Plataforma web para análisis de datos

AGILE BARCELONA

Nino Dafonte

Memoria de proyecto final de grado
Universidad Oberta de Barcelona, Grado Multimedia

Área de especialidad: Desarrollo de aplicaciones interactivas
Consultor: *Kenneth Capseta Nieto* | Profesor: *Carlos Casado Martínez*

Mayo 2017

UOC
Universitat Oberta de Catalunya

Licencia

Reconocimiento-NoComercial-SinObraDerivada 3.0 España[1]

Usted es libre de:

- **Compartir** — copiar y redistribuir el material en cualquier medio o formato
- El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia.

Bajo las condiciones siguientes:

- **Reconocimiento** — Debe **reconocer adecuadamente** la autoría, proporcionar un enlace a la licencia e **indicar si se han realizado cambios<**. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.
- **NoComercial** — No puede utilizar el material para una **finalidad comercial**.
- **SinObraDerivada** — Si **remezcla, transforma o crea** a partir del material, no puede difundir el material modificado.
- **No hay restricciones adicionales** — No puede aplicar términos legales o **medidas tecnológicas** que legalmente restrinjan realizar aquello que la licencia permite.

[1] Extracto de la licencia original: https://creativecommons.org/licenses/by-nc-nd/3.0/es/

Agradecimientos

A mi pareja, Bea, que me ha apoyado desde el primer día, hasta llegar a donde nos encontramos hoy.

A mis padres, que me enseñaron el valor de las cosas y a no rendirse ante los obstáculos.

A la UOC y toda la ayuda recibida por parte de profesores, tutores, consultores, administrativos y un largo etc., por su apoyo durante esta etapa.

Abstract

El objetivo de esta herramienta es el de visualizar cómo la información de las redes sociales se propaga a través de sus usuarios, países y dispositivos.

Nos centraremos en un campo concreto (en nuestro ejemplo: las metodologías ágiles), en el que definiremos una serie de palabras clave que queremos observar en las redes.

A través de la captura y la gestión de todos los mensajes provenientes de Twitter que mencionen esas palabras, así como cualquier blog que trabaje o escriba algún post relacionado con ellas, seremos capaces de obtener un rastro verificable del flujo de la información.

Tendremos fechas y horas, lugares en el mapa y usuarios moviendo la información. A través de una serie de representaciones visuales y análisis estadístico, se espera descubrir cosas como:

- Quiénes son los iniciadores de tendencias.
- De dónde provienen esas tendencias.
- Qué países están a la cabeza de estos temas (en dónde se habla más sobre ellos).
- Y un largo etcétera.

Tanto el aspecto funcional como el tecnológico de este proyecto hacen que se pueda considerar un desarrollo en el que se reflejan los conocimientos adquiridos a lo largo de este Grado Multimedia, así como permite expandir y adquirir competencias adicionales orientadas al mundo profesional.

Como elemento adicional, se hace hincapié en el trasfondo de análisis y estadística que la aplicación provee, haciendo que la propia plataforma no sea solamente un fin (trabajo final de grado) sino un medio que promoverá un nuevo conocimiento, nuevos datos, basados en los resultados y las conclusiones que la misma generará.

Palabras clave:

Datos, análisis, visualización, redes sociales, trabajo de fin de grado, multimedia, sna, uoc.

Abstract (English version)

The project objective is to visualise the information travelling through the social networks and how this information lands into our very own devices.

This tool, based on a specific set of keywords, users, countries and devices, will be able to track all the information we need, store it and process it afterwards.

For testing purposes, we'll focus in a specific field (agile methodologies), using the Twitter social network and around 100 bloggers who write periodically about those methodologies (although the experiment could be applied to any set of keywords or fields we would like to study)

Once we have a big enough set of data, we'll be able to visualise that information using their timestamps, location information and users tweeting or re-tweeting the main keywords. We would be able to find out the answers to these questions and probably some others:

- Identify the people who kick-starts the trends.
- Where are those trends coming from?
- What countries are leading the discussion about those topics.
- Probably a lot more insights still to discover.

Considering both the functional and the technological aspects of the project lead us to believe that qualifies to demonstrate the knowledge and skills acquired through this multimedia degree, as well as allows an opportunity to expand and discover additional skills directly related with the professional development.

Additionally, it's worth to mention the analytical and statistical approach that the application is covering, as it would be a tool providing new data, which could be the basis for new studies and discoveries.

Keywords:

Data, analysis, visualization, social networks, memory, multimedia, sna, uoc.

Títulos previos a la introducción
Arial 11, código de color: Black, Text 1, Lighter 35%

Títulos de sección
Arial 16, código de color: Blue-Gray, Accent 1, Darker 25%

Título secundario (Título 2)
Arial 12, código de color: Blue-Gray, Accent 1, Darker 50%

Índice de Contenidos

INTRODUCCIÓN 10

DESCRIPCIÓN 11

OBTENCIÓN DE DATOS 11
VISUALIZACIÓN DE DATOS 11
TECNOLOGÍA Y ADECUACIÓN AL GRADO MULTIMEDIA 12

OBJETIVOS 13

PRINCIPALES 13
SECUNDARIOS 13

MARCO TEÓRICO 14

CONTENIDOS 15

METODOLOGÍA 16

ARQUITECTURA DE LA APLICACIÓN 18

SISTEMA DE ALMACENAMIENTO DE DATOS 18
SISTEMA DE COMUNICACIÓN DE DATOS 18
SISTEMA DE VISUALIZACIÓN (CLIENTE) 19

PLATAFORMA DE DESARROLLO 20

HARDWARE 20
SOFTWARE 20
OTROS 20

PLANIFICACIÓN 21

PROCESO DE TRABAJO 24

APIS UTILIZADAS 26

DIAGRAMAS DE OBJETOS UTILIZADOS 27

PROTOTIPOS 32

LO-FI 32
HI-FI 33

GUION 34

PERFIL DE USUARIO 35

PERFIL DE USUARIO ESTÁNDAR 35

USABILIDAD / UX 37

SEGURIDAD 38

TESTS 39

VERSIONES DE LA APLICACIÓN 40

REQUISITOS DE INSTALACIÓN 41

INSTRUCCIONES DE INSTALACIÓN 42

INSTRUCCIONES DE USO 43

BUGS 44

PROYECCIÓN A FUTURO 45

PRESUPUESTO 46

ANÁLISIS DE MERCADO 47

MARKETING Y VENTAS 48

CONCLUSIONES 50

ANEXO 1. ENTREGABLES DE PROYECTO 51

ANEXO 2. CÓDIGO FUENTE (EXTRACTOS) 53

ANEXO 3. LIBRERÍAS, CÓDIGO EXTERNO UTILIZADO 58

ANEXO 4. CAPTURAS DE PANTALLA 61

ANEXO 5. LIBRO DE ESTILO 62

LOGO 62
PALETA DE COLORES 62
TIPOGRAFÍA 62
DATOS ADICIONALES 62

ANEXO 6. RESUMEN EJECUTIVO 63

ANEXO 7. GLOSARIO 64

ANEXO 8. BIBLIOGRAFÍA 65

Figuras y tablas

Captura 1. Ejemplo de panel de control. 15
Captura 2. Ejemplo de organización de tareas en JIRA 17
Captura 3. Ejemplo de tareas a realizar. 17
Captura 4. Arquitectura a alto nivel de la herramienta. 19
Captura 5. Sprints iniciales. 22
Captura 6. Sprints 1 y 2. 22
Captura 7. Sprints 3 y 4. 23
Captura 8. Últimos Sprints y Backlog de tareas. 23
Captura 9. Prototipo Lo-Fi. 32
Captura 10. Prototipo Hi-Fi. 33
Captura 11. Landing page para acciones de marketing. https://agile.barcelona 49
Captura 12. Escuchando en la API de Twitter 53
Captura 13. Recibiendo y escribiendo los tweets en la base de datos. 54
Captura 14. El servidor NodeJS preparado para enviar los datos a través de la AP 55
Captura 15. El controlador de la vista, pidiendo los datos a la API 56
Captura 16. Detalle de las peticiones a la API desde la vista 57
Captura 17. Documento HTML en donde se muestran los datos al usuario 57
Captura 18. Resultado final en el navegador 57
Captura 19. Listado completo de librerías de Backend 58

Introducción

Desde que tengo recuerdos he estado vinculado de alguna manera al mundo de los datos y los sistemas informáticos.

Como programador siempre me ha obsesionado entender los detalles de los sistemas con los que trabajo. Observar los procesos, entender cómo fluye la información desde un punto hasta el siguiente de la aplicación. Ser capaz de comprender desde el detalle más pequeño, a la imagen global de lo que ocurre delante de mí.

Hace unos años que me dedico al mundo de las metodologías ágiles. Es decir, ayudo a individuos y empresas a buscar maneras alternativas de conseguir sus objetivos (cambiando o mejorando el cómo hacen las cosas, buscando nuevas opciones y/o estrategias de desarrollo, etc.).

Al igual que con la tecnología, cada día salen nuevas tendencias en este campo, nuevas herramientas, nuevas técnicas para mejorar, etc.

¿Cómo nos llega la información de estas nuevas tendencias?

¿Cuándo se habló de esto por primera vez?

¿Quién es el que empezó a trabajar o mencionar esta nueva técnica en nuestro país?

Ésta y otras incógnitas son las que intentaré despejar a través del proyecto que presento como trabajo final de grado.

A través de este desarrollo, crearemos un sistema para poder analizar los flujos de información a través de redes sociales y gestores de contenidos de acceso público, investigando el origen y la distribución geográfica de ciertos temas en concreto y observando su evolución a través de gráficos e infografías creadas por nuestra aplicación.

Si concluimos el proyecto de manera exitosa, podremos llegar a observar (de manera gráfica) ejemplos como:

La nueva herramienta “X”, se introdujo en Europa a través de un Tweet de alguien en Francia en diciembre de 2013, se expandió a España una semana después gracias a un usuario de Málaga, y acabó siendo tendencia en la conferencia ágil de Victoria de 2016.

