UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA DE LAS TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Curso Académico 2014/2015

Proyecto Fin de Grado

APLICACIÓN HTML5 DE SEGUIMIENTO DE DESARROLLO DE SOFTWARE

Autor : Jesús Alonso Barrionuevo

Tutor : Dr. Jesús M. González Barahona

Levántate pronto si anhelas vida o tierra ajenas.

El lobo perezoso no caza corderos.

Las batallas no se ganan en la cama.

Hávamál

Dedicado a

mi familia

II

Agradecimientos

Estando en el instituto, un profesor me dijo en una ocasión ”Los estudios es la única cosa personal que existe en este mundo. Tu estudias, tu te apruebas y tu te lo curras para tener algo sin que nadie meta las narices en ello”. La verdad es que aquél hombre no podría estar más equivocado, y ahora lo veo claro.

Durante todos estos años he estado acompañado, aun que en muchas ocasiones no he podido verlo. Mi padre y mi madre que siempre han estado a mi lado, así como mis abuelos dándome ánimos, aguantando mis cabreos y mis rachas de bajona, pero sobre todo creyendo en mi cuando ni yo mismo creía. Gracias por aguantar mi mal genio, y por ayudarme cuando realmente lo necesitaba.

He de reconocer que el camino no ha sido fácil, y debo decir del mismo modo que entré engañado a la universidad, pues antes de entrar todos decían que no sería igual de complicado y estresante que el bachillerato. Sin embargo, no todo ha sido malo, y he sacado algunas cosas buenas de mis años de estudiante universitario. He aprendido que el compañerismo y la camaradería son las mejores herramientas a la hora de enfrentarte a los problemas, y por ello debo darle las gracias a Quan y Javier (Sanjuan) con quienes más tiempo he pasado estos años y con quienes he ganado las batallas más grandes. No por ello debo olvidar al resto de mis amigos de la universidad, sin los cuales mucho de lo que he conseguido no sería posible, y aun que vuestro nombre no aparezca entre estas líneas, quiero que sepáis que también va para vosotros.

Debo agradecerle del mismo modo el esfuerzo que han realizado algunos de los profesores que he tenido durante todos estos años, Jesús, Gregorio, Inmaculada, Ana Belén y una lista interminable que haría que la memoria pasara al rango de libro. Gracias por enseñarme, por aguantar mi cabezonería, mi dura sesera y por aguantar mi legendaria pesadez.

En todo camino existen piedras, algunas forman parte del mismo, y otras son colocadas por los viajeros que te encuentras en él. Pese a tropezarme con ellas, debo darles las gracias del

mismo modo que se lo he dado a los aliados que he tenido durante este viaje. Si no fuera por vosotros, tal vez ahora sería más débil de lo que hubiera sido sin vuestros guijarros. A vosotros os dedico estas palabras.

El secreto de la educación reside en respetar al estudiante.-Ralph Waldo Emerson.

No menosprecies al cachorro débil, podría convertirse en un tigre feroz.-Proverbio mongol

Resumen

Este trabajo desarrolla una aplicación basada en HTML5, CSS3 y JavaScript de representación de datos y estadísticas procedente de los JSON de producción de un proyecto en el que pueden intervenir una o más empresas.

El objetivo principal de la aplicación era proporcionar un entorno de desarrollo semejante al que ofrecían otros tales como Freeboard, pero orientado a un objetivo claro, y cuyo uso puede ser aplicado a empresas o proyectos más cercanos como Openstack o Libresoft.

Entre los objetivos del desarrollo del presente proyecto se encuentra la creación de una aplicación cuya arquitectura es simple a la par que completa. Buscamos con ello dar la oportunidad al usuario de realizar una configuración personalizable en todos los aspectos de su DashBoard.

Por otro lado dar a los desarrolladores la oportunidad de ampliarla en un futuro, exponiendo una arquitectura escalable y fácilmente aplicable a otros ámbitos.

Summary

This project develops an application based in HTML5, CSS3 and JavaScript that can represent data and stradistic from ficheros JSON of production from many companies.

The final objective of this application is to provide us an enviroment such like other applications like Freeboard, but oriented to a concise objetive to be aplied to other projects or companies like Openstack or Libresoft.

In the objectives of this project we can find the creation of an application whose architecture is simple and complete.We want to give to the user the opportunity to make his own personalizable configuration in all of aspects of a DashBoard.

On the other hand, this architecture gives to developments the possibility to extend this.

Índice general

1. Introducción **1**

1.1. Objetivos 1

1.1.1. Descripción del problema 1

1.1.2. Objetivo principal 2

1.1.3. Requisitos 2

1.2. Demos 3

2. Contexto del proyecto **5**

2.1. Software libre 5

2.1.1. Ingeniería del software libre 6

2.1.2. Herramientas para el análisis de proyectos de software libre 8

2.2. Recogida de datos 8

2.2.1. MetricsGrimoire 9

2.2.2. vizGrimoire 9

2.3. Otros ámbitos relacionados 10

2.3.1. OpenStrack 10

2.3.2. Freeboard 11

2.3.3. GitHub 12

2.3.4. Many Eyes 13

3. Tecnologías utilizadas **15**

3.1. HTML5 15

3.1.1. Concepto 15

3.1.2. Nuevas etiquetas 16

3.2. CSS/CSS3 17
3.2.1. Concepto 17
3.2.2. CSS 1 17
3.2.3. CSS 2 18
3.2.4. CSS 2.1 18
3.2.5. CSS3 18
3.3. JavaScript 19
3.3.1. Concepto 19
3.3.2. Características 20
3.4. Jquery 22
3.4.1. Concepto 22
3.4.2. Características 22
3.5. Bootstrap 23
3.6. Python 24
3.6.1. Concepto 24
3.6.2. Historia 25
3.6.3. Características 27
3.6.4. Expresiones regulares 28
3.7. Django 29
3.7.1. Concepto 29
3.7.2. Características 29
3.8. Módulo SQLite 30
3.8.1. Concepto 30
3.8.2. Características 31
3.9. AJAX 32
3.9.1. Concepto 32
3.9.2. Funcionamiento 32
3.9.3. Características 34
3.10. Highcharts 34
3.10.1. Gridster 36

4. Desarollo **37**

4.1. Modelo de desarollo 37

4.2. Iteración 0. Estudio previo 40

4.2.1. Introducción 40

4.2.2. Desarrollo del ciclo 40

4.2.3. Bibliotecas y aplicaciones exploradas 46

4.3. Iteración 1. Creación de un *DashBoard* genérico 51

4.3.1. Introducción 51

4.3.2. Desarrollo del ciclo 51

4.4. Iteración 2. Creación de un sistema de guardado 58

4.4.1. Introducción 58

4.4.2. Desarrollo del ciclo 58

4.5. Iteración 3. Implementación mediante descomposición por objetos 62

4.5.1. Introducción 62

4.5.2. Desarrollo del ciclo 62

4.6. Iteración 4. Ficheros de configuración complejos, independencia en representación y múltiples empresas 70

4.6.1. Introducción 70

4.6.2. Desarrollo del ciclo 70

4.7. Arquitectura general 75

5. Conclusiones **79**

5.1. Lecciones aprendidas 80

5.2. Conocimientos aplicados 80

5.3. Trabajos futuros 81

A. Instalación y uso **83**

A.1. Requisitos 84

B. Formato de objeto Highchart **85**

C. Implementación, funciones importantes (Iteración 2) **89**

D. API **99**

D.1. Objeto panel 99

D.2. Objeto widget 101

Bibliografía **107**

Índice de figuras

2.1. Ejemplos de vizGrimoire 10
2.2. Ejemplo de Freeboard 12
2.3. Ejemplos de Many Eyes 14

3.1. Plantilla Bootstrap 24
3.2. Ejemplo realizado con Highcharts 35
3.3. Demo de Gridster 36

4.1. Esquema del modelo scrum 38
4.2. Ejemplo de widget de la biblioteca Chart.js 41
4.3. Ejemplo de widget de la biblioteca Highcharts.js 42
4.4. Ejemplo de widget de la biblioteca Highcharts.js con datos reales 45
4.5. Primer prototipo de DashBoard 46
4.6. DashBoard vacío 52
4.7. Primeros paneles creados 52
4.8. Primer prototipo de creación de una gráfica de tipo tiempo 53
4.9. Primer prototipo de creación de una gráfica de tipo demografía 53
4.10. Primer prototipo de creación de una gráfica de tipo información 54
4.11. Primeros widgets creados 54
4.12. Primeros widgets creados 55
4.13. Ejemplo de widget demográfico modificado 55
4.14. Gráfica resultante de clicar sobre una gráfica de demografía 56
4.15. Widgets recien creados 59
4.16. Guardado completado 60

4.17. Visión del administrador del servidor 60
4.18. Ejemplo de los resultados con esta nueva arquitectura 68
4.19. Chart de panel 1 en panel 3 68
4.20. Ejemplo de widget con video 69
4.21. Ejemplo de nuevo formato de JSON de configuración 69
4.22. Menú de compañías 73
4.23. Arquitectura general 75
4.24. Arquitectura en objetos 76

D.1. Creación de un panel 99
D.2. Opciones del panel 101
D.3. Menú de widgets 101
D.4. Menú de cada widget 102
D.5. Menú del wiget de datos de evolución temporal 102

Capítulo 1

Introducción

En este primer apartado se hará una introducción al problema que vamos a tratar, objetivos a cumplir y objetivos genéricos del propio proyecto, con el objetivo de entrar en contexto para entender con mayor facilidad el desarrollo de la aplicación y el hilo de la memoria.

