicho nodo tenga abiertos en el firewall los puertos de PostgreSQL y Redis, al menos con permiso para que el nodo 2 (esclavo) tenga acceso a ellos. También es necesario que ambos nodos tengan abiertos en el firewall los puertos correspondientes del servidor web Apache.

También es recomendable crear un usuario sin privilegios encargado de ejecutar la plataforma. A lo largo de este documento nos referiremos a él como *user*. Este usuario será propietario de un directorio en el que se copiarán todos los ficheros necesarios para el correcto funcionamiento de Aragón Open SocialData, por ejemplo `/home/user/escucha/`. Todos los comandos expuestos a continuación deben ejecutarse desde ese directorio.

G.2. Instalación

Los siguientes pasos, salvo que se indique lo contrarion, deben aplicarse en ambos nodos.

En primer lugar es necesario crear un entorno virtual de Python sobre el que instalaremos todas las dependencias con el sistema gestor de paquetes de Python *pip*.

```
virtualenv venv
```

El comando anterior creará el directorio venv (`/home/user/escucha/venv`) en el que se habrán copiado los principales ficheros ejecutables y librerías necesarias para lanzar un interprete de Python 2.7.

Gracias a este comando aislamos el intérprete de Python del resto del sistema, y él y todas las dependencias instaladas en ese entorno virtual, sólo pueden ser ejecutadas por el usuario *user*.

Activamos el entorno virtual recién creado e instalamos las dependencias de la plataforma usando pip:

```
source venv/bin/activate
pip install -r requirements.txt
```

En el fichero requirements.txt se encuentra la lista de paquetes Python requeridos por Aragón Open SocialDat (este fichero ha sido generado con el comando freeze de pip).

A continuación crearemos la base de datos *escucha*:

```
createdb escucha
psql escucha < tmp/migrations/escucha_schema_final.sql
```

Y crearemos el usuario administrador de la plataforma con el comando:

```
python launcher.py web syncdb
```

Este usuario será el administrador de la interfaz web y podrá crear nuevos usuarios desde ella.

Es necesario crear un directorio desde el que servir ficheros estáticos en el nodo 1 (maestro), nodo encargado de servir la interfaz web.

```
sudo mkdir -p /var/www/escucha/static
sudo chown -R user /var/www/escucha
```

A continuación, en ese nodo, recolectaremos los ficheros estáticos de la interfaz en el directorio recién creado:

```
python launcher.py web collectstatic
```

G.3. Configuración

Los pasos a seguir para la correcta configuración de Aragón Open SocialData son los siguientes:

El sistema de escucha de fuentes posee un sistema de reporte de errores por correo electrónico. Puede ser necesario modificar el fichero siguiente y actualizar la configuracion del servidor SMTP si es necesario:

```
escucha/conf/logger/logger_production.conf
```

Hay que configurar el servidor Apache con `mod_wsgi` en ambos nodos según lo indicado en el fichero `docs/apache_conf.txt`.

También, hay que configurar Celery en el nodo 1 (maestro) de acuerdo a lo indicado en el fichero `docs/celery/install_celery`.

A continuación es necesario recargar la configuración de Apache como super-usuario.

```
service apache2 reload
```

G.4. Interfaz por línea de comandos

Se incluye una interfaz en linea de comandos para el *daemon* y el control de *scrapers*.

Nota: sólo puede ejecutarse una única instancia del *daemon* a la vez. Para ello, al ejecutar el *daemon*, se crea de forma automática un fichero llamado `.daemon.pid` que almacena el *pid* del *daemon* en ejecución.

G.4.1. Datos iniciales

En primero lugar se debe inicializar la base de datos con la información sobre municipios del fichero municipios.xls. Para ello se utilizará el siguiente script:

```
python scripts/init_db.py
```

Muchos *scrapers* se realimentan de los datos de los trending topics. Para una correcta primera ejecución lanzamos el *scraper* de *trendings* en modo *standalone* (sin necesidad de ejecutar el *daemon*):

```
python scrapers/trendings.py
```

G.4.2. Daemon

Para lanzar el *daemon* que ejecuta los scripts con los *scrapers* (añadir `&` para que se ejecute en segundo plano):

```
python launcher.py rundaemon
```

Para lanzar el *daemon* en modo verbose (salidas de *log* se muestran por pantalla tambien):

```
python launcher.py rundaemon -v
```