Descripción

El desarrollo del proyecto, a nivel conceptual, se estructura en dos partes principales:

Obtención de datos

Creación de un sistema de gestión basado en la plataforma **Symfony** de PHP, versión 3.1, en dónde se definirán el tipo de palabras clave que queremos seguir y almacenar en nuestro sistema.

Una vez definidas, procesaremos datos externos con la intención de obtener contenido relacionado con esas palabras clave, procedentes de la **Streaming API** de **Twitter** y de fuentes **RSS** seleccionadas de unos **100 blogs** específicos sobre los temas ágiles.

La plataforma sobre la que se trabajará el almacenamiento de datos dependerá del tipo de datos en concreto: **MongoDB** para almacenar los datos recibidos desde la API de Twitter, los hilos de RSS y los datos procesados. **MySQL** para la gestión interna de la aplicación y los datos de usuarios.

Este sistema se encargará también de gestionar la frecuencia de consulta de estos datos y su procesamiento estadístico posterior. Tendrá formato de panel de control y será de acceso privado para administradores y usuarios de la plataforma.

Visualización de datos

La plataforma pública del sistema, se basa en 2 aspectos del mismo sistema:

- El proveedor de los datos: se creará una API dedicada, basada en la plataforma NodeJS y Express, que gestionaría el envío de datos, separando la parte de visualización de los datos en sí.

- El interfaz público: construido utilizando tecnologías web como AngularJS y Bootstrap entre otras.

A través de ellos, el usuario podrá llegar a visualizar el flujo de la información basada en palabras clave concretas. Se podría estudiar su transferencia geográfica, su propagación a través de la red de determinados usuarios, analizar sus redes desde el punto de vista de grafos y un largo etcétera de tratamientos estadísticos.

Tecnología y adecuación al Grado Multimedia

La plataforma, a nivel técnico, conforma un reto tecnológico en el que se integrarán:

- Lenguajes de programación múltiples y frameworks profesionales de desarrollo actuales (*Symfony 3 – PHP*, *NodeJS*, *Express* y *AngularJS*).
- Bases de datos relacionales (*MySQL*) y basadas en tecnologías *NoSql* (*MongoDB*).
- Maquetación y estilo de contenidos (*HTML5* y *CSS*).
- Diseño "*responsive"* y adaptado a diferentes dispositivos (basado en *Bootstrap*).
- Diseño de API para actualización de la plataforma de visualización utilizando tecnologías asíncronas (*AJAX*), seguridad y protección ante ataques web de carácter general (*XSS*, *CSRF*).

Objetivos

Principales

- Desarrollo de una aplicación web que permita la obtención de datos a través de APIs y RSS (fuentes externas) y almacene esa información para su utilización posterior.

- Desarrollo de una aplicación, basada en *NodeJS* / *Express*, y definición de API propia para comunicar con el portal público que mostrará los datos procesados.

- Desarrollo de un portal público y uno privado, utilizando tecnologías modernas de *frontend* para mejorar la interactividad y la usabilidad hacia el usuario final.

- Puesta en producción de ambas aplicaciones en un servidor real, accesible a través del dominio base "*agile.barcelona*", en la que se utilizarán técnicas de desarrollo continuo e integración continua.

- Utilización de metodologías ágiles para la realización del proyecto, basadas en objetivos por *sprint* (2 semanas de trabajo) y demostrables con entrega de valor real para el usuario al final de cada uno de los mismos.

Secundarios

- Diseñar la aplicación pública de manera que permita la consulta de datos para diferentes tipos de audiencia:

    - Usuarios no autentificados: Consulta de datos con un granulado y un nivel de detalle mensual.

    - Usuarios autentificados: de carácter gratuito, pero con registro (*email/password*). Proporcionará el valor añadido de poder consultar los datos a un nivel de detalle más alto (diario o incluso en tiempo real).

    - Usuarios de pago: Tendrían acceso a una parte del panel de administración, en la que podrían personalizar las palabras clave o términos de búsqueda, y que sólo ellos podrían consultar esos resultados.

Los objetivos secundarios se plantean como una posible continuación del proyecto de cara a su expansión y monetización futura. En ningún caso entran dentro de la planificación ni del alcance de este trabajo final de grado.

Marco teórico

Hace 30 años el problema con la información era lo difícil en cuanto a su obtención y proceso. Internet empezaba a dar sus primeros pasos, las universidades ya tenían cierto acceso a información compartida, pero para un usuario anónimo era un hito casi imposible de realizar. Se utilizaban procesos manuales, encuestas, introducción manual de datos en bases de datos, etc.

Además de lo rudimentario en cuanto a su obtención, la caducidad de esos datos era alta (se quedaban obsoletos muy rápidamente) y no se podían analizar a tiempo real.

En el momento actual, el problema es la cantidad excesiva de información que tenemos que gestionar. La explosión del uso de internet para particulares, las redes sociales, blogueros, revistas online, además de todos los canales tradicionales, hacen que el reto ahora sea cómo gestionar y catalogar esa información.

El proyecto sobre el que se basa esta memoria trata de dar un paso hacia la solución a nuestro problema actual. Ser capaces de seleccionar la información sobre la que queremos trabajar, eliminar toda la que no nos sirva y catalogar lo que nos sea útil de una manera en la que podamos manipularla.

Para ir un paso más allá en cuanto a la clasificación y catalogado, el proyecto aportará información de carácter visual (gráficas e infografías), basadas en los datos obtenidos para facilitar todavía más su tratamiento y comprensión posterior.

A día de hoy existen proyectos similares, a nivel profesional, para el seguimiento de imagen de marca, de eventos y de detección de sucesos (Twitter ha probado ser una herramienta muy interesante a la hora de seguir catástrofes naturales y eventos deportivos mundiales), aunque no se ha llegado a aplicar a seguimiento de tendencias a nivel público, al menos en nuestro país.

Desde que el concepto de "*Big Data*" apareció, se ha escrito mucho sobre análisis de redes sociales, procesamiento de grafos y de conceptos de redes como centralidad, distancia, iniciadores de los mensajes, seguidores, etc.

Durante el transcurso de este proyecto, desarrollaremos éstos y otros conceptos aplicados a un campo específico de la gestión de proyectos y productos: las metodologías ágiles.

Contenidos

Dejando a un lado el contenido propio de la arquitectura de obtención de datos (desarrollada más en detalle en el apartado de Arquitectura de la aplicación) la aplicación pública constará de un formato de "Aplicación de una sola página" o SPA[2].

Tendrá la apariencia de un cuadro de mandos o panel de control, con un menú lateral izquierdo, en el que se podrá seleccionar la categoría de datos que deseamos consultar (datos por países, por tiempo, análisis específicos, palabras clave, etc.).

Una vez seleccionada la sección en concreto, la parte derecha de la pantalla estará formada por gráficos y tablas de agrupaciones de datos. Se desarrollarán elementos con entidad propia (Widgets[3]) que proporcionarán al usuario información y resultados analizados sobre relaciones y agrupaciones de datos.

Se plantea también tener una sección específica para datos en tiempo real, en el que podamos observar determinados eventos de interés para la comunidad ágil y cómo la información se distribuye durante el mismo.

Tanto el diseño como la estructuración de los contenidos seguirá un patrón minimalista, en el que lo importante será la clara representación de los datos y la selección de los mismos para su análisis.

Captura 1. Ejemplo de panel de control.

[2] Single Page Application : https://es.wikipedia.org/wiki/Single-page_application

[3] Widgets : https://es.wikipedia.org/wiki/Widget

Metodología

Como ya se ha ido introduciendo en otros puntos, la metodología de desarrollo estará marcada por las **metodologías ágiles**.

El trabajo se planificará, desarrollará y se pondrá en funcionamiento en **ciclos de 2 semanas**, denominados *Sprints*.

En cada Sprint, se hará una planificación de qué es posible llevar a cabo durante esas 2 próximas semanas. La estimación se realizará teniendo en cuenta el conocimiento adquirido en anteriores *Sprints*, las incógnitas sobre la tecnología a utilizar y la complejidad de las tareas que valoremos.

Existirá un *Backlog*, **una lista priorizada de tareas** a realizar. Sobre ellas, se identificarán las que sea posible atacar en las próximas 2 semanas y se establece el objetivo del *Sprint*. Se trabaja durante ese tiempo y, al finalizar el *Sprint*, se hará una **demostración** (con el consultor, si está disponible) y una **retrospectiva**.

La demostración (*Sprint Review*) tiene como objetivo recibir ***feedback*** del consultor sobre lo desarrollado, tener un espacio para visualizar la realidad del trabajo realizado y decidir cuáles serían los siguientes pasos (las siguientes tareas del *Backlog*) a trabajar.

La **retrospectiva** (*Sprint Retrospective*), tiene como finalidad hacer una reflexión del cómo se ha realizado el trabajo. ¿La planificación ha sido adecuada? ¿Las herramientas han sido las adecuadas? ¿Hay algo que se podría hacer diferente para poder entregar más o mejor? ¿Hay algo que se debería dejar/empezar a hacer?

Con lo aprendido durante esas 2 semanas, se repite el **proceso de manera iterativa**.