1.1. Objetivos

1.1.1. Descripción del problema

Este proyecto puede considerarse ligado a uno más grande que pretende potenciar una rama de la ingeniería del software: la ingeniería del software libre. Esta pretende aprovechar la existencia de una ingente cantidad de información accesible derivada de las formas de desarrollo abiertas (código fuente, comunicaciones entre desarrolladores, etc.) que llevan a cabo los proyectos del software libre, de manera que puedan ser cuantificados, medidos y estudiados. Los resultados y su consiguiente análisis ayudarán enormemente en la comprensión de los fenómenos asociados a la generación del software libre, al tiempo que facilitarán la toma de deciones a partir de la experiencia adquirida.

Por otro lado la versatilidad de lo que aquí presentamos puede ser aplicado del mismo modo a proyectos no libres, cuyos resultados pueden ser útiles para las propias empresas que los producen.

Antes de concluir mencionar la extensa diferencia entre lo que aquí se pretende desarrollar y lo existente en la mayoría de los casos en lo referente a la representación de datos. Por norma

general la mayoría de DashBoards desarrollados hasta la fecha presentan una apariencia estática dónde se muestra un análisis extenso de los datos sobre el cual poder sacar conclusiones, impidiendo al usuario acceder únicamente a los datos que él desee. Desarrollaremos de este modo un DashBoard dinámico en el que a diferencia de lo que se encuentra comúnmente en la red permita al usuario crear su propio escritorio personalizado de datos.

1.1.2. Objetivo principal

El presente proyecto se centra en el examen de datos relevantes recopilados durante el desarrollo de proyectos de desarrollo de software. Como objetivo propone la elaboración de una herramienta que sirva de base para la creación de gráficos con la extracción de métricas específicas a cada tipo de fuente de información.

Estas utilidades han sido desarrolladas usando JavaScript como lenguaje de programación, basando la captación de la información de interés. Una información, que ha de ser agrupada y empaquetada en un formato flexible y no adscrito a ninguna de las distintas herramientas de extracción o de análisis, utilizando para ello Python y GitHub, ofreciendo con ello el acceso a estos entornos ya creados en un futuro.

1.1.3. Requisitos

El proyecto cumplirá ciertos requisitos básicos:

- Definir una interfaz unificado para herramientas futuras.
- Almacenar resultados en formato intermedio para su posterior análisis
- Fácil puesta en marcha y actualizaciones.
- El tiempo de extracción de datos no será crítico.
- Permitir ampliar la aplicación con la creación de nuevos widgets de forma sencilla gracias a su arquitectura.
- Posibilidad de compartir los entornos ya creados sin necesidad de registro en GitHub.

1.2. Demos

Si se esta leyendo la versión pdf del presente proyecto, accediendo a los siguientes enlaces podréis haceros una idea general con las demos que están situadas en GitHub de cual es la idea de os objetivos del proyecto y de lo que se busca.

Ejemplo con datos OpenStack

Ejemplo sin datos

Repositorio

Capítulo 2

Contexto del proyecto

En este segundo apartado se realizara una breve introducción al software libre, explicando sus conceptos básicos, pero centrándonos sobre todo en la ingeniería dedicada al estudio de este tipo de software. Se incluye, además, una mención a otras herramientas que comparten el objetivo de analizar proyectos de software libre.

El presente proyecto que se expone en las subsiguientes líneas ha sido enteramente realizado con software libre en todos sus aspectos. Las bibliotecas de las que nos hemos valido, las tecnologías y las bases de datos están al alcance de cualquiera en la web sin ninguna clase de cargo.

Como ya se ha mencionado, pese a poderse aplicar a cualquier ámbito de seguimiento de datos, muy útil para universidades o empresas, tiene un mayor valor en proyectos de software libre donde el conocimiento empírico obtenido mediante la recopilación de datos de la evolución de los proyectos adquiere un valor especial.

2.1. Software libre

Todo programa que sea considerado software libre debe ofrecer una serie de libertades. Se resumen en: libertad de usar el programa con cualquier fin, sin necesidad de comunicarlo a los desarrolladores; libertad de estudiar el código fuente del programa y modificarlo adaptándolo a nuestras necesidades, sin necesidad de hacer públicas las modificaciones; libertad de distribuir copias, tanto binarios como código fuente, modificadas o no, gratis o cobrando por su distribución; libertad de modificar el programa y publicar las mejoras para beneficio de la comunidad.

2.1.1. Ingeniería del software libre

El enfoque sistemático y cuantificable que propone la ingeniería del software siempre tuvo como barreras las propias de las formas en que el software ha sido desarrollado, publicado y distribuido. Aspectos como el formato binario, oscurantismo en modelo de negocio y limitaciones comerciales han impedido validar resultados por parte de equipos independientes.

El reciente auge del software libre aporta novedades a esta ingeniería del software. La implantación de Internet junto con las licencias que fomentan la colaboración en el desarrollo del software, han favorecido a que además del código fuente, se diponga de repositorios de versiones donde observar la evolución del software o listas de correo que reflejan las comunicaciones durante el desarrollo. De estas fuentes puede obtenerse gran cantidad de datos de valor, incluso de forma automatizada.

Varios factores son los aportados a la ingeniería del software tradicional desde ingeniería del software libre:

- Visión temporal incorporada al análisis: necesaria ya que el proceso de creación cambia y su evolución analizada de forma continua proporciona información muy interesante (lenguajes más usados, evolución de colaboradores de un proyecto) destinada a servir de ayuda en la toma de decisiones.

- Análisis a gran escala: dada la inexistencia de impedimentos para ampliar el análisis al conjunto global de los proyectos de software libre gracias a la disponibilidad de la información generada durante su desarrollo. La ingeniería del software libre hace posible evaluar un proyecto dentro de entornos globales y de menor envergadura, ofreciendo información desde distintos puntos de vista, lo cual beneficia en la mencionada toma de decisiones.

En cierto modo, la ingeniería del software libre plantea cuantificar unos parámetros que nos permitan pronosticar con precisión costes, recursos y plazos.

En la actualidad el software libre carece de estos métodos, aunque la disponibilidad del código fuente y la información generada durante su desarrollo constituye un enorme potencial para que cambie esta situación.

La ingeniería del software pretende también aplicar las cualidades de la ingeniería del software en el desarrollo de proyectos de software libre, de modo que se garantice a los desa-

rrolladores la forma de generar software de calidad siguiendo los paradigmas adecuados. La ingeniería del software pretende aportar resultados objetivos y contrastables acerca de la evolución del software y desterrar así apreciaciones que algunos dan por ciertas. A corto plazo, la ingeniería del software libre tiene por objetivo realizar un análisis completo del desarrollo del software libre permitiendo indagar en los procesos que están involucrados. Puede considerarse que el software libre funciona gracias a una "mano negra", que hace que el software se genere mágicamente. Es por ello que la ingeniería del software libre busca comprender los elementos e interacciones que engloba esta laguna de conocimiento denominada "mano negra".

Se hace imprescindible un análisis de los datos relacionados con el software libre para poder alcanzar los objetivos, previamente descritos, que la ingeniería del software libre se propone. Debe procurarse que las herramientas empleadas para ello están disponibles para que grupos independientes puedan verificar los resultados.

En este proceso de análisis pueden diferenciarse dos fases. En la primera etapa, un grupo de utilidades independientes entre sí recogen datos cuantificables del código fuente y otros flujos de información y almacenan los resultados en un formato intermedio, que serán analizados en la siguiente fase. Lo ideal es que esta fase se realice de forma automática.

La segunda fase, no tan madura como la anterior, integra programas que toman como entrada los parámetros almacenados en el formato intermedio y se dedican a su análisis, procesado e interpretación. Se han propuesto varios modos de analizar los resultados, de entre las que destacamos: herramientas de análisis de clásters, que a partir de porciones reducidas de datos, agrupan los interrelacionados con el objetivo de categorizarlos; herramientas de análisis estadístico, que simplifican el procesado de grandes cantidades de datos y permiten mostrar gráficamente los resultados; una interfaz web, cuyo fin es, además de proporcionar acceso a los resultados de este gran proyecto, la aplicación de los programas que generan este interfaz a otros proyectos de software libre, de modo que surja una realimentación del proyecto global.

En cuanto a las fuentes a analizar, la que alberga mayor información en potencia es el código fuente. De él pueden extraerse parámetros como tamaño, número de líneas lógicas o físicas, número de desarrolladores, lenguaje de programación, etc. Uno de los estudios pioneros en este campo se encarga de calcular el número de líneas físicas de código de proyectos de software libre, tiempo y recursos empleados en el desarrollo del software.

Otras fuentes de interés son aquellas donde se produce intercambio de información entre desarrollCadores, como listas de correo o canales de IRC.

De estos últimos aún no se han definido claramente los parámetros a buscar, mientras que de las listas interesa recuperar de cada mensaje del archivo: el nombre y dirección del autor, la fecha, e incluso podría cuantificarse la longitud del mensaje.

Existe otro tipo de fuentes de información compuesto por una serie de herramientas que sincronizan el trabajo de los distintos desarrolladores de un software. Las más comunes son los sistemas de control de versiones, de los cuales obtener conclusiones acerca de la participación de cada desarrollador; y los sistemas de gestión de errores.

Una modalidad más de recopilación, quizás aplicable en un futuro, es la relacionada con la información personal de los desarrolladores, que a día de hoy no suele facilitarse, y que ayudaría a conocer con mayor profundidad la comunidad del software libre. Además, si se dispusiera de los datos laborales de estos desarrolladores podría establecerse una previsión de costes y recursos para futuros proyectos de software libre.

2.1.2. Herramientas para el análisis de proyectos de software libre

El presente proyecto se integra con los previamente desarrollados por el proyecto Libre Software Engineering, perteneciente al GSyC (Grupo de Sistemas y Comunicaciones) de la Universidad Rey Juan Carlos. Sus trabajos se centran en la medición cuantitativa de las características del software libre, con especial atención en las herramientas utilizadas en su desarrollo, los agentes que en él intervienen y los métodos seguidos; todo ello bajo una perspectiva ingenieril, y no tanto social o económica.