Para detener el *daemon*:

```
python launcher.py stopdaemon
```

Para detener o reiniciar un *scraper* conociendo su id (identificador del *scraper* de la tabla sources de la DB):

```
python launcher.py kill id
python launcher.py restart id
```

G.4.3. Logs

La ejecución del *daemon* y los *scrapers* genera un *log* en el fichero `log/daemon.log`

Para filtrar los *logs* se incluye la herramienta watchlog:

```
python scripts/watchlog.py
```

G.5. Interface web

Los comandos del script manage.py de Django se han incorporado a launcher.py anteponiendo el parámetro *web*. Por ejemplo, para lanzar el servidor web de desarrollo de Django:

```
python launcher.py web runserver
```

G.6. Tests

Se utiliza *nose* como herramienta para ejecutar todos los tests unitarios y de integración de la plataforma. Nose busca todos los test del proyecto y los ejecuta:

```
nosetests [-v]
```

Para excluir los tests unitarios lentos (workers con sleep, interacción entre procesos, etc.):

```
nosetests -a '!slow' [-v]
```

Bibliografía

[1] P. Miller, R. Styles, and T. Heath, "Open Data Commons, a license for Open Data." *LDOW*, vol. 369, 2008.

[2] "How Open Data networks influence business performance and market structure," *Communications of the ACM*, vol. 39, no. 7, pp. 62–73, 1996.

[3] A. McAfee and E. Brynjolfsson, "Big data: the management revolution," *Harvard business review*, vol. 90, no. 10, pp. 60–68, 2012.

[4] S. Abdulhamid, S. Ahmad, V. Waziri, and F. Jibril, "Privacy and national security issues in social networks: The challenges," *International Journal of the Computer, the Internet and Management*, vol. 19, no. 3, pp. 14–20, September 2011.

[5] A. Lipp, R. Davis, R. Peter, and J. Davies, "The use of social media among health care professionals within an online postgraduate diabetes diploma course," *Practical Diabetes*, vol. 31, no. 1, pp. 14–17a, September 2013.

[6] A. Rodriguez, "Restful web services: The basics," *Online article in IBM DeveloperWorks Technical Library*, vol. 36, 2008.

[7] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, 2006.

[8] Y. Sugimori, K. Kusunoki, F. Cho, and S. Uchikawa, "Toyota production system and kanban system materialization of just-in-time and respect-for-human system," *The International Journal of Production Research*, vol. 15, no. 6, pp. 553–564, 1977.

[9] M. Ikonen, P. Kettunen, N. Oza, and P. Abrahamsson, "Exploring the sources of waste in kanban software development projects," in *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*. IEEE, 2010, pp. 376–381.

[10] J. James, "AJAX: A new approach to web applications," 2005, adaptive Path LLC.

[11] C. Wells, *Securing Ajax Applications: Ensuring the Safety of the Dynamic Web*. O'Reilly Media, Inc., 2007.

[12] "Official documentation of Redis," http://redis.io/documentation, May 2014.

[13] H. Kniberg and M. Skarin, *Kanban and Scrum - Making the Most of Both*, 1st ed. C4Media, 2010.

[14] "Official documentation of Django project," https://docs.djangoproject.com, May 2014.

[15] D. Thomas and A. Hunt, *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999.

[16] "Official documentation of Celery project," https://docs.celeryproject.org, May 2014.

[17] R. Fielding, U. Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," June 1999, rFC 2616.

[18] M. Turner, D. Budgen, and P. Brereton, "Turning software into a service." *Computer.*, vol. 36, no. 10, pp. 38–44, 2003.

[19] T. Berners-Lee, R. Fielding, U. Irvine, and L. Masinter, "Uniform resource identifiers (uri): Generic syntax," August 1998, rFC 2396.

[20] E. Hammer-Lahav, "The OAuth 1.0 protocol," April 2010, rFC 5849.

[21] E. Hardt, "The OAuth 2.0 authorization framework," October 2012, rFC 6749.

[22] "Architectural comparison and implementation of cloud tools and technologies," *International Journal of Future Computer and Communication*, vol. 3, no. 3, pp. 153–160, June 2014.

[23] R. Branson, "Celery messaging at scale at instagram," 2013, pyCon 2013.

[24] "Source code of rocksteady," https://code.google.com/p/rocksteady/, August 2010.

Todos los sitios web citados en este documento fueron accedidos por última vez el 23 de Mayo de 2014.