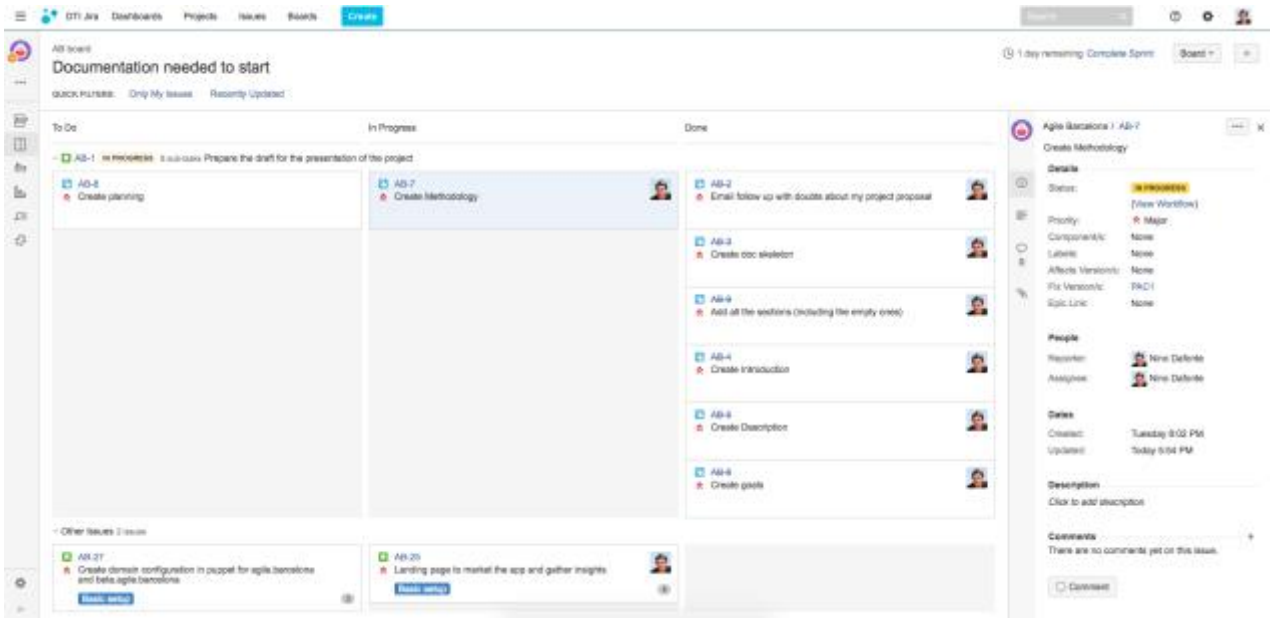
En cada *Sprint*, el producto va creciendo de manera **incremental**, acercándose a cómo se imagina en un estado final, incorporando los nuevos conocimientos (tanto de las tecnologías utilizadas como del mercado al que está orientado y los **usuarios finales**).

La justificación del uso de esta metodología se basa en que, para productos con un nivel de incertidumbre alto, el trabajo iterativo genera conocimiento en cada ciclo, ayudando a conseguir un producto mucho más cercano a las necesidades del usuario final, sin la inversión y suposiciones que un proyecto basado en metodologías tradicionales asume (meses de análisis, meses de ejecución, para finalmente llegar al mercado con un producto que los usuarios no han visto hasta el día de la puesta en producción).

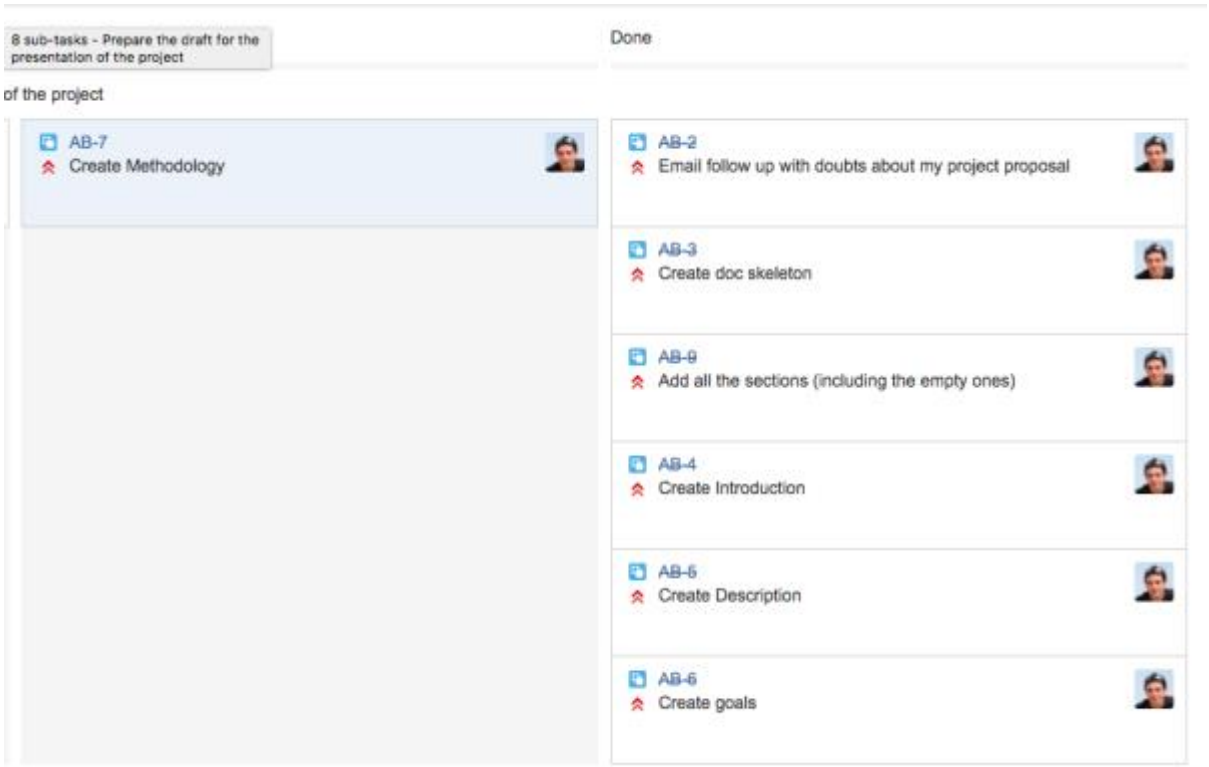
Como soporte técnico para la metodología, se utilizará la herramienta **JIRA**, de la empresa *Atlassian*, conocida y utilizada por muchas empresas para la gestión de sus productos que, además, soporta de manera nativa el uso de *Backlogs* y *Sprints* en el marco de trabajo de *Scrum*[4].

[4] Scrum: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

Ejemplos del contenido de un Sprint al inicio del proyecto en JIRA:



Captura 2. Ejemplo de organización de tareas en JIRA



Captura 3. Ejemplo de tareas a realizar.

Arquitectura de la aplicación

El sistema técnico de la aplicación se distribuye en 3 módulos principales:

- Dos sistemas en el lado del servidor ("*Backends*")
- Un sistema de visualización de la información ("*Frontend*" o cliente)

Sistema de almacenamiento de datos

Aplicación basada en Symfony Framework 3.x, conocido marco de trabajo para desarrollo de aplicaciones en PHP. Sus objetivos principales son:

- Gestión de trabajos en segundo plano para la obtención y almacenamiento de datos.

- Gestión de usuarios y accesos a la administración.

- Administración de los contenidos a examinar (qué palabras clave estamos procesando, cada cuanto tiempo, etc.).

- Configuración del sistema de obtención de datos.

Se apoyará en 2 motores de bases de datos, seleccionados en base a las características de los datos a almacenar:

- MongoDB (v3.4.2): base de datos NoSQL, que sacrifica algunas opciones de las bases de datos relacionales (claves y relaciones entre tablas, por ejemplo), a cambio de no requerir modelos estructurados y proporcionar una velocidad de inserción y proceso superiores (pensada para almacenar la información de Twitter en tiempo real).

- MySQL (v5.7.16): base de datos relacional. En ella se almacenarán los datos procedentes de los blogs y otras fuentes de datos. Las funcionalidades de una base de datos relacional como la estructuración y la organización de los datos, en este caso, son un beneficio extra para su posterior proceso.

Sistema de comunicación de datos

Esta es la parte del sistema que se encargará de recuperar los datos de la base de datos, aplicar la lógica de negocio necesaria y comunicar los datos a los clientes a través de un interfaz de comunicación estándar de programación (una API).

El objetivo es el de aislar los datos completamente de la interfaz, de manera que implementar una versión móvil en algún momento futuro, sólo implica generar el cómo mostrar los datos, en lugar de rehacer partes de la aplicación.

Para este sistema se utilizará un servidor creado en NodeJS (v7.7.3) y ExpressJS (v4.15.2).

Sistema de visualización (cliente)

La aplicación cliente, como ya hemos mencionado en previos apartados, se compone de una estructura de aplicación de página única (SPA), basada en tecnologías Javascript (AngularJS v1.5.x) y apoyada en marcos de trabajo y diseño para múltiples dispositivos ("*responsive design*") como Bootstrap 3.x.

Captura 4. Arquitectura a alto nivel de la herramienta.

Plataforma de desarrollo

El desarrollo de esta herramienta se realizará íntegramente con un portátil con sistema Mac OSX. Los requisitos para su desarrollo no son demasiado altos y sólo requiere tener bien configurado el entorno de desarrollo y sus herramientas.

Al ser un sistema Mac, basado en Linux, todas las herramientas de desarrollo (la mayoría de código abierto) están disponibles en formato nativo.

Hardware

MacBook Pro (Retina, Mediados de 2012)
Procesador: 2.3 GHz Intel Core i7
Memoria: 8 GB 1600 MHz DDR3
Espacio en disco: 256 GB SSD.
Tarjeta gráfica: NVIDIA GeForce GT 650M 1024MB
Monitor adicional: Philips 236V (23 pulgadas).

Software

Editor de código fuente: Intellij IDEA 2016.3
Servidor web: Apache 2.4
Bases de datos: MySQL y MongoDB
Entornos de programación: PHP, NodeJS
Herramientas multimedia: Adobe Creative Cloud
Herramientas ofimáticas: Microsoft Office 2016 (v15.x)
Navegadores: Chrome y Firefox

Otros

Herramientas y aplicaciones web de apoyo al desarrollo:

Atlassian JIRA (gestión de tareas)
Atlassian Bamboo (para despliegues e integración continua)

Servidor de producción:

Hetzner EX40, servidor dedicado con 32 GB de RAM y 2 TB de disco, situado en Alemania.

Planificación

Para la planificación utilizaremos un *Backlog* de tareas priorizadas y una agrupación en *Sprints*. Un *Backlog* se compone de tareas con diferentes niveles de especificación: cuanto más arriba se encuentre una tarea (más inminente su desarrollo), poseerá un nivel más alto de especificación (estará más claro el qué hay que desarrollar).