En la web se encuentran, aparte de documentos relacionados con su actividad, una relación de las herramientas usadas para recopilar la información sobre el proceso de desarrollo de software libre, junto con los resultados obtenidos de su aplicación sobre un conjunto representativo de proyectos de software libre.

2.2. Recogida de datos

Para poder realizar esta representación de datos de la que hemos estado hablando, en primer lugar necesitamos dichos datos. Estos datos son proporcionados por las empresas normalmente

de forma abierta, siguiendo en algunos casos algunas de las siguientes herramientas para el proceso.

2.2.1. MetricsGrimoire

MetricsGrimoire es un conjunto de herramientas diseñadas para la obtención de datos procedentes de los repositorios relacionados con el desarrollo de software (listas de correo, seguimientos de problemas, modificaciones en el código fuente, etc). Los datos y metadatos sobre el proceso de desarrollo de software son recuperados desde estos repositorios, organizados y guardados en bases SQL que pueden ser posteriormente recuperados para el diseño de recopilación de actividad o búsqueda de patrones.

MetricsGrimoire se nutre de multitud de tipos de repositorios, incluyendo entre estos GitHub. Actualmente se está utilizando para el desarrollo de multitud de tipos de proyectos para posteriormente realizar su visualización como ocurre en VizGrimoire, el cual es posible ver en DashBoards proporcionados por Bitergia por ejemplo.

Su origen reside en Libresoft, un grupo de investigación de la Universidad Rey Juan Carlos. Después de muchos años de desarrollo, una pequeña comunidad de desarrolladores comenzaron con este proyecto, ahora ampliado a desarrolladores procedentes de todo el mundo.

2.2.2. vizGrimoire

VizGrimoire es un conjunto de herramientas orientadas al análisis y visualización de datos sobre el desarrollo de software. Actualmente, está enfocado en la producción de datos provenientes de MetricsGrimoire.

VizGrimoire está promovido por Bitergia, una compañía que promueve el desarrollo de servicios de análisis de software. Dicho proyecto es totalmente abierto, y cualquiera que lo desee es bienvenido a participar.

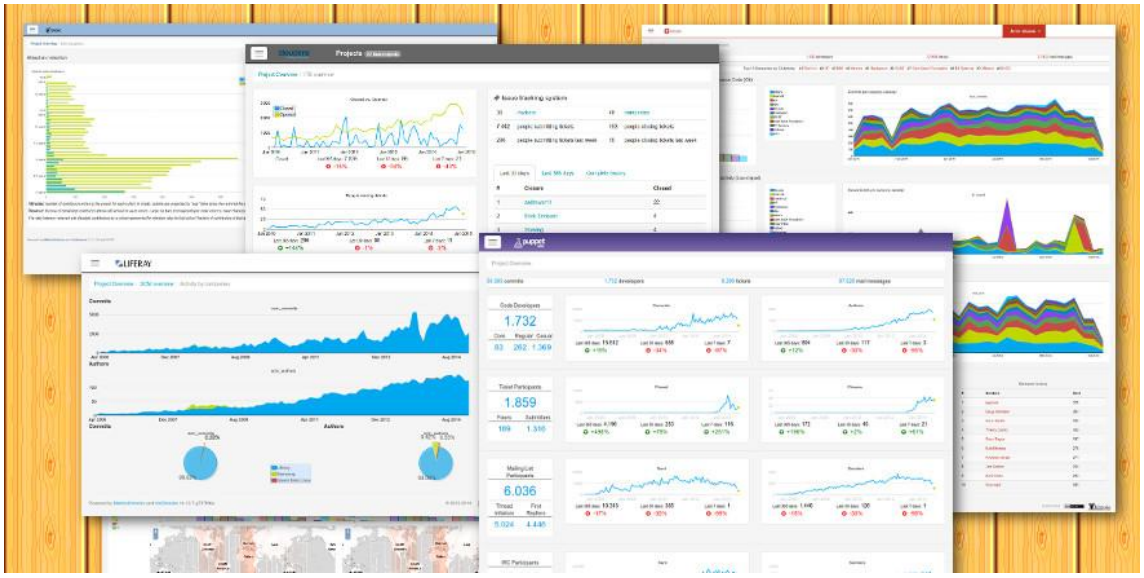

Figura 2.1: Ejemplos de vizGrimoire

2.3. Otros ámbitos relacionados

En el ámbito de desarrollo de software así como su seguimiento existen otras herramientas y grupos de investigadores ó programadores metidos. A continuación pasearemos no solo por estas herramientas, si no por algunos ejemplos de DashBoards.

2.3.1. OpenStrack

OpenStack es un proyecto de computación en la nube para proporcionar una infraestructura como servicio (IaaS).

Es un software libre y de código abierto distribuido bajo los términos de la licencia Apache. El proyecto está gestionado por la Fundación OpenStack, una persona jurídica sin fines de lucro creada en septiembre de 2012 para promover el software OpenStack y su comunidad.

Más de 200 empresas se unieron al proyecto entre las que destacan AMD, Brocade Communications Systems, Canonical, Cisco, Dell, Ericsson, Groupe Bull, HP, IBM, InkTank, Intel, NEC, Rackspace Hosting, Red Hat, SUSE Linux, VMware y Yahoo!.

La tecnología consiste en una serie de proyectos relacionados entre sí que controlan estanques de control de procesamiento, almacenamiento y recursos de red a través de un centro de datos, todos administrados a través de un panel de control que permite a los administradores

controlar mientras potencia a sus usuarios proveyendo los recursos a través de una interfaz web.

La comunidad OpenStack colabora en torno a un ciclo de lanzamiento con hitos de desarrollo de frecuencia semestral. Durante la fase de planificación de cada lanzamiento, la comunidad se reúne para la Cumbre de Diseño OpenStack para facilitar sesiones de trabajo para desarrolladores y armar planes a futuro.

Los datos que se han utilizado para la muestra de este proyecto provienen de dicha empresa, pero cabe destacar que el proyecto es aplicable a cualquier otro ámbito de cualquier otra empresa, ya que la personalización de las gráficas se basa en el hecho de que los datos de producción poseen las mismas métricas de base.

2.3.2. Freeboard

Freeboard es una aplicación basada en ingeniería HTML para la creación de DashBoards. Además de un buen motor de diseño, proporciona una arquitectura para la creación de fuentes de datos (recuperación de datos) y widgets (presentación de los datos) uniendo las dos partes.

Una de sus grandes características es su capacidad para funcionar en un navegador cualquiera como una simple aplicación web estática sin la necesidad de un servidor. Esto lo hace muy atractivo para dispositivos que poseen una capacidad limitada para servir páginas web complejas y dinámicas.

Figura 2.2: Ejemplo de Freeboard

2.3.3. GitHub

GitHub es una forja para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por GitHub, Inc. (anteriormente conocida como Logical Awesome). Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago. Algunas de sus características son:

- Wiki para cada proyecto
- Página web para cada proyecto
- Gráfico para ver cómo los desarrolladores trabajan en sus repositorios y bifurcaciones del proyecto
- Funcionalidades como si se tratase de una red social, como por ejemplo: seguidores
- Bueno para trabajo colaborativo entre programadores

De este modo GitHub se convierte en un almacén de resultados de desarrollo de software donde muchas empresas sitúan sus resultados al alcance de todos. Del mismo modo existen

bibliotecas tales como Hello.js que nos permiten hacer uso de GitHub como base de datos, utilizando ”gh-pages” como un método con el que poder presentar nuestra aplicación.

2.3.4. Many Eyes

IBM Many Eyes es una comunidad web que conecta a expertos, profesionales, académicos y aficionados en visualización ofreciendo una tecnología y experiencia de compartir con el resto de la comunidad democratizando la representación de datos.

Para participar en dicha web, no es necesario tener conocimientos técnicos o de programación, por lo que casi todo el mundo tiene la capacidad de crear representaciones de los datos siguiendo tres simples pasos:

- Subir el conjunto de datos de forma pública, en un formato que puede ser simple, como un archivo de texto o una hoja de cálculo.

- Elegir una de la gran variedad de plantillas de visualización que ofrece Many Eyes.

- Dar rienda suelta a su imaginación para desarrollarla y posteriormente compartirla a través de internet, usando para ello la capacidad que ofrece Many Eyes para incrustar la visualización en un blog, Facebook o Twitter.

Figura 2.3: Ejemplos de Many Eyes

Capítulo 3

Tecnologías utilizadas

En esta sección daremos un pequeño paseo por las tecnologías que hemos utilizado y explorado para el desarrollo de este proyecto. Las mencionaremos e intentaremos entrar levemente en profundidad para conocerlas con más detalle.

3.1. HTML5

3.1.1. Concepto

HTML5 es la actualización de HTML, el lenguaje en el que es creada la web. HTML5 también es un termino de marketing para agrupar las nuevas tecnologías de desarrollo de aplicaciones web: HTML5, CSS3 y nuevas capacidades de JavaScript.

La versión anterior y más usada de HTML, HTML4, carece de características necesarias para la creación de aplicaciones modernas basadas en un navegador. El uso fuerte de JavaScript ha ayudado a mejorar esto, gracias a frameworks como jQuery[1],jQuery UI[2], entre otros.

Flash en especial ha sido usado en reemplazo de HTML para desarrollar web apps que superaran las habilidades de un navegador: Audio, video, webcams, micrófonos, datos binarios, animaciones vectoriales, componentes de interfaz complejos, entre muchas otras cosas. Ahora HTML5 es capaz de hacer esto sin necesidad de plugins y con una gran compatibilidad entre navegadores.