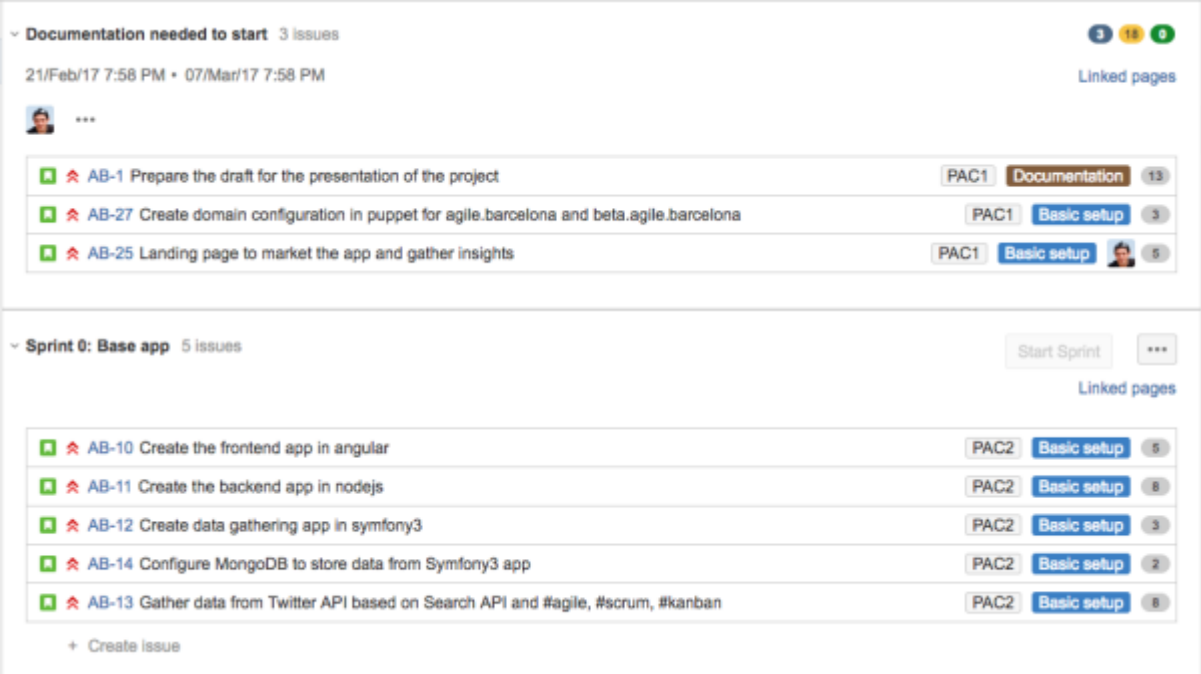
Por este motivo, un *Backlog* es un elemento vivo, en el cual se realiza un mantenimiento, añadiendo tareas nuevas que han surgido o eliminando tareas que no se consideran necesarias (debido al aprendizaje continuo sobre la aplicación, su tecnología y sus usuarios).

A nivel planificación temporal, las fechas destacadas se corresponden con cada *Sprint* de desarrollo, coincidiendo la fecha de entrega de las PACs con cada 2 *Sprints* (aproximadamente).

Sprint	Fechas	Objetivo	Tareas
Documentación	21 feb – 07 mar	PAC1 – Memoria	Definición inicial del documento de la memoria y puesta en producción de la página de descripción del proyecto con fines de marketing y prospección de mercado.
Sprint 0	08 mar – 22 mar	Aplicación base	Creación de las aplicaciones base y obtención de los primeros datos reales.
Sprint 1	23 mar – 06 abr	PAC2 y prueba de concepto	Entrega de los avances en la memoria y prueba de concepto de la aplicación (un gráfico mostrado en el *frontend*, recuperando datos reales desde el *backend*, cerrando el círculo y creando la base para el resto de las funcionalidades. Además de esto, incluirá la configuración del sistema de integración y despliegue continuo para las aplicaciones.
Sprint 2	07 abr – 21 abr	Panel de control y administración de usuarios.	Puesta en marcha del panel de control y gestión de usuarios para modificación de los datos a obtener por la aplicación de *backend*.
Sprint 3	22 abr – 06 mayo	PAC3 y más gráficos	Avances con la memoria y desarrollo de más gráficos y análisis de datos provenientes de la base de datos de Twitter.
		Incluimos los RSS de los blogs	Obtención de datos de nuestra segunda fuente (Blogs) en la base de datos local.
Sprint 4	07 mayo – 21 mayo	Entrega final y *tagclouds* para mostrar análisis de blogs.	Finalización de la documentación y ajustes de la parte pública para mostrar nubes de tags sobre temáticas en los blogs.
Sprint 5	22 mayo – 05 jun	Revisión y entrega	Últimos ajustes a la memoria y a la aplicación para su entrega definitiva.

En la herramienta de gestión del producto (JIRA) se pueden observar los detalles de cada Sprint (al menos de los 2-3 más inmediatos), así como su estimación de esfuerzo y fechas estimadas en la planificación (es posible asignar un acceso al consultor para un trabajo conjunto más cercano).

En las siguientes imágenes se puede ver una muestra de los *Sprints* más próximos, así como el contenido del *Backlog*:



Captura 5. Sprints iniciales.

Captura 6. Sprints 1 y 2.

Sprint 3: More charts 2 issues

Linked pages

AB-28 Third delivery for the project documentation PAC3 Documentation

AB-29 Additional charts for phase 1 of the public frontend PAC3 Data Visualization

+ Create issue

2 issues Estimate 0

Sprint 4: RSS gathering 2 issues

Linked pages

AB-30 RSS listing and data collection for backend Entrega Final Data Gathering

AB-32 RSS API endpoints in nodejs app Entrega Final Data Visualization

+ Create issue

2 issues Estimate 0

Captura 7. Sprints 3 y 4.

Sprint 5: RSS visualization 2 issues

Linked pages

AB-31 RSS tagcloud generation for public frontend Entrega Final Data Visualization

AB-33 Final delivery for the project's documentation Entrega Final Documentation

+ Create issue

2 issues Estimate 0

Sprint 6: Delivery and checks 2 issues

Linked pages

AB-34 Final review of the project's documentation and delivery Entrega Final Documentation

AB-35 Final review of the architecture and system's functionality Entrega Final

+ Create issue

2 issues Estimate 0

Backlog 4 issues Create Sprint

AB-36 Analytics events to learn about user's activity

AB-37 Caching of the results to improve performance (analysis and map-reduce processes)

AB-38 Introduce layer of logged vs not logged users (chart details from month to day or even real time)

AB-39 Introduce paying customers (custom definition for topics and reporting)

+ Create issue

Captura 8. Últimos Sprints y Backlog de tareas.

Como se ha mencionado anteriormente, es bastante probable que el contenido exacto de cada *Sprint* y del *Backlog* en sí mismo varíe con cada iteración, ya que tendremos un mejor conocimiento de las tecnologías y la complejidad de cada tarea.

Proceso de trabajo

El proyecto comienza con una conceptualización general del proyecto y de sus componentes arquitectónicos a alto nivel.

Una vez analizados los riesgos propios de un desarrollo de carácter innovador (es decir, que no es algo que se pueda desarrollar con una lista de pasos, sin opción de interpretación o descubrimiento de nuevo conocimiento), se desarrolla un plan para eliminar (o minimizar) esos riesgos de manera anticipada.

El desarrollo iterativo consta de realizar una versión mínima de la aplicación (una funcionalidad mínima) pero que utilice todos los componentes de la arquitectura, eliminando así los riesgos de selección de tecnologías y de integración.

Se empieza a desarrollar cada uno de los componentes en sus versiones mínimas:

- El componente de recolección de datos: sin configuración de palabras clave ni usuarios. Se añaden las palabras clave en un fichero de configuración y se programa el sistema de obtención de datos sin acceso a través de ningún tipo de interfaz (sólo a través de la consola y el código).

- El componente de transmisión de datos (nuestra API): un servidor con funcionalidades mínimas, que es capaz de transmitir datos preestablecidos (sin conexión a base de datos todavía).

- El componente de visualización de datos: Diseño mínimo y datos estáticos (todavía no obtenidos a través de la API).

Una vez que está todo en funcionamiento se crean los procesos de integración de componentes y ejecución de pruebas previas al despliegue en el servidor. También se configuran los procesos de despliegue y se comprueba su correcto funcionamiento una vez puesto online (todavía protegido con contraseñas gestionadas por el servidor web, para evitar el acceso).

Superado este paso, el riesgo se ha reducido dramáticamente. Todas las tecnologías han probado ser útiles y la aplicación ya está en producción.

Se activan los procesos de captación de datos en producción y las bases de datos comienzan a almacenar la base de nuestro estudio.

A partir de aquí, de manera iterativa, se hace crecer la aplicación en cuanto a funcionalidades:

- Una iteración para añadir la integración de la API y las bases de datos, así como mostrar datos reales en los clientes de visualización.

- Iteración para añadir la gestión de usuarios y permisos en el "*backend*".

- Iteración para añadir el proceso de Blogs (en un primer paso solo se procesaba Twitter.

- Iteración para incrementar el número de bloques que la aplicación muestra (más gráficos, más secciones, más análisis de los datos obtenidos).

El proceso continúa de esta manera añadiendo funcionalidades en base a las necesidades de los usuarios (que son analizadas cada 2 semanas), incluyendo su feedback sobre la aplicación actual y hacia dónde debería dirigirse.

APIs utilizadas

En el desarrollo del proyecto se utilizan las APIs internas propias de los *frameworks* utilizados (Symfony 3, NodeJS, ExpressJS, AngularJS, etc.) así como APIs de terceros, detalladas en la siguiente lista:

- Search API de Twitter: para búsquedas concretas (seguidores de un usuario en particular, su lista de tweets, etc.)

- Streaming API de Twitter: Canal abierto en el cual Twitter proporciona el 1% de todo el tráfico mundial. Se sirve en tiempo real y hay que mantenerlo a la escucha en todo momento. Con un filtro proporcionado y añadido al canal de escucha, podemos decidir qué tipo de información queremos que la API nos envíe.

Durante el desarrollo de este proyecto también crearemos nuestra propia API, a través de la que los clientes obtendrán los datos de nuestros análisis. Es viable que esta API pueda convertirse en pública para que otros usuarios puedan utilizarla de manera programática.

Así mismo, el número de APIs internas que manejaremos en el proyecto es enorme, si consideramos cada uno de los paquetes y dependencias que cada uno de los *frameworks* incluye para determinadas funcionalidades. El nombrar todas y cada una de ellas queda fuera del alcance de este apartado, pero pueden consultarse en los archivos de dependencias del repositorio de código fuente.

Diagramas de objetos utilizados

El principal objeto que manejaremos en nuestra aplicación son los tweets recibidos a través de la API de Twitter. Compondrán la base de nuestros análisis y, por norma general, tienen la siguiente estructura:


```
{
        "_id" : ObjectId("58d261493902f7ae580041a7"),
        "created_at" : "Wed Mar 22 11:34:31 +0000 2017",
        "id" : NumberLong("844512612356370433"),
        "id_str" : "844512612356370433",
        "text" : "RT @Tidal_Shift: Agile Organisational Change Management: Fashion or Future? via @APMG_Inter #Change
#OrgCulture #CSuite #Agile",
        "source" : "<a href=\"http://dustcloud.io\" rel=\"nofollow\">Dustcloud.io</a>",
        "truncated" : false,
        "in_reply_to_status_id" : null,
        "in_reply_to_status_id_str" : null,
        "in_reply_to_user_id" : null,
        "in_reply_to_user_id_str" : null,
        "in_reply_to_screen_name" : null,
        "user" : {},
        "geo" : null,
        "coordinates" : null,
        "place" : null,
        "contributors" : null,
        "retweeted_status" : {},
        "is_quote_status" : false,
        "retweet_count" : NumberLong(0),
        "favorite_count" : NumberLong(0),
        "entities" : {},
        "favorited" : false,
        "retweeted" : false,
        "filter_level" : "low",
        "lang" : "en",
        "timestamp_ms" : "1490182471728"
}
```


(Se han eliminado ciertas partes internas de la estructura para mejorar su legibilidad en este apartado).

La estructura no es estándar al 100%. Es decir, el número de campos y su composición puede ser variable dependiendo de las características del mensaje.

A partir de este objeto se derivarán todos los análisis de la aplicación, utilizando tecnologías derivadas de MapReduce que MongoDB trae incorporadas por defecto.

Además de los tweets, se utilizan otros objetos para el tratamiento de datos. Como hablamos de una base de datos documental (no relacional), no existe un diagrama de entidad-relación, aunque sí es posible desarrollar los documentos utilizados y su uso:

Elementos de los hilos RSS (FeedItems)

```
/**
 * @MongoDB\Document(repositoryClass="AppBundle\Repository\FeedItemRepository")
 */
class FeedItem
{
    /**
     * @MongoDB\Id(strategy="NONE", type="string")
     */
    protected $id;

    /**
     * @MongoDB\Field(type="string")
     */
    protected $title;

    /**
     * @MongoDB\Field(type="string")
     */
    protected $content;

    /**
     * @MongoDB\Field(type="date")
     */
    protected $date;

    /**
     * @MongoDB\Field(type="string")
     */
    protected $sourceUrl;

    /**
     * @MongoDB\Field(type="string")
     */
```

```
protected $sourceBlog;

/**
 * @MongoDB\Field(type="string")
 */
protected $link;

/**
 * @MongoDB\Field(type="string")
 */
protected $lang;
```

(se omiten los asignadores por legibilidad –getters y setters-)

Palabras a eliminar en las analíticas de texto (Stop-words)

```
/**
 * @MongoDB\Document(repositoryClass="AppBundle\Repository\StopWordRepository")
 */
class StopWord
{
    /**
     * @MongoDB\Id
     */
    protected $id;

    /**
     * @MongoDB\Field(type="string")
     */
    protected $word;
```

(se omiten los asignadores por legibilidad –getters y setters-)

Blog posts a los que hacemos seguimiento para capturar posts

```
/**
 * @MongoDB\Document(repositoryClass="AppBundle\Repository\BlogRepository")
 */
class Blog
{
    /**
     * @MongoDB\Id
     */
    protected $id;

    /**
     * @MongoDB\Field(type="string")
     */
    protected $title;

    /**
     * @MongoDB\Field(type="string")
     */
    protected $url;
```

```
/**
 * @MongoDB\Field(type="string")
 */
protected $rss;

/**
 * @MongoDB\Field(type="string")
 */
protected $lang;
```

(se omiten los asignadores por legibilidad –getters y setters-)

Prototipos

El formato de los prototipos se basa en la estructura típica de un panel de control o cuadro de mandos en formato web:

- Secciones a elegir a través de un menú lateral (en la izquierda)
- Contenidos relacionados con esa sección en el área de la derecha (mayor espacio de pantalla)

Lo-Fi

Captura 9. Prototipo Lo-Fi.

Hi-Fi

Una vez aplicado el *framework* Bootstrap y un poco de personalización a sus hojas de estilos, podemos ver un posible prototipo final, no lejos de la línea definitiva:

Captura 10. Prototipo Hi-Fi.

Guion

Al ser una herramienta pensaba para la interacción directa del usuario, no existe un guion predeterminado ni unos pasos a seguir.

Se pone a disposición del usuario la elección del camino a seguir, los tipos de datos a consultar e incluso la petición de nuevos datos a incluir en la aplicación (a través de los sistemas de *feedback* añadidos a la misma).

Al tratar de ser un proyecto relacionado con la comunidad ágil de Barcelona (como público directo, siendo el público del resto de España el público indirecto), se prevé la participación de esta comunidad en la definición de contenido a analizar, basándose en sus inquietudes y preguntas a responder por este conglomerado de datos.

Perfil de usuario

A pesar de que la herramienta en general es muy genérica y se podría prestar a diversos perfiles de usuarios basados en sus temáticas preferidas, la prueba piloto la hemos basado en el tema de las metodologías ágiles.

Al estar hablando de una temática tan específica, el perfil de usuario se reduce drásticamente (lo cual beneficia al concepto de "prueba piloto" ya que nos enfrentaremos al feedback de un grupo más reducido de personas que si hubiésemos escogido una temática general).

Perfil de usuario estándar

Perfil de usuario estándar	
Edad	Entre 20 y 55 años
Género	Femenino y masculino
Nivel de estudios	Estudios medios y/o universitarios
Intereses	Gestión de proyectos, metodologías de trabajo y de gestión de equipos
Experiencia y conocimientos	Experiencia en el sector del desarrollo de software, tanto de desarrollador como en algún otro rol relacionado (Project manager, CTO, director de desarrollo, Scrum Masters, Agile Coaches, Product Owners, etc.)

En cuanto a los tipos de usuarios (perfiles de usuarios) que podremos gestionar o utilizar a través de la web, nos encontramos los siguientes:

Usuario de consulta / visitante	
Punto de entrada	https://beta.agile.barcelona
Descripción	Será el usuario común de la aplicación. Podrá acceder a todos los datos disponibles y variar filtros de fechas o agrupaciones a través de las opciones que estén disponibles en la interfaz de usuario. También podrá ayudar a la validación de los datos (como eliminando post retweeteados que no tengan valor pulsando en la "x" correspondiente). No podrá gestionar datos a bajo nivel (gestionar palabras clave, blogs, etc.)

Usuario administrador	
Punto de entrada	https://admin.agile.barcelona
Descripción	Es un usuario de administración a todos los niveles. Una vez validado su acceso, este usuario podrá gestionar blogs e hilos de RSS, añadir “stopwords” para las tendencias de Twitter, etc. La única función administrativa que no estará a su alcance será la gestión de palabras clave a seguir en Twitter. Debido al proceso que gestiona esta parte, sólo será accesible a través de un administrador especial con acceso a los procesos a bajo nivel del servidor, independientemente de los usuarios gestionados en la aplicación.

Usabilidad / UX

El diseño de interacción de la aplicación se basa en la capacidad de acceso a la información casi de manera exclusiva.

No existe un mapa de navegación (toda la acción transcurre en lo que el usuario podría pensar que es una única página) ni un mapa del sitio definido.

La interacción se plantea en dos vertientes:

- La selección de la sección que contiene la información en la que estamos interesados (a través de un enlace en el menú lateral).

- La observación de los diferentes gráficos y tendencias. Es posible que se definan interacciones específicas para determinados gráficos o infografías, pero quedarían fuera del alcance de este proyecto.

Una tercera posibilidad de interacción sería la utilización de un formulario de contacto en forma de ventana emergente, para la solicitud de feedback y de temas a incluir (sugerencias del usuario final).

Seguridad

En el sistema de visualización de datos, debido a la falta de elementos en los que el usuario pueda introducir texto u otros datos, los requisitos de seguridad se reducen más al rendimiento de la aplicación que a su interacción.

Se habrán de tener en cuenta los formularios y las peticiones de feedback, tanto de esta sección del cliente como de la parte de administración (también visible al público, aunque protegida por un sistema de usuarios con nombre de usuario / contraseña). En ese aspecto se han de instalar contramedidas sobre ataques de inyección de código (XSS) y CSRF (*Cross-Site request forgery*), como *tokens* generados para autenticar la procedencia del formulario, etc.

Además de estas medidas, en lo referente a la utilización de la API creada por nosotros, será necesario realizar pruebas de carga y estrés para medir si sería necesario restringir su uso público y para evitar ataques de denegación de servicio (DDOS) que podrían dejar a la plataforma inestable o parada.

Es necesario añadir a nuestra preocupación sobre seguridad todos los componentes estándar del sistema (sistemas operativos, servidores web, bases de datos, etc.) para que estén actualizadas a sus versiones más estables y seguras (LTS o similares), con todos los tipos de parches de seguridad aplicados, lo que reducirá en cierto modo un posible ataque de *hacking* o intentos de acceso a las máquinas en producción.

Tests

La aplicación se entregará con una serie de pruebas que asegurarán (dentro de lo posible) su correcto funcionamiento en varios niveles:

- Pruebas unitarias: guían y proporcionan seguridad a los componentes unitarios de código (a cada clase u objeto desarrollado). Sirven para ser ejecutados con cada nueva integración de código, asegurando que lo que ya se ha escrito mantiene su funcionalidad y que el nuevo código también reacciona como se espera.