[1] http://jquery.com/

[2] http://jqueryui.com/

3.1.2. Nuevas etiquetas

HTML4 y HTML5 son 100 % compatibles entre sí. Todo el código que tienes en HTML normal seguirá funcionando sin problemas en HTML5. Para empezar a usar HTML5 lo único que tienes que hacer es colocar este DOCTYPE antes de la etiqueta $< html >$:

$<!DOCTYPEhtml>$

Es un DOCTYPE mucho más simplificado que XHTML (cuyas reglas siguen siendo usadas) y te permite usar todas las habilidades de HTML5 sin que nada de lo que ya tienes programado deje de funcionar. Las principales etiquetas HTML5 nuevas no tienen una representación especial en pantalla. Todas se comportan como un $< div >$ o un $< span >$. Pero cada una tiene un significado semántico superior a un simple div o span.

Las etiquetas semánticas, a pesar de ser claves para posicionamiento en buscadores y buen desarrollo web, no son la razón por la que todo el mundo habla de HTML5. Video, audio y animación vectorial están en la lista de prioridades y en la boca de todas las personas que evangelizan su uso. Específicamente, las nuevas etiquetas son:

$< video >$

Inserta video sin necesidad de plugins. Es muy fácil usarla, pero cada navegador soporta codecs diferentes de video, lo que hace necesario recodificar un video en múltiples codecs. En un futuro capítulo hablamos un poco del drama que este tag está generando.

$< audio >$

Lo mismo que video, pero sin video. Puede usar múltiples formatos, en especial mp3, pero también depende del navegador.

$< input >$

Input ya existía como la etiqueta para insertar cajas de texto y botones. Ahora es más poderosa, con la capacidad de insertar cajas tipo ”email” que se autovalidan, calendarios tipo ”date”, sliders, números, entre otras.

$< canvas >$

Un área de dibujo vectorial y de bitmaps con JavaScript. Es un API de dibujo entero para JavaScript.Hágase especial énfasis en esta nueva etiqueta pues será básica a la hora de desarrollar nuestra aplicación. Si bien está abierta a nuevas posibilidades para el uso de los widgets que se presentan en esta memoria, su aplicación original está orientada al seguimiento del desarrollo de un grupo de software. Para hacer esto posible haremos uso de gráficas para la representación

de los datos que nos permitan esto.

$< svg >$

Una etiqueta, igual que ¡img¿, para insertar dibujos y animaciones vectoriales al estilo de Flash. Todo basado en el estándar abierto SVG (Scalable Vector Graphics), derivado de XML.

3.2. CSS/CSS3

3.2.1. Concepto

Hoja de estilo en cascada o CSS (siglas en inglés de cascading style sheets) es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML).

El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

La información de estilo puede ser definida en un documento separado o en el mismo documento HTML. En este último caso podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo style.

El avance de la tecnología ha llevado a CSS hasta su desarrollo en CSS3, dónde se establecen nuevas posibilidades, como la creación de animaciones, entornos 3d y mucho más. A continuación paso a listar su progreso a lo largo de sus diversas versiones.

3.2.2. CSS 1

La primera especificación oficial de CSS, recomendada por la W3C fue CSS1, publicada en diciembre 1996, y abandonada en abril de 2008. Algunas de las funcionalidades que ofrece son:

- Propiedades de las fuentes, como tipo, tamaño, énfasis...
- Color de texto, fondos, bordes u otros elementos.
- Atributos del texto, como espaciado entre palabras, letras, líneas, etcétera.

- Alineación de textos, imágenes, tablas u otros.
- Propiedades de caja, como margen, borde, relleno o espaciado.
- Propiedades de identificación y presentación de listas.

3.2.3. CSS 2

La especificación CSS2 fue desarrollada por la W3C y publicada como recomendación en mayo de 1998, y abandonada en abril de 2008. Como ampliación de CSS1, se ofrecieron, entre otras:

- Las funcionalidades propias de las capas (¡div¿) como de posicionamiento relativo/absoluto/fijo, niveles (z-index), etcétera.
- El concepto de ”media types”
- Soporte para las hojas de estilo auditivas
- Texto bidireccional, sombras, etcétera.

3.2.4. CSS 2.1

La primera revisión de CSS2, usualmente conocida como ”CSS 2.1”, corrige algunos errores encontrados en CSS2, elimina funcionalidades poco soportadas o inoperables en los navegadores y añade alguna nueva especificación.

De acuerdo al sistema de estandarización técnica de las especificaciones, CSS2.1 tuvo el estatus de ”candidato” (candidate recommendation) durante varios años, pero la propuesta fue rechazada en junio de 2005; en junio de 2007 fue propuesta una nueva versión candidata, y ésta actualizada en 2009, pero en diciembre de 2010 fue nuevamente rechazada. En abril de 2011, CSS 2.1 volvió a ser propuesta como candidata , y después de ser revisada por el W3C Advisory Committee, fue finalmente publicada como recomendación oficial el 7 de junio de 2011.

3.2.5. CSS3

A diferencia de CSS2, que fue una gran especificación que definía varias funcionalidades, CSS3 está dividida en varios documentos separados, llamados ”módulos”.

Cada módulo añade nuevas funcionalidades a las definidas en CSS2, de manera que se preservan las anteriores para mantener la compatibilidad. Los trabajos en el CSS3, comenzaron a la vez que se publicó la recomendación oficial de CSS2, y los primeros borradores de CSS3 fueron liberados en junio de 1998.

Debido a la modularización del CSS3, diferentes módulos pueden encontrarse en diferentes estados de su desarrollo de forma que a fechas de noviembre de 2011, hay alrededor de cincuenta módulos publicados, tres de ellos se convirtieron en recomendaciones oficiales de la W3C en 2011: ”Selectores”, ”Espacios de nombres” y ”Color”.

Algunos módulos, como ”Fondos y colores”, ”Consultas de medios” o ”Diseños multicolumna” están en fase de ”candidatos”, y considerados como razonablemente estables, a finales de 2011, y sus implementaciones en los diferentes navegadores son señaladas con los prefijos del motor del mismo.

A continuación podemos ver un ejemplo de lo que ofrece una de las versiones de css, en este caso css3d:

https://www.youtube.com/watch?v=nPEYdw2Ssa8

3.3. JavaScript

3.3.1. Concepto

JavaScript (abreviado comúnmente ”JS”) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

Una cuarta edición está en desarrollo e incluirá nuevas características tales como paquetes, espacio de nombres y definición explícita de clases.

3.3.2. Características

Imperativo y estructurado

JavaScript es compatible con gran parte de la estructura de programación de C (por ejemplo, sentencias if, bucles for, sentencias switch, etc.). Con una salvedad, en parte: en C, el ámbito de las variables alcanza al bloque en el cual fueron definidas; sin embargo JavaScript no es compatible con esto, puesto que el ámbito de las variables es el de la función en la cual fueron declaradas. Esto cambia con la versión de JavaScript 1.7, ya que añade compatibilidad con block scoping por medio de la palabra clave let.

Como en C, JavaScript hace distinción entre expresiones y sentencias. Una diferencia sintáctica con respecto a C es la inserción automática de punto y coma, es decir, en JavaScript los puntos y coma que finalizan una sentencia pueden ser omitidos.

Dinámico

Como en la mayoría de lenguajes de scripting, el tipo está asociado al valor, no a la variable. Por ejemplo, una variable x en un momento dado puede estar ligada a un número y más adelante, religada a una cadena. JavaScript es compatible con varias formas de comprobar el tipo de un objeto, incluyendo duck typing. Una forma de saberlo es por medio de la palabra clave typeof.

Objetual

JavaScript está formado casi en su totalidad por objetos. Los objetos en JavaScript son arrays asociativos, mejorados con la inclusión de prototipos.

Los nombres de las propiedades de los objetos son claves de tipo cadena: obj.x = 10 y obj['x'] = 10 son equivalentes, siendo la notación con punto azúcar sintáctico. Las propiedades y

sus valores pueden ser creados, cambiados o eliminados en tiempo de ejecución. La mayoría de propiedades de un objeto (y aquellas que son incluidas por la cadena de la herencia prototípica) pueden ser enumeradas a por medio de la instrucción de bucle for... in. JavaScript tiene un pequeño número de objetos predefinidos como son Function y Date.

Evaluación en tiempo de ejecución

JavaScript incluye la función eval que permite evaluar expresiones como expresadas como cadenas en tiempo de ejecución. Por ello se recomienda que eval sea utilizado con precaución y que se opte por utilizar la función JSON.parse() en la medida de lo posible, pues resulta mucho más segura.

Funcional

A las funciones se les suele llamar ciudadanos de primera clase; son objetos en sí mismos. Como tal, poseen propiedades y métodos, como .call() y .bind().

Una función anidada es una función definida dentro de otra. Esta es creada cada vez que la función externa es invocada. Además, cada función creada forma una clausura; es el resultado de evaluar un ámbito conteniendo en una o más variables dependientes de otro ámbito externo, incluyendo constantes, variables locales y argumentos de la función externa llamante. El resultado de la evaluación de dicha clausura forma parte del estado interno de cada objeto función, incluso después de que la función exterior concluya su evaluación.

Prototípico

JavaScript usa prototipos en vez de clases para el uso de herencia. Es posible llegar a emular muchas de las características que proporcionan las clases en lenguajes orientados a objetos tradicionales por medio de prototipos en JavaScript.

Una diferencia notable con su primo (aún que solo en nombre) Java, es que JavaScript impide la creación de variables privadas dentro de clases que después se puedan heredar. Esto implica tener sumo cuidado con la caracterización de las propiedades de un prototipo.