- Pruebas de integración: Son piezas de código que se ejecutan en el servidor de pruebas y aseguran la correcta comunicación entre diferentes componentes de la arquitectura del sistema. No solamente prueban su ecosistema sino también su integración con otros elementos, como la parte de cliente comunicándose con la API, etc.

- Pruebas “end-to-end”: A través de una serie de tecnologías podemos simular la interacción de un usuario real. Es decir, podemos programar la interacción, la selección de secciones y los gráficos como si las acciones las estuviera realizando un usuario real: “Si hago click en esta sección, debería ver el gráfico X y poder hacer click en el botón Y”.

Todos estos sistemas de pruebas nos proporcionan más seguridad a la hora de hacer refactorizaciones de código o de incorporar (o eliminar) funcionalidades a nuestra aplicación, aparte de considerarse una buena práctica en el desarrollo de software.

Adicionalmente, se desarrollará una prueba de carga y esfuerzo para tener una estimación de qué número de usuarios la aplicación será capaz de soportar con la infraestructura actual (todos los servicios en un solo servidor dedicado) para poder anticiparnos a su crecimiento.

Versiones de la aplicación

Las versiones que se estima obtener durante el desarrollo de la aplicación se basan de manera específica con las entregas programadas en el calendario de desarrollo de esta memoria de final de grado.

La primera versión (**alpha**), se proporcionará a la entrega de la PAC2:

Su funcionalidad será mínima, aunque todos sus componentes estarán en funcionamiento y probados. Concretamente, la aplicación cliente mostrará el número de tweets que la base de datos posee, organizamos por países de procedencia y/o idioma.

Es un hito importante, ya que con esa (aparentemente) simple funcionalidad, hemos delimitado mucho el riesgo de cambio de tecnologías o componentes arquitectónicos que existía al comienzo de la aplicación y probamos los siguientes puntos:

- El *backend* ya recibe datos de la API de Twitter y los almacena en su correspondiente sistema de base de datos (MongoDB en este caso).

- La API generada a partir de nuestro otro *backend* en NodeJS procesa peticiones que son redireccionadas a MongoDB y responde con los resultados generados.

- El cliente (*frontend* basado en AngularJS) lanza peticiones a la API y muestra los datos obtenidos.

Este proceso será similar para cualquier otro gráfico o información que generemos en el futuro, con lo que la incertidumbre se ha reducido a niveles muy bajos.

Seguiremos con la versión "**beta**" (programada para la entrega de la PAC3), en la que encontraremos las secciones disponibles en el menú lateral, así como los gráficos y análisis realizados hasta la fecha.

En una **versión 1.0**, hacia el final de nuestro trabajo de final de grado, habremos incorporado la información procedente de los blogs. De esta manera podremos cruzar la información y hacer crecer la riqueza y fiabilidad de nuestros datos (o al menos, basarlos en información más contrastada y ampliada).

Requisitos de instalación

A nivel **cliente**, y debido a la naturaleza de la aplicación (en entorno web, accesible a través de navegador o dispositivos móviles), no es necesaria ninguna configuración especial en el entorno que estemos utilizando.

A través de un simple navegador (relativamente moderno), se puede acceder a toda la funcionalidad ofrecida, tanto a la parte de consulta como de administración de los datos.

Desde el punto de vista del **servidor** que alojará la aplicación, necesitará cumplir una serie de requisitos, tanto de hardware como de software:

- Hardware: Un servidor con un mínimo de:
    - 2 Tb de disco duro (para almacenar suficientes datos para los análisis pertinentes)
    - 16 Gb de RAM (con el objetivo de que los servicios dispongan de margen para realizar operaciones secundarias y trabajos en segundo plano)
    - Conectividad a Internet
- Software: Sistema operativo Ubuntu (preferentemente, ya que es en donde se han realizado todas las pruebas)
    - Servidor web: Apache 2.4
    - PHP, versión mínima 5.6.
    - Servidores de base de datos: MySQL y MongoDB
    - Python 2
    - Node v7.6.0 en adelante.

A nivel software, será necesaria la instalación de librerías y paquetes secundarios para cada una de las aplicaciones. Las dependencias de las mismas son demasiado extensas como para listarlas en este documento, pero están declaradas en la documentación de cada aplicación (en sus ficheros de dependencias).

Instrucciones de instalación

Diferenciando las secciones de cliente y servidor nuevamente:

- El Cliente no necesita instrucciones de instalación específicas de la aplicación ya que se ejecuta en el entorno del navegador. Con tener un navegador actual instalado y funcionando, no hay nada más que configurar o instalar.

- A nivel servidor, las instrucciones de instalación y configuración varían dependiendo del módulo que estemos intentando instalar. A continuación, se resumen brevemente los pasos necesarios para la instalación, en caso de disponer de los ficheros de código fuente:

    - Administración / Backend:
        - Es una aplicación PHP – Symfony. Para su instalación es necesario descargar el código fuente, resolver sus dependencias a otros módulos a través de composer y preparar un subdominio en apache para su acceso.
        - Asimismo, será necesario configurar las bases de datos necesarias, en este caso, MySQL y MongoDB.

    - Visualización de datos:
        - Es una aplicación en AngularJS. Para instalarla se requiere el acceso a su código fuente, ejecutar los comandos necesarios para resolver sus dependencias con módulos externos (npm y bower) y gestionar el dominio necesario en apache.
        - Esta aplicación no dispone de base de datos propia ya que todo el acceso se hace a través de una API interna desarrollada para este propósito.

    - API para gestión de acceso a datos y utilidades:
        - Es una aplicación desarrollada sobre NodeJS. Se necesita el código fuente y la resolución de dependencias a través de la herramienta npm.
        - La base de datos a utilizar está basada en MongoDB.
        - También necesita un dominio en Apache que haga de proxy hasta el puerto en dónde se ejecuta la aplicación en node (normalmente distinto al 80 o 443 habituales).

    - Trabajos en segundo plano:
        - Es necesario gestionar trabajos en segundo plano para abrir la conexión con Twitter y recibir los datos, procesar nuevos blogs procedentes de los RSS registrados y, por último, generar los archivos de TagClouds para los blogs a través de unos scripts en Python.

Instrucciones de uso

Tanto a nivel administración como a nivel usuario de consulta, las instrucciones de uso son las mismas que para cualquier otra aplicación web.

Será necesario acceder a las direcciones en dónde estén alojadas las aplicaciones, introducir nuestros usuarios/contraseñas de acceso (o registrarnos, si es la primera vez que accedemos a la aplicación pública).

Ambos módulos están compuestos de un menú lateral desde el que se puede acceder a todas las opciones disponibles.

En la parte de administración, las opciones serán las relativas a mantener la lista de RSS que la aplicación consulta cada día para extraer los nuevos artículos o generar la nube de tags, mantener la lista de “stopwords” para las listas de tendencias y alguna otra opción de configuración a través de los formularios correspondientes.

En cuanto a la parte pública, cada sección del menú se centra en una serie de análisis diferentes (Twitter, Blogs, Análisis de Red, Influenciadores, etc.), así que será tan fácil como ir seleccionando la opción que más nos interese consultar y navegar así a través de la aplicación.

Bugs

El único detectado en este momento es la recuperación de contraseña para un usuario registrado. En realidad, no es un bug, es que la implementación de esta funcionalidad ha quedado aplazada para poder terminar otros temas de mayor prioridad.

Ha habido algo de incertidumbre respecto al funcionamiento de la API desarrollada y los clientes (debido a CORS, una directiva de seguridad que afecta a cómo los navegadores se comportan cuando una aplicación en un determinado dominio, solicita datos a otro dominio).

El dominio actual de la aplicación es https://beta.agile.barcelona y el dominio utilizado por la API para servir los datos es https://api.agile.barcelona. Esto hacía que Chrome y Firefox (entre otros) no permitieran al cliente mostrar los datos recibidos por la API.

La configuración actual del servidor añade una cabecera extra, informando al navegador que puede utilizar la información, lo que soluciona temporalmente el problema. La API se ha programado utilizando autentificación a través de token (JWT) de manera que los servicios solo son accesibles para usuarios logueados en la aplicación.

Proyección a futuro

Uno de los principales focos da la aplicación y del proyecto en sí mismo ha sido la investigación. Para esto, el mayor reto hasta el momento ha sido principalmente técnico. Es decir, entender cómo ser capaces de obtener todos los datos necesarios, cómo procesarlos, cómo analizarlos y cómo mostrarlos.

Con este hito ya conseguido, la proyección a futuro de la aplicación es la de entender progresivamente toda la información de la que ahora ya se dispone. Poco a poco se pretenden incorporar nuevas secciones con nuevos análisis, desde diferentes perspectivas y sobre diferentes temas.

Algunos ejemplos:

- Análisis sobre movimiento de los datos a nivel temporal y a nivel geográfico.
- Análisis de tendencias sobre el tiempo. ¿Cuándo se empezó a hablar de “x” y dónde? ¿Cómo ha llegado la información hasta nosotros en Barcelona?
- Predicción de tendencias basadas en la propagación de las analizadas hasta la fecha.
- Análisis de red (aplicar conceptos de SNA, Social Network Analysis, tales como centralidad, distancias, etc. entre miembros de la red de Twitter interesada en las metodologías ágiles).

Una opción más alejada en el tiempo podría ser la industrialización de la aplicación de cara al uso por terceras personas o empresas. Es decir, si buscásemos la monetización de la aplicación, se podría crear una “marca blanca” del sistema, que en lugar de seguir palabras como “agile”, “scrum”, etc, siguiese palabras relacionadas con cualquier otro tema o marca comercial.

Presupuesto

Para la realización de este proyecto se dispone únicamente de los recursos propios, no se ha contratado ningún servicio externo, como es obvio dada la finalidad del trabajo de final de grado.

Los gastos generados han sido los relativos a la puesta online de los servicios que componen la aplicación, que se resumen en los siguientes:

Tipo de gasto	Concepto	Total
Dominio	Agile.barcelona alojado en Dinahosting	40,00 € / año
Alojamiento	Servidor dedicado en Hetzner (EX40)	40,00 € / mes

Además de esto, podríamos contabilizar las horas que se le han dedicado en cuanto a desarrollo, codificación y pruebas, que se pueden resumir en, aproximadamente:

- 7 Sprints (iteraciones bisemanales) dedicados al desarrollo de la aplicación y sus componentes, a una media de 30h por Sprint, se resumirían en unas 210 horas dedicadas.

- Un desarrollador profesional autónomo podría cobrar una media de 60 € la hora, lo que haría un total de 12.600 € para llegar a la versión 1.0

Análisis de mercado

En este caso, la aplicación o herramienta no está pensada para satisfacer una necesidad real del mercado, ni con la intención de generar un beneficio / negocio posterior.

Dado este caso específico, se ha realizado un estudio de mercado sobre la existencia (o no) de herramientas de uso público que permitieran recuperar esta información y moldearla a través de análisis y estadísticas.

Los resultados han mostrado que este tipo de herramientas existen, pero son propietarias, dirigidas principalmente al sector del marketing, para gestionar audiencias. Muchas de ellas tienen una licencia anual (de unos 5.000 € / año) y permite a las empresas analizar las redes sociales para detectar el “estado de ánimo” de sus clientes.

Por este motivo, se afianza la idea de que hay un espacio para este tipo de aplicaciones de análisis de dominio público que no está siendo ocupado por ningún otro “competidor” en estos momentos.

Marketing y ventas

Al estar hablando de una aplicación que, en un principio, no tiene un objetivo económico detrás, se opta por una estrategia de marketing destinada a aumentar la imagen de marca, la presencia online y su popularidad en los círculos a los que está dirigida (el mundo de las metodologías ágiles).

Para empezar, se ha creado una "*landing page*" (ya que no tenemos una versión funcional hasta el momento) en donde se explica el proyecto, qué se quiere conseguir y cómo se puede colaborar (se desea que el usuario final esté involucrado en cierto modo en el desarrollo: proponiendo temas de estudio, tipos de estadísticas, etc.).

Esto hará que los usuarios lo tomen como "su" proyecto también y hagan de evangelizadores popularizando la herramienta allí donde no lleguen nuestros actos de marketing directo.

Por otro lado, se crean vías de comunicación en esa página para que los usuarios interesados ("*early adopters*") dejen su email y se les avisará una vez que haya una versión beta de la aplicación. Ellos serán los que aporten nuevas ideas, proporcionen detección de errores y beta-testing, a la vez que ayudan a mejorar el producto final.

También se ha promocionado esta página en foros de la comunidad ágil de Barcelona, en Twitter y a través de LinkedIn.

Una vez que la aplicación esté en producción (incluso en fase beta), se empezarán a publicar pequeños posts o tweets con los primeros gráficos o análisis que se hayan generado, para despertar todavía más la curiosidad de los usuarios.

Si se barajase la idea de monetizar el esfuerzo dedicado, se podría pensar en varias opciones:

- Publicidad dentro de la aplicación
- Sistema de suscripción en el que se podría permitir al usuario añadir sus propias palabras clave del tema que le interese, incluso con diferentes niveles de suscripción (free – 1 palabra, pro – hasta 10 palabras, Enterprise – ilimitado).

En el momento actual se descarta la opción de monetización por completo. Se pretende que sea un proyecto por y para la comunidad de manera gratuita.

Captura 11. Landing page para acciones de marketing. https://agile.barcelona

Conclusiones

El proyecto comenzó como una idea para experimentar y generar nuevo conocimiento a través del análisis de información sobre un nicho muy concreto.

Las sensaciones han sido muy positivas. Ha habido mucho aprendizaje a nivel técnico (sobre NodeJS y AngularJS sobre todo) y también mucha información procesada sobre análisis de redes y grafos, minería de texto y conceptos abstractos sobre relaciones entre nodos en redes sociales.

En cuanto a problemas descubiertos, nada que haya parado o modificado dramáticamente la planificación. Como en la mayoría de los proyectos de desarrollo de software, hay incógnitas. En este caso se han podido resolver de manera aceptable, tanto en cuanto a tiempo como en lo relativo a la calidad de las soluciones.

En resumen, me llevo una gran satisfacción por haberlo llevado a cabo y porque es todavía el principio de un proyecto que apunta a largo recorrido.

Anexo 1. Entregables de proyecto

Los entregables del proyecto se dividen en módulos o componentes de la arquitectura de la aplicación y sus bases de datos correspondientes:

Proyecto/Frontend_AngularJS/agile.barcelona.view.zip

El módulo que compone la parte cliente de la aplicación. Es el componente gestionado a través de AngularJS que recibe datos de la API y los procesa en la parte cliente para su visualización.

Proyecto/Backend_API_NodeJS/agile.barcelona.node.zip

El módulo que gestiona los puntos de acceso a la API. A su vez, estos puntos de la API gestionan la lógica de negocio y los accesos a base de datos.

Proyecto/Backend_Gestion_PHP/agile.barcelona.php.backend.zip

En este fichero se encuentra la aplicación Symfony que se dedica a la obtención de datos de las diferentes fuentes (Twitter API y RSS procedentes de blogs). También gestiona los usuarios con acceso a la parte administrativa del backend y, en un futuro, las palabras clave que la aplicación está gestionando.

Proyecto/Landing_Page/agile.barcelona.landing.zip

Web estática creada para acciones de marketing ("*landing page*") mencionada en el apartado de Marketing y ventas.

Proyecto/Bases de datos/agilebcn.sql

Extracto de la base de datos MySQL que gestiona los usuarios y los trabajos en segundo plano.

Proyecto/Bases de datos/mongoTweets.zip

Extracto de la base de datos MongoDB con una colección de Tweets para pruebas (1200 tweets aproximadamente).

Proyecto/Bases de datos/mongoBlogs.zip

Extracto de la base de datos MongoDB con una colección de RSS a los que seguir para su proceso posterior.

Proyecto/Bases de datos/mongoFeedItems.zip

Posts procesados sobre los que se realiza la selección de los últimos publicados, así como las nubes de tags generadas.

Proyecto/Scripts de background/pythonScriptsForTextMining.zip

Script en Python para el proceso offline de minería de texto (como la generación de nubes de tags para los blogs).

Anexo 2. Código fuente (extractos)

Con los siguientes extractos de código fuente se pretende mostrar, de una manera muy resumida, el núcleo de la aplicación y su funcionamiento de extremo a extremo:

```php
<?php

namespace AppBundle\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

class TwitterCommand extends ContainerAwareCommand
{
    protected function configure()
    {
        $this->setName('twitter:stream')->setDescription('Listen for Live Twitter Feeds.');
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        // TRACKS or keywords to listen to from the Streaming API.
        $twitterTracks = ['agile', 'scrum', 'cynefin', 'kanban'];

        // Get twitter service and open the low-latency communication channel.
        $this->getContainer()->get( id: 'twitter.service')->listenForTweets(
            $twitterTracks
        );
    }
}
```

Captura 12. Escuchando en la API de Twitter

A través de este extracto podemos observar el comando que se ejecuta en segundo plano para mantener la conexión abierta con Twitter y recibir los Tweets relativos a las palabras seleccionadas:

- La variable $twitterTracks es un array en el que se definen las palabras sobre las que queremos “escuchar” en el canal abierto de Twitter.

- A partir de ahí, el código utiliza una función de “callback” para que, cada vez que se reciba un Tweet relacionado con los temas que queremos, el Tweet se procese por la aplicación de alguna manera (almacenándolo en la BD en nuestro caso).

```php
<?php

namespace AppBundle\Services;

use Symfony\Bundle\FrameworkBundle\Routing\Router;

// Include the third-party library
require_once __DIR__.'/../../../vendor/fennb/phirehose/lib/OauthPhirehose.php';

class TwitterListener extends \OauthPhirehose
{
    private $mongodb_database = null;

    public function __construct($twitter_streaming_api, $mongodb_database) {
        parent::__construct(
            $twitter_streaming_api['application_token'],
            $twitter_streaming_api['application_token_secret'],
             method: \Phirehose::METHOD_FILTER
        );

        $this->mongodb_database = $mongodb_database;
        $this->consumerKey = $twitter_streaming_api['consumer_key'];
        $this->consumerSecret = $twitter_streaming_api['consumer_secret'];
    }

    public function enqueueStatus($status)
    {
        $twitterFeed = json_decode( json: $status,  assoc: true);
        if (is_array($twitterFeed) && isset($twitterFeed['user']['screen_name'])) {

            // Real-time Twitter Feed. Write your own implementation over here.
            $m = new \MongoClient();
            $db = $m->{$this->mongodb_database};
            $collection = $db->tweets;
            $collection->insert($twitterFeed);
        }
    }
}
```

Captura 13. Recibiendo y escribiendo los tweets en la base de datos.

En la captura 13 podemos observar qué ocurre en el momento en el que recibimos un nuevo Tweet del canal de Twitter:

- La función __construct configura la base de datos y los parámetros necesarios para abrir la conexión con la API de Streaming de Twitter.

- La función enqueueStatus se encarga de recibir uno de esos Tweets y procesarlo:

    - Se comprueba que la estructura de datos sea correcta (que sea un json y que el screen_name esté definido)

    - Se realiza la conexión a la base de datos de Mongo, usando la base de datos y la colección definidas en la configuración de la aplicación y se procede a inserta el Tweet correspondiente.

```
const express = require('express');
const cors = require('cors');
const monk = require('monk');
const charts = require('./routes/charts');
const tweets = require('./routes/tweets');
const bodyParser = require('body-parser');

const app = express();
const db = monk('localhost:27017/tweetsRaw');

app.set('view engine', 'pug');

app.use(cors());
app.use(bodyParser.json()); // support json encoded bodies
app.use(bodyParser.urlencoded({ extended: true })); // support encoded bodies

// Make our db accessible to our router
app.use(function(req,res,next){
  req.db = db;
  next();
});

app.get('/v1/charts/:id', charts.findById);
app.get('/v1/tweets/count', tweets.totalNumberOfTweets);

app.listen(3000);
console.log('Listening on port 3000...');
```

Captura 14. El servidor NodeJS preparado para enviar los datos a través de la AP

En la Captura 14 pasamos a observar otra de las partes de la aplicación. Las capturas anteriores se basaban en la parte de administración o backend, mientras que ahora estamos viendo la aplicación que gestiona la API de puente entre nuestros datos y el frontal que los mostrará.