Las funciones también se comportan como constructores. Prefijar una llamada a la función con la palabra clave new crear una nueva instancia de un prototipo, que heredan propiedades y métodos del constructor (incluidas las propiedades del prototipo de Object). ECMAScript 5 ofrece el método Object.create, permitiendo la creación explícita de una instancia sin tener que heredar automáticamente del prototipo de Object (en entornos antiguos puede aparecer el prototipo del objeto creado como null). La propiedad prototype del constructor determina el objeto

usado para el prototipo interno de los nuevos objetos creados. Se pueden añadir nuevos métodos modificando el prototipo del objeto usado como constructor. Constructores predefinidos en JavaScript, como Array u Object, también tienen prototipos que pueden ser modificados. Aunque esto sea posible se considera una mala práctica modificar el prototipo de Object ya que la mayoría de los objetos en JavaScript heredan los métodos y propiedades del objeto prototype, objetos los cuales pueden esperar que estos no hayan sido modificados.

3.4. Jquery

3.4.1. Concepto

jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC. jQuery es la biblioteca de JavaScript más utilizada.

jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privados. jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Las empresas Microsoft y Nokia anunciaron que incluirán la biblioteca en sus plataformas. Microsoft la añadirá en su IDE Visual Studio4 y la usará junto con los frameworks ASP.NET AJAX y ASP.NET MVC, mientras que Nokia los integrará con su plataforma Web Run-Time.

3.4.2. Características

- Selección de elementos DOM. posicionamiento relativo/absoluto/fijo, niveles (z-index), etcétera.

- Interactividad y modificaciones del árbol DOM, incluyendo soporte para CSS 1-3 y un plugin básico de XPath.

- Eventos.
- Manipulación de la hoja de estilos CSS.
- Efectos y animaciones.
- Animaciones personalizadas.
- AJAX.
- Soporta extensiones.
- Utilidades varias como obtener información del navegador, operar con objetos y vectores, funciones para rutinas comunes, etc.
- Compatible con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+.

3.5. Bootstrap

Es una colección de herramientas para creación de sitios web y aplicaciones web. Contiene HTML y CSS diseñando plantillas con tipografía propia, botones, formatos, barras de navegación y otros componentes de interfaz. En Junio de 2014 supuso el primer proyecto en cabeza en GitHub con más de 73000 stars y más de 27000 forks. Su uso se extiende a lugares como NASA o MSNBC. De este modo constituye una de las herramientas para la creación de entornos web más utilizadas, y con ello amplía la comunidad que se ha interesado por ella. Gracias a este interés y avance comunitario podemos encontrar por la red multitud de plantillas gratuitas que nos pueden facilitar el entorno de nuestra aplicación, así como otras muchas de pago de las que podemos hacer uso del mismo modo.A continuación podemos ver el ejemplo de una de estas plantillas para hacernos una idea rápida sobre lo que estamos hablando.

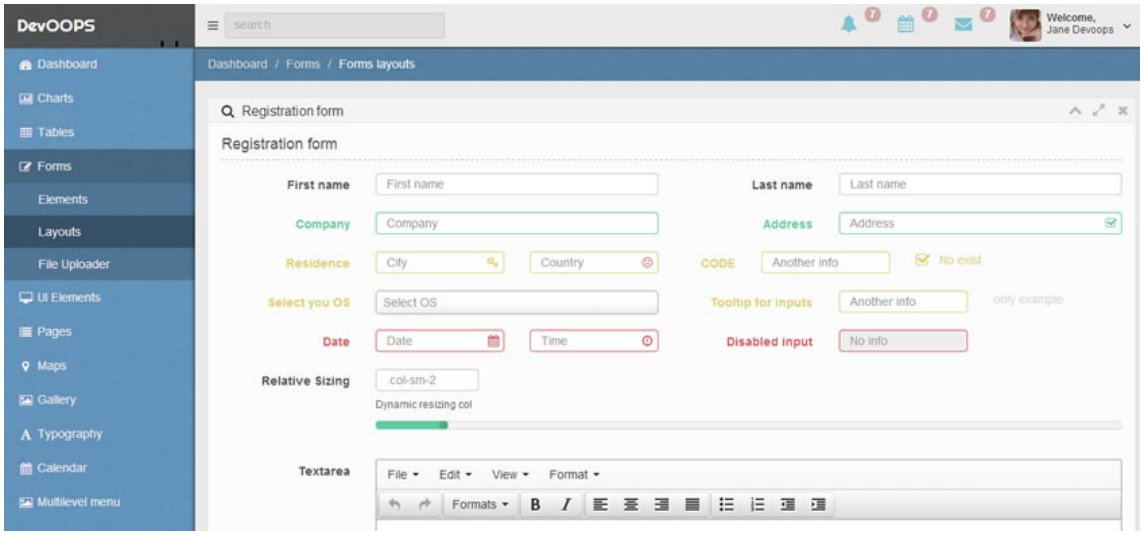

Figura 3.1: Plantilla Bootstrap

3.6. Python

3.6.1. Concepto

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable. Un claro ejemplo de esto lo podemos ver en la programación de aprendizaje robótico de nuestra universidad en la que se hace uso de la plataforma jdrobot. Para un mejor aprendizaje del uso de robots como el pioneer o kobuki, existen programas de ejecución de estos mismos robots programados para que la inteligencia artificial del mismo se pueda programar en LUA o en Python. De esta forma el robot es capaz de realizar distintas operaciones simplemente compilando el programa con un nuevo script de IA.

3.6.2. Historia

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python. Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como Benevolente Dictador Vitalicio (en inglés: Benevolent Dictator for Life, BDFL).

En 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: str, list, dict, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3; van Rossum describe el módulo como ”una de las mayores unidades de programación de Python”. El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula else. En el año 1994 se formó comp.lang.python, el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje.

Python alcanzó la versión 1.0 en enero de 1994. Una característica de este lanzamiento fueron las herramientas de la programación funcional: lambda, reduce, filter y map. Van Rossum explicó que ”hace 12 años, Python adquirió lambda, reduce(), filter() y map(), cortesía de un pirata informático de Lisp que las extrañaba y que envió parches”. El donante fue Amrit Prem; no se hace ninguna mención específica de cualquier herencia de Lisp en las notas de lanzamiento. La última versión liberada proveniente de CWI fue Python 1.2. En 1995, van Rossum continuó su trabajo en Python en la Corporation for National Research Initiatives (CNRI) en Reston, Virginia, donde lanzó varias versiones del software.

Durante su estancia en CNRI, van Rossum lanzó la iniciativa Computer Programming for Everybody (CP4E), con el fin de hacer la programación más accesible a más gente, con un nivel de ’alfabetización’ básico en lenguajes de programación, similar a la alfabetización básica en inglés y habilidades matemáticas necesarias por muchos trabajadores. Python tuvo un papel crucial en este proceso: debido a su orientación hacia una sintaxis limpia, ya era idóneo, y las metas de CP4E presentaban similitudes con su predecesor, ABC. El proyecto fue patrocinado por DARPA.7 En el año 2007, el proyecto CP4E está inactivo, y mientras Python intenta ser fácil

de aprender y no muy arcano en su sintaxis y semántica, alcanzando a los no-programadores, no es una preocupación activa. En el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo BeOpen PythonLabs. CNRI pidió que la versión 1.6 fuera pública, continuando su desarrollo hasta que el equipo de desarrollo abandonó CNRI; su programa de lanzamiento y el de la versión 2.0 tenían una significativa cantidad de traslapo.9 Python 2.0 fue el primer y único lanzamiento de BeOpen.com. Después que Python 2.0 fuera publicado por BeOpen.com, Guido van Rossum y los otros desarrolladores de PythonLabs se unieron en Digital Creations. Python 2.0 tomó una característica mayor del lenguaje de programación funcional Haskell: listas por comprensión. La sintaxis de Python para esta construcción es muy similar a la de Haskell, salvo por la preferencia de los caracteres de puntuación en Haskell, y la preferencia de Python por palabras claves alfabéticas. Python 2.0 introdujo además un sistema de recolección de basura capaz de recolectar referencias cíclicas.

Posterior a este doble lanzamiento, y después que van Rossum dejó CNRI para trabajar con desarrolladores de software comercial, quedó claro que la opción de usar Python con software disponible bajo GNU GPL era muy deseable. La licencia usada entonces, la Python License, incluía una cláusula estipulando que la licencia estaba gobernada por el estado de Virginia, por lo que, bajo la óptica de los abogados de Free Software Foundation (FSF), se hacía incompatible con GPL. CNRI y FSF se relacionaron para cambiar la licencia de software libre de Python para hacerla compatible con GPL. En el año 2001, van Rossum fue premiado con FSF Award for the Advancement of Free Software.

Python 1.6.1 es esencialmente el mismo que Python 1.6, con unos pocos arreglos de bugs, y con una nueva licencia compatible con GPL.1 Código Python con coloreado de sintaxis.

Python 2.1 fue un trabajo derivado de Python 1.6.1, así como también de Python 2.0. Su licencia fue renombrada a: Python Software Foundation License. Todo el código, documentación y especificaciones añadidas, desde la fecha del lanzamiento de la versión alfa de Python 2.1, tiene como dueño a Python Software Foundation (PSF), una organización sin ánimo de lucro fundada en el año 2001, tomando como modelo la Apache Software Foundation. Incluido en este lanzamiento fue una implementación del scoping más parecida a las reglas de static scoping (del cual Scheme es el originador).

Una innovación mayor en Python 2.2 fue la unificación de los tipos en Python (tipos escritos en C), y clases (tipos escritos en Python) dentro de una jerarquía. Esa unificación logró un

modelo de objetos de Python puro y consistente. También fueron agregados los generadores que fueron inspirados por el lenguaje Icon.

Las adiciones a la biblioteca estándar de Python y las decisiones sintácticas fueron influenciadas fuertemente por Java en algunos casos: el package logging, introducido en la versión 2.3, está basado en log4j; el parser SAX, introducido en 2.0; el package threading,14 cuya clase Thread expone un subconjunto de la interfaz de la clase homónima en Java.

3.6.3. Características

- Tipado dinámico y fuertemente tipado. No requiere que nos tomemos la molestia de declarar variables ya que reconoce su tipo desde que se asigna un valor por primera vez. Eso sí, durante el tiempo de vida de una variable ésta sólo puede pertenecer a un tipo.
- Proporciona estructuras de datos flexibles y sencillas de usar como listas, diccionarios y strings, como parte intrínseca al lenguaje. Para procesar cada uno de estos tipos de objeto, incorpora un conjunto de operaciones que ahorrarán tiempo y esfuerzo al implementar tareas de uso habitual como ordenaciones, búsquedas, etc.
- Python posee una amplia colección de librerías dedicadas a tareas específicas.
- Administra automáticamente la gestión de memoria. Se encarga de solicitar memoria en la creación de objetos y de liberarla cuando no van a volver a ser utilizados.
- Permite la construcción de sistemas de gran tamaño al incorporar herramientas como clases, módulos, y excepciones.
- A los programas escritos en Python pueden integrarse componentes escritos en otros lenguajes. Por ejemplo, mediante el API de Python/C, un código Python puede extender su funcionalidad incorporando componentes escritos en C o C++, lo cual convierte a Python en un lenguaje de prototipado rápido: las aplicaciones pueden ser implementadas en primera instancia con Python para aumentar su velocidad de desarrollo y posteriormente ciertas partes reescritas en C por motivos de eficiencia.

Con Python es perfectamente viable el desarrollo de proyectos software de gran entidad, ejemplo de ello son el servidor de aplicaciones *Zope* y el sistema de intercambio de ficheros *BitTorrent*, incluyendo al propio *Mailman*.

Otro aspecto que nos interesa particularmente es la incorporación, dentro de la completa librería que ofrece, de módulos que manejan estándares comunes en Internet (HTML, FTP, XML, HTTP, ...) y APIs para la comunicación con bases de datos (para gestores como PostgreSQL, Oracle o MySQL).

A continuación hacemos mención a otro recurso que Python nos brinda y que será de gran utilidad durante la implementación de este proyecto: las expresiones regulares.

3.6.4. Expresiones regulares

Estrictamente aplicado a este campo, una expresión regular es un patrón escrito en una sintaxis compacta y bastante críptica que se corresponde con un conjunto de cadenas de caracteres (en adelante *strings*). Python proporciona un módulo especializado para procesarlas.

Las expresiones regulares se componen de caracteres especiales, que permiten desde representar cualquier carácter salvo fin de línea ('.') o la repetición en una o más ocasiones de la expresión regular predecesora ('+'), hasta indicar que existe correspondencia sólo si una expresión regular va inmediatamente precedida de otra en concreto. Así pues, mediante la combinación de metacaracteres se puede simbolizar un número de conjuntos de strings prácticamente ilimitado.

Una vez definido un patrón pueden realizarse diferentes operaciones aplicadas sobre un string. A través del módulo *re* de Python se tiene acceso a métodos que buscan posibles coincidencias del patrón en del string, permitiendo, si la comparación tuvo éxito, obtener las posiciones de inicio y final del substring coincidente. Existen otros métodos que segmentan el string en rodajas delimitadas por el patrón o sustituyen las partes del string coincidentes con el patrón. El módulo aporta, además, facilidades para el manejo de grupos, que delimitan zonas dentro del patrón que podrán ser recuperadas a posteriori. Esto será de gran utilidad en caso de que sea necesario obtener un texto cuyo entorno sea cambiante.

Como puede desprenderse de lo comentado, las expresiones regulares aportan una gran potencia al manejo de strings dentro de un lenguaje de programación. A pesar de ello, emplearlas para tareas en las que no son estrictamente necesarias, aparte de ser un mal hábito, va en detrimento de la claridad del código, complica sobremanera su depuración y repercute negativamente sobre su eficiencia.

3.7. Django

3.7.1. Concepto

Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el paradigma conocido como Model Template View. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt.

En junio del 2008 fue anunciado que la recién formada Django Software Foundation se haría cargo de Django en el futuro.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

3.7.2. Características

Al igual que Ruby on Rails, otro popular framework de código abierto, Django se usó en producción durante un tiempo antes de que se liberara al público; fue desarrollado por Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss y Wilson Miner mientras trabajaban en World Online, y originalmente se utilizó para administrar tres sitios web de noticias: The Lawrence Journal-World, lawrence.com y KUsports.com.

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos, que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios

(incluyendo una asignación detallada de permisos).

La distribución principal de Django también aglutina aplicaciones que proporcionan un sistema de comentarios, herramientas para sindicar contenido via RSS y/o Atom, ”páginas planas” que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs. Otras características de Django son:

- Un mapeador objeto-relacional
- Aplicaciones ”enchufables” que pueden instalarse en cualquier página gestionada con Django.
- Una API de base de datos robusta.
- Un sistema incorporado de ”vistas genéricas” que ahorra tener que escribir la lógica de ciertas tareas comunes.
- Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Un despachador de URLs basado en expresiones regulares.
- Un sistema ”middleware” para desarrollar características adicionales; por ejemplo, la distribución principal de Django incluye componentes middleware que proporcionan cacheo, compresión de la salida, normalización de URLs, protección CSRF y soporte de sesiones.
- Soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.
- Documentación incorporada accesible a través de la aplicación administrativa (incluyendo documentación generada automáticamente de los modelos y las bibliotecas de plantillas añadidas por las aplicaciones).

3.8. Módulo SQLite

3.8.1. Concepto

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, contenida en una relativamente pequeña (275 kiB) biblioteca escrita en C. SQLite es un proyecto de

dominio público1 creado por D. Richard Hipp.

A diferencia de los sistema de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos.

El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

En su versión 3, SQLite permite bases de datos de hasta 2 Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB.

El autor de SQLite ofrece formación, contratos de soporte técnico y características adicionales como compresión y cifrado.

3.8.2. Características

La biblioteca implementa la mayor parte del estándar SQL-92, incluyendo transacciones de base de datos atómicas, consistencia de base de datos, aislamiento, y durabilidad (ACID), triggers y la mayor parte de las consultas complejas.

SQLite usa un sistema de tipos inusual. En lugar de asignar un tipo a una columna como en la mayor parte de los sistemas de bases de datos SQL, los tipos se asignan a los valores individuales. Por ejemplo, se puede insertar un string en una columna de tipo entero (a pesar de que SQLite tratará en primera instancia de convertir la cadena en un entero). Algunos usuarios consideran esto como una innovación que hace que la base de datos sea mucho más útil, sobre todo al ser utilizada desde un lenguaje de scripting de tipos dinámicos. Otros usuarios lo ven como un gran inconveniente, ya que la técnica no es portable a otras bases de datos SQL. SQLite no trataba de transformar los datos al tipo de la columna hasta la versión 3.

Varios procesos o hilos pueden acceder a la misma base de datos sin problemas. Varios accesos de lectura pueden ser servidos en paralelo. Un acceso de escritura sólo puede ser servido si no se está sirviendo ningún otro acceso concurrentemente. En caso contrario, el acceso de escritura falla devolviendo un código de error (o puede automáticamente reintentarse hasta

que expira un tiempo de expiración configurable). Esta situación de acceso concurrente podría cambiar cuando se está trabajando con tablas temporales. Sin embargo, podría producirse un interbloqueo debido al multihilo. Este punto fue tratado en la versión 3.3.4, desarrollada el 11 de febrero de 2006.

Existe un programa independiente de nombre sqlite que puede ser utilizado para consultar y gestionar los ficheros de base de datos SQLite. También sirve como ejemplo para la escritura de aplicaciones utilizando la biblioteca SQLite.

3.9. AJAX

3.9.1. Concepto

AJAX es una técnica de de desarrollo WEB, por la cual se pueden crear aplicaciones WEB más rápidas y cómodas para el usuario. Por medio de esta técnica el cliente puede interactuar con el servidor de manera asincrónica, actualizando las páginas, sin necesidad de volver a cargarlas.

Esta técnica, no solo es más cómoda y amigable para el usuario (ya que se asemeja a las aplicaciones de escritorio) sino que además es más rápida, porque cada vez que se necesita actualizar un dato en una página, no es necesario recargarla nuevamente (solo se recarga la sección necesaria de la misma).

AJAX, no es una tecnología en si, sino que es un conjunto de tecnologías aplicadas de manera que logran el resultado explicado anteriormente (es decir, logran AJAX).

AJAX significa Asynchronous JavaScript And XML, y como su nombre lo indica, se trata de la combinación de JavaScript y XML. JavaScript hace las peticiones al servidor, el mismo le devuelve un resultado (response) en XML, y este es procesado por JavaScript para actualizar los datos de la página, sin tener que recargarla por completo (logrando así una interacción asincrónica entre el servidor y el cliente).

3.9.2. Funcionamiento

Para poder utilizar AJAX, con saber un poco de JavaScript y XML es suficiente. Pero si se quiere explotar AJAX al máximo, es necesario saber otras cosas como XHTML, CSS, XSLT,

DOM, JSON, etc.

Para entender como funciona AJAX, primero vamos a hacer un sintético repaso de como funcionan las aplicaciones web ”tradicionales”.

- El cliente le hace una petición (request) HTTP al servidor WEB (generalmente por medio de un navegador WEB).
- El servidor WEB, procesa la petición dinámicamente. Por medio de un lenguaje de lado de servidor (como PHP o Perl).
- Una vez procesada la petición HTTP por el servidor WEB, este le devuelve una respuesta en HTML al cliente (lo que se conoce como ”página WEB”).
- Por medio de esa página generalmente el ciclo vuelve a empezar, ya que el cliente puede hacer otras peticiones HTTP al servidor, que a su vez van a ser procesadas y el servidor WEB va a devolver otros resultados HTML, etc. Por otra parte, del lado del cliente, existen los lenguajes de script, que están embebidos en la página HTML. Por ejemplo, JavaScript, es el lenguaje de script que soportan la mayoría de los navegadores (hay otras tecnologías client-side que pueden utilizarse, como VBScript o JScript, pero hay que recordar que JavaScript fue adoptado como estándar en la ECMA y es soportado por prácticamente todos los navegadores).