Este extracto nos muestra el fichero principal de la aplicación de NodeJS, en donde se crea el servidor que gestionará las llamadas y los servicios.

Aquí vemos una primera parte en donde se declaran todas las dependencias y objetos a utilizar (todas las líneas iniciales con “const”).

A partir de ahí configuramos el objeto “app” para que utilice un determinado motor de plantillas (“pug”), para que ofrezca servicios que soporten peticiones de diferentes dominios (“cors”), algunas utilidades para gestionar parámetros enviados en las peticiones de manera más sencilla (“bodyParser”) y una opción para que sea fácil acceder a la base de datos desde dentro de cada uno de los servicios.

A partir de ahí se declaran los “endpoints” del API (“app.get”) y se ejecuta el servidor para que se quede a la escucha en el puerto 3000 y reciba las peticiones de los frontales que la necesiten.

```
'use strict';

angular.module('agileBarcelona').controller('DashboardController',
  function ($log, TweetsTotalCount, DashboardCharts) {

    var vm = this;

    TweetsTotalCount.get({},
      function success(response) {
        vm.totalTweets = response.numTotalTweets;
      },
      function error(errorResponse) {
        $log.debug("Error:" + angular.toJson(errorResponse));
      });

    TweetsTotalCount.get({lang: 'en'},
      function success(response) {
        vm.totalTweetsEN = response.numTotalTweets;
      },
      function error(errorResponse) {
        $log.debug("Error:" + angular.toJson(errorResponse));
      });
```

Captura 15. El controlador de la vista, pidiendo los datos a la API

La Captura 15 nos ofrece ahora una visión del controlador de una de las pantallas de la aplicación en AngularJS. Es decir, del frontal de la aplicación que mostrará los datos.

En él vemos cómo se declara el controlador que gestionará la vista, se hacen las llamadas a cada uno de los “endpoints” que nos envían los datos necesarios (TweetsTotalCount totales y en español si enviamos el parámetro “lang”).

Una vez obtenida la respuesta, el valor se almacena en una variable que hace que sea accesible desde la plantilla de la vista directamente (vm.totalTweets, por ejemplo).

```
(function() {
  'use strict';

  /* Agile Barcelona API Services */

  angular.module('agileBarcelonaApiServices', ['ngResource', 'agileBarcelonaEnvConfig'])

  factory('TweetsTotalCount',
    function ($resource, EnvironmentConfig) {

      return $resource(EnvironmentConfig.api + "/v1/tweets/count", {}, {
        get: {method: 'GET', cache: false, isArray: false}
      });

    })

  .factory('DashboardCharts',
    function ($resource, EnvironmentConfig) {

      return $resource(EnvironmentConfig.api + "/v1/charts/:id", {}, {
        get: {method: 'GET', cache: false, isArray: false}
      });

    });

})();
```

Captura 16. Detalle de las peticiones a la API desde la vista

En esta captura se nos ofrece un detalle de cómo está construida la llamada anterior a TweetsTotalCount. De esta manera podemos ver cómo está relacionada con el código que hemos visto cuando hablábamos de la API y los “endpoints” que hemos definido allí.

```
<div class="col-lg-3">
    <div class="ibox float-e-margins">
        <div class="ibox-title">
            <span class="label label-primary pull-right">En Español</span>
            <h5>En Español</h5>
        </div>
        <div class="ibox-content">
            <h1 class="no-margins">{{ chart.totalTweetsES }}</h1>
        </div>
    </div>
</div>
```

Captura 17. Documento HTML en donde se muestran finalmente los datos al usuario

Para finalizar, la Captura 17 nos enseña el último paso, cómo utilizar la variable del controlador para mostrar los datos dentro de una estructura HTML común, que genera el resultado final:

Captura 18. Resultado final en el navegador

Anexo 3. Librerías, código externo utilizado

Pasamos a identificar las librerías externas que forman la base de cada uno de los componentes y de la herramienta en sí misma:

Captación de datos – Backend

Symfony 3 y librerías (dependencias) necesarias:

```
"require": {
    "php": ">=5.5.9",
    "symfony/symfony": "3.2.*",
    "doctrine/orm": "^2.5",
    "doctrine/doctrine-bundle": "^1.6",
    "doctrine/doctrine-cache-bundle": "^1.2",
    "symfony/swiftmailer-bundle": "^2.3.10",
    "symfony/monolog-bundle": "^3.0.2",
    "symfony/polyfill-apcu": "^1.0",
    "sensio/distribution-bundle": "^5.0",
    "sensio/framework-extra-bundle": "^3.0.2",
    "incenteev/composer-parameter-handler": "^2.0",
    "twig/twig": "^1.0||^2.0",
    "jms/serializer-bundle": "^1.2.1",
    "aboutcoders/job-bundle": "1.0.0",
    "sonata-project/notification-bundle": "^3.1",
    "doctrine/mongodb-odm": "^1.1",
    "doctrine/mongodb-odm-bundle": "^3.2",
    "fennb/phirehose": "^1.0",
    "friendsofsymfony/user-bundle": "dev-master"
```

Captura 19. Listado completo de librerías de Backend

Del listado completo de las librerías de Symfony, podemos descatar:

- Doctrine: utilizada para todas las funcionalidades de acceso a datos.
- Swiftmailer: para servicios de envio de email y mensajes.
- Monolog: para hacer logging de eventos dentro de la aplicación (debug, por ejemplo).
- Twig: librería para la gestión de plantilla html.
- Phirehose: librería de acceso a la API de Twitter.

API de servicio de datos

```
"devDependencies": {
  "babel": "^5.8.21",
  "gulp": "^3.9.1",
  "gulp-babel": "^6.1.2",
  "gulp-eslint": "^3.0.1",
  "gulp-help": "^1.6.1",
  "gulp-istanbul": "^1.0.0",
  "gulp-jsdoc3": "^0.3.0",
  "gulp-mocha": "^2.2.0",
  "isparta": "^3.0.3",
  "pre-commit": "^1.1.3",
  "run-sequence": "^1.2.2"
},
"dependencies": {
  "body-parser": "^1.17.1",
  "cors": "^2.8.3",
  "express": "^4.15.2",
  "mongodb": "^2.2.25",
  "monk": "^4.0.0"
}
```

Entre ellos, destacaremos los siguientes:

- Gulp: librería para la compilación y distribución de la aplicación.
- Cors: para gestionar las peticiones de dominios externos a la API.
- Express: capa middleware para gestionar los puntos de acceso de la API.
- Mongodb: librería de acceso a MongoDB para NodeJS.
- Monk: librería de acceso a datos que extiende MongoDB.

Componente de visualización

```
"dependencies": {
  "angular-animate": "~1.5.0",
  "angular-cookies": "~1.5.0",
  "angular-touch": "~1.5.0",
  "angular-sanitize": "~1.5.0",
  "angular-messages": "~1.5.0",
  "angular-aria": "~1.5.0",
  "angular-flot": "~0.0.16",
  "jquery": "^3.1.1",
  "angular-resource": "~1.5.0",
  "angular-ui-router": "~0.2.15",
  "bootstrap": "~3.3.7",
  "angular-bootstrap": "~1.1.2",
  "moment": "~2.10.6",
  "animate.css": "~3.4.0",
  "angular": "~1.5.0",
  "pace": "~1.0.2",
  "metisMenu": "~2.0.2",
  "fontawesome": "~4.7.0",
  "oclazyload": "~1.1.0",
  "jquery-flot": "0.8.2"
},
```

El componente de visualización también incluye librerías comentadas en el módulo anterior, como Gulp, aparte de las propias de la parte de la vista:

- AngularJS: librería javascript para gestionar aplicaciones de página única (SPA)
- Jquery: librería para gestionar determinados eventos en la vista, que sirve de base para otras librerías que extienden el core de Jquery.
- Bootstrap: librería para añadir capacidades “responsive” y soportar multidispositivo.
- JQuery-flot: librería que extiende Jquery y añade funcionalidades específicas para la generación de gráficos e infografías.

Anexo 4. Capturas de pantalla

Todas las capturas de pantalla han sido incluidas en sus respectivos apartados. También se ha proporcionado un índice de capturas al inicio del documento.

Anexo 5. Libro de estilo

Logo

Basado en la idea de un gráfico (relacionado con los análisis y la idea de información), englobado en un círculo que refleja un tema concreto a analizar.

Paleta de colores

Fondo del menú: #1A1A1A

Fondo de la aplicación principal: #F3F3F4

Titulares y resaltados: #A51111

Tipografía

Open Sans, de diferentes tamaños y pesos dependiendo del lugar de la aplicación (títulos, encabezados, texto normal, etc.)

Datos adicionales

Para el comportamiento “responsive” de la maquetación se ha recurrido a las especificaciones del framework “Bootstrap”, modificado y adaptado a las necesidades concretas de esta aplicación.

Anexo 6. Resumen ejecutivo

Debido a la naturaleza de la aplicación y el desarrollo basado en la aportación a la comunidad y no a un ánimo de lucro, no se adjunta plan de negocio o resumen ejecutivo.

Anexo 7. Glosario

Las palabras técnicas o denominaciones especiales están recogidas en las anotaciones a pie de página de cada una de las páginas que realizan las menciones.

Anexo 8. Bibliografía

Elad Elrom (diciembre 2016). PRO MEAN Stack Development. Apress

Ken Williamson (marzo 2015). Learning Angular JS. O'Reilly Media Inc.

Lukas Ruebbelke y Brian Ford (Julio 2015). AngularJS in Action. Manning Publications.

Kevin Makice (marzo 2009). Twitter API: Up and Running. O'Reilly Media Inc.

Matthew A. Russell (octubre 2013). Mining the social web, 2nd edition. O'Reilly Media Inc.

Alexander Kouznetsov y Maksim Tsvetovat (Septiembre 2011). Social Network Analysis for Startups. O'Reilly Media Inc.