Con AJAX, cuando el cliente hace una petición HTTP al servidor, la hace por medio de JavaScript. El servidor procesa la petición y en vez de devolverle al cliente una página HTML, le devuelve un resultado en XML (no necesariamente, ya que también podría ser un resultado en JSON), que es procesado por JavaScript, y este actualiza solo las secciones de la página necesarias (sin tener que cargar una nueva página).

- El cliente por medio del navegador hace produce algún evento. Este evento es procesado por JavaScript (o alguna otra tecnología client-side) y le envía al servidor WEB una petición HTTP.
- El servidor WEB, procesa la petición como siempre, pero devolviendo el resultado en XML.

- Este resultado es procesado por JavaScript. Que recarga las secciones de la página necesarias para mostrar el resultado al usuario.
- Por medio de esta misma página, el ciclo comienza de nuevo. Sin haberse tenido que recargar la página.

3.9.3. Características

Ventajas

- Las páginas no se recargan constantemente.
- El tiempo de espera es menor.
- Se pueden lograr cosas que sin AJAX definitivamente no se podrían hacer, como el conocido Google Maps por ejemplo.

Desventajas

- Falta de integración con el botón ”retroceder” de los navegadores. Esto se debe a que siempre estamos en la misma página (no la recargamos). Y algunas veces puede llegar a confundir al usuario.
- Es necesario que el navegador soporte y tenga habilitado JavaScript. No es una gran desventaja, ya que casi todos los navegadores modernos soportan JavaScript.
- Al tener que ejecutar más código del lado del cliente, puede enlentecerse el rendimiento de la máquina del cliente. Por eso debe usarse AJAX con moderación.
- Al no recargar las páginas, y siempre estar en la misma, no se tiene una URL a la cual poder referirse, en caso de querer recomendar la página, o volver a esa página. Por eso debe saberse cuando usar AJAX y cuando no.

3.10. Highcharts

Highcharts es una biblioteca gráfica escrita puramente en JavaScript, ofreciendo una manera fácil de añadir gráficos a tu aplicación web. Highcharts actualmente soporta muchos tipos de

gráficos, incluyendo de line, spline, area, areaspline, columns, bar, pie, scatter, bubble, gauge y polar chart. Muchos de estos tipos de gráficos pueden ser unificados en uno solo, ofreciendo por otro lado además la posibilidad de exportarlos en formatos PNG,JPG,PDF o SVG con un solo click.

Esta biblioteca es capaz de trabajar sobre navegadores modernos usados en tabletas o teléfonos inteligentes, no requiere de instalaciones extras ni hace uso de Flash, Java, servidores de cualquier tipo, PHP ó ASP.NET.

Una de sus grandes características además de las ya mencionadas es que es de código abierto y no está restringida a licencias o pagos. Cualquiera es capaz de descargarse la biblioteca y realizar las modificaciones que necesite en el propio código fuente, otorgando con ello una gran flexibilidad.

Por último mencionar que las opciones que ofrece para crear un gráfico Highcharts no requieren de ninguna clase de habilidad especial en programación. Las opciones son dadas en un objeto con estructura JavaScript, el cual básicamente contiene sus valores y claves conectadas de la misma manera que cualquier objeto que dicho lenguaje.

Figura 3.2: Ejemplo realizado con Highcharts

3.10.1. Gridster

Gridster es un plugin de jQuery que permiten construir un sistema de columnas arrastrable fácil e intuitivo. En él las columnas se ordenan de manera dinámica, y permite la eliminación de elementos de un mismo canvas grister de manera fácil y sencilla.

Pese a ser un proyecto aún en desarrollo, se trata de un plugin extremadamente útil para utilizar en aplicaciones de esta naturaleza, dando un aspecto más dinámico a la aplicación, y dando con ello un extra en configuración de nuestro DashBoard. Nos ofrece por tanto la posibilidad de ordenar los objetos de la manera que nosotros deseemos, y gracias a las funciones que aún posee, obtener los datos de situación de dichos elementos.

En un futuro, se pretende conseguir que los elementos arrastrables sean añadibles o interactúen de un gridster a otro, mejorando de este modo nuestra interfaz de traslado de widgets.

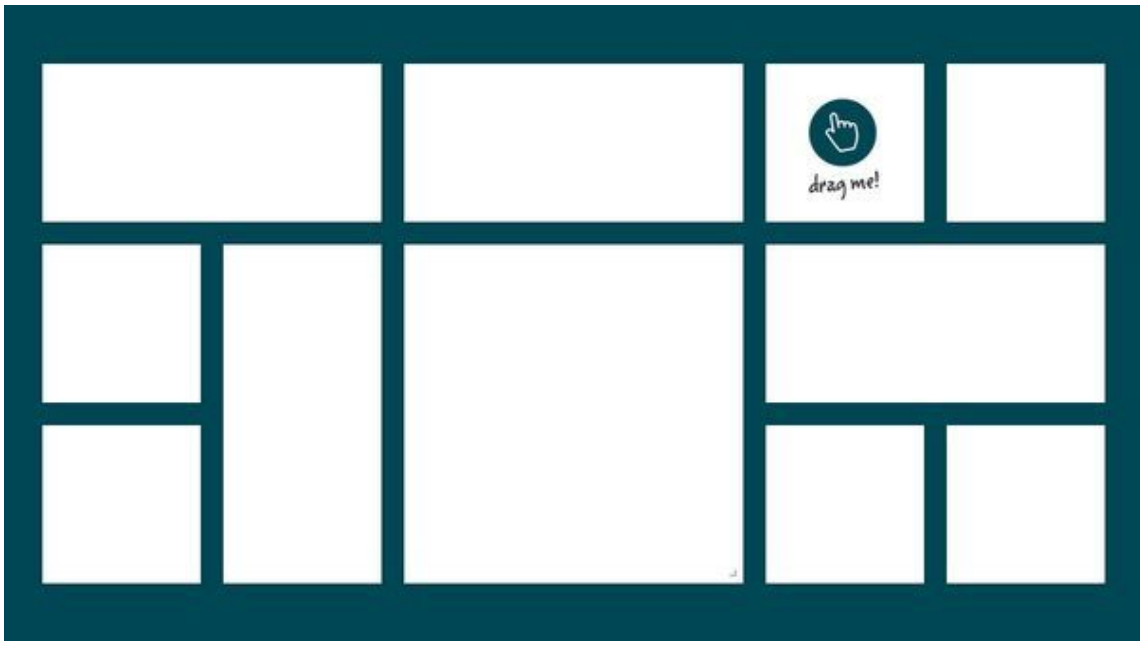

Figura 3.3: Demo de Gridster

Capítulo 4

Desarollo

4.1. Modelo de desarollo

Este capítulo constituye el eje principal de la memoria, en él, analizaremos el uso y desarrollo de la aplicación desde un punto de vista incremental, guiando de la mano al lector por cada una de las fases por las que ha pasado el proyecto como si él mismo lo hubiera desarrollado.

El desarrollo realizado de manera incremental dando lugar a la aparición de diversos prototipos basados en las iteraciones por las que avanzará la creación. Este formato se corresponde a un modelo semejante a scrum que pasamos a explicar a continuación, pero que en nuestro caso, por equipo de desarrollo entenderemos autor del proyecto y por cliente el tutor del mismo.

Modelo scrum:

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es

aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

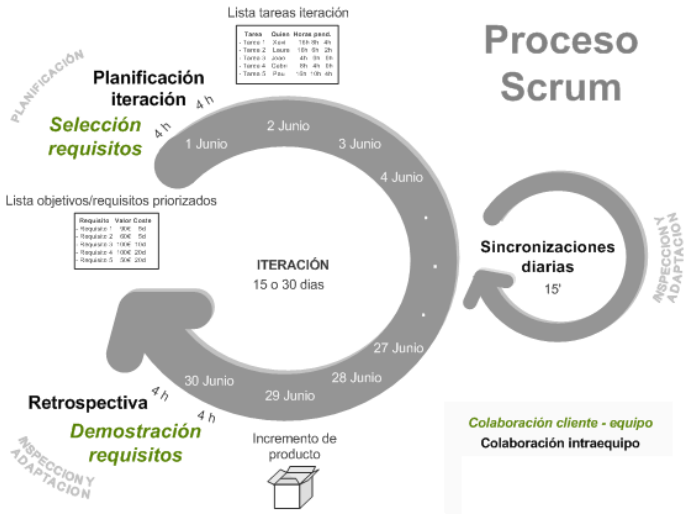

Figura 4.1: Esquema del modelo scrum

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas. De manera regular el cliente puede maximizar la utilidad de lo que se desarrolla y el retorno de inversión mediante la replanificación de objetivos del producto, que realiza durante la iteración con vista a las siguientes iteraciones.

Dicho lo cual, presentamos las siguientes iteraciones de desarrollo:

- **Iteración 0**. Una fase previa que se limitará al estudio de la estructura de los diferentes documentos. En realidad se trata de un paso anterior al desarrollo y, por tanto, no implica un avance en la espiral ni desemboca en un prototipo, pero la vital importancia de esta labor nos obliga a dedicarle un lugar destacado en la evolución del proyecto.

- **Iteración 1**. Una primera fase cuyo objetivo es la creación del DashBoard básico que establecerá una interfaz común para el resto de sus homónimos derivados de éste.
- **Iteración 2**. Una segunda fase que pretende extender la funcionalidad del DashBoard generado en la fase anterior y crear especializaciones adaptadas a las fuentes de información mencionadas de forma recurrente a lo largo del presente documento.
- **Iteración 3**. Tercera fase donde se busca la mejora del DashBoard en cuanto a su estructura, con el objetivo de abrirla más para dar lugar a la implementación de una mayor cantidad de funcionalidades futuras.
- **Iteración 4**. Fase que pretende extender la funcionalidad del DashBoard generado en la fase anterior y crear especializaciones adaptadas a las fuentes de información mencionadas de forma recurrente a lo largo del presente documento.

4.2. Iteración 0. Estudio previo

4.2.1. Introducción

En este primer ciclo se establecieron unas pautas de seguimiento para la realización del TFG. Por otro lado se pusieron sobre la mesa una lista de bibliotecas y recursos de los que poder echar mano, para comenzar así explorando y refrescando conocimientos sobre estos para el correcto desarrollo del mismo.

4.2.2. Desarrollo del ciclo

Como objetivo final en primera instancia se estableció la creación de un DashBoard dinámico en el que pudiéramos elegir los datos que fuéramos a utilizar a la hora de crear nuestro escritorio personalizado de datos. Hasta la fecha existían diversos DashBoards pero estos trabajaban con un tipo de datos muy específico, e impedían la personalización de los resultados obtenidos. Una herramienta útil y aplicable al desarrollo del software que nos permitiría realizar un trabajo interesante y ameno.

Acordado el modelo cascada ya descrito se procedió al comienzo del trabajo de los requisitos que se esperaban para la primera fase:

- Refrescar uso de JavaScript
- Uso de jqueryUI.
- Búsqueda de bibliotecas gráficas de representación para los datos
- Creación de un nuevo proyecto en GitHub
- Uso y experimentación con todo lo anterior para pasar a utilizar los verdaderos datos.

Extras:

- Creación de un proyecto en pivotal traker para el seguimiento del proyecto

Iniciando el ciclo se decidió empezar por la experimentación. El número de bibliotecas exploradas fue bastante extenso debido a la gran variedad que ofrecía la red, en especial desarrolladas en el entorno de GitHub, y cuyo software era libre. La mayoría de estas bibliotecas

ofrecían unos usos muy específicos y se trataba de un entorno poco flexible en el que el trabajo podría convertirse en algo tedioso debido a la complejidad de las mismas, la falta de documentación y la poca forma en cuanto al trabajo realizado con estas. De las mismas se hizo una basta selección para probar con ellas con el objeto de conseguir discernir cuales de estas podrían adaptarse mejor a lo que buscábamos. Como se puede ver en la siguiente figura, se hicieron varias pruebas, en este caso con charts.js, mirando sus posibilidades de representación pero sobre todo la versatilidad del código que ofrecía para adaptarlo a nuestras necesidades.

Cabe destacar que lo importante no era en sí la representación de los datos, si no dar la oportunidad de una vez creada la plataforma y estructura de los objetos en nuestra jerarquía personal, que cualquier usuario interesado pudiera crear sus propios widgets enriqueciendo así el trabajo realizado.

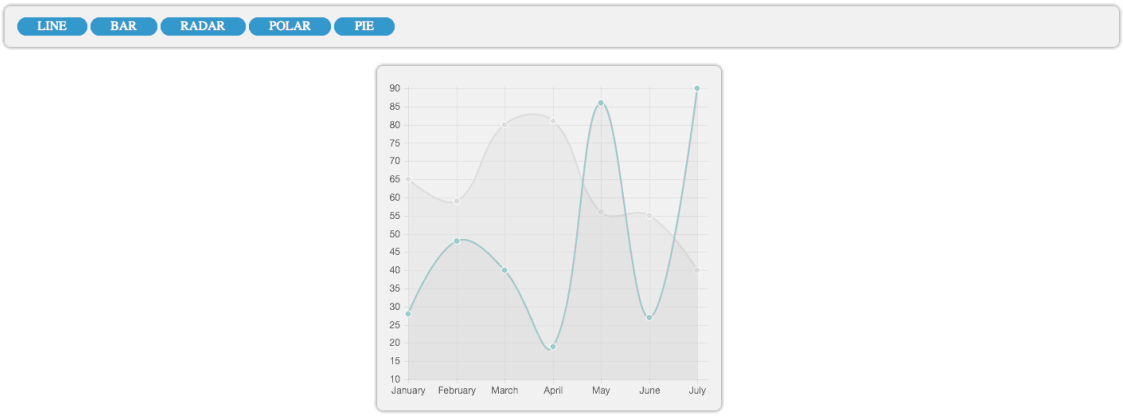

Figura 4.2: Ejemplo de widget de la biblioteca Chart.js

Finalmente y debido a su gran flexibilidad a la hora de representar datos de distinta naturaleza, la posibilidad de acceder a los mismos datos con relativa sencillez y a las facilidades que ofrecía documentación se decidió hacer uso de Highcharts.js . Como añadido la biblioteca no sólo permitía la creación de gráficas específicas si no que también permitía la posibilidad de la creación de gráficas que mezclaban propiedades de otras muchas como se puede comprobar en la siguiente imagen:

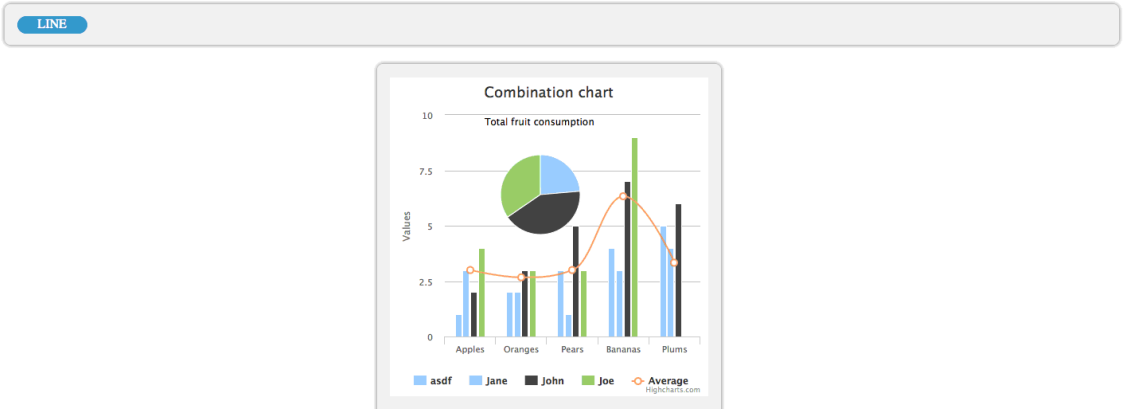

Figura 4.3: Ejemplo de widget de la biblioteca Highcharts.js

El uso de una biblioteca que permitiera la creación de una gráfica totalmente personalizada, cuya creación en el código sólo dependiera de un array de datos la hacía perfecta. Además de este tipo de gráficas highcharts.js ofrece numerosos tipos de extras en las gráficas, en comparación a las de código algo más específico, poco flexible y cerrado, pero que también podrían suponer un aliciente en cuanto a la representación de los datos.

A diferencia de otras bibliotecas, esta en concreto daba la oportunidad de tener un acceso sencillo y limpio a los datos. La personalización de la representación de datos a priori y posteriori suponía un aliciente en cuanto al interés de la misma teniendo en cuenta que además podíamos saber con facilidad qué estábamos representando una vez la gráfica estuviera construida.

A continuación podemos ver un fragmento de código que permite crear este tipo de gráficas:

```
$(function () {
  $('#container').highcharts({
    title: {
      text: 'Combination chart'
    },
    xAxis: {
      categories: ['Apples', 'Oranges', 'Pears', 'Bananas', 'Plums']
    },
    labels: {
```

```
    items: [{
      html: 'Total fruit consumption',
        style: {
          left: '50px',
          top: '18px',
          color: (Highcharts.theme &&
          Highcharts.theme.textColor) || 'black'
        }
    }]
},
series: [{
    type: 'column',
    name: 'Jane',
    data: [3, 2, 1, 3, 4]
}, {
    type: 'column',
    name: 'John',
    data: [2, 3, 5, 7, 6]
}, {
    type: 'column',
    name: 'Joe',
    data: [4, 3, 3, 9, 0]
}, {
    type: 'spline',
    name: 'Average',
    data: [3, 2.67, 3, 6.33, 3.33],
    marker: {
        lineWidth: 2,
        lineColor: Highcharts.getOptions().colors[3],
        fillColor: 'white'
    }
```

```
    }, {
        type: 'pie',
        name: 'Total consumption',
        data: [{
            name: 'Jane',
            y: 13,
            color: Highcharts.getOptions().colors[0] // Jane's color
        }, {
            name: 'John',
            y: 23,
            color: Highcharts.getOptions().colors[1] // John's color
        }, {
            name: 'Joe',
            y: 19,
            color: Highcharts.getOptions().colors[2] // Joe's color
        }],
        center: [100, 80],
        size: 100,
        showInLegend: false,
        dataLabels: {
            enabled: false
        }
    }]
  });
});
```

Como se puede comprobar en el código cada elemento añadido en el chart permite ser incluido mediante el array "series" dentro del cual podemos añadir desde un "pie" hasta un formato de datos estándar formado por un simple array de números.

Highcharts.js por su parte también permite la correcta representación de grandes cantidades

de datos a diferencia de otras bibliotecas y DashBoards de código abierto y adaptado a nuestra página web, pero fijos en cuanto a funcionalidad. Así se puede ver en la siguiente figura, donde demuestra su flexibilidad a la hora de adaptarse a los datos impidiendo la superposición de los mismos si se trata de archivos con una gran cantidad de valores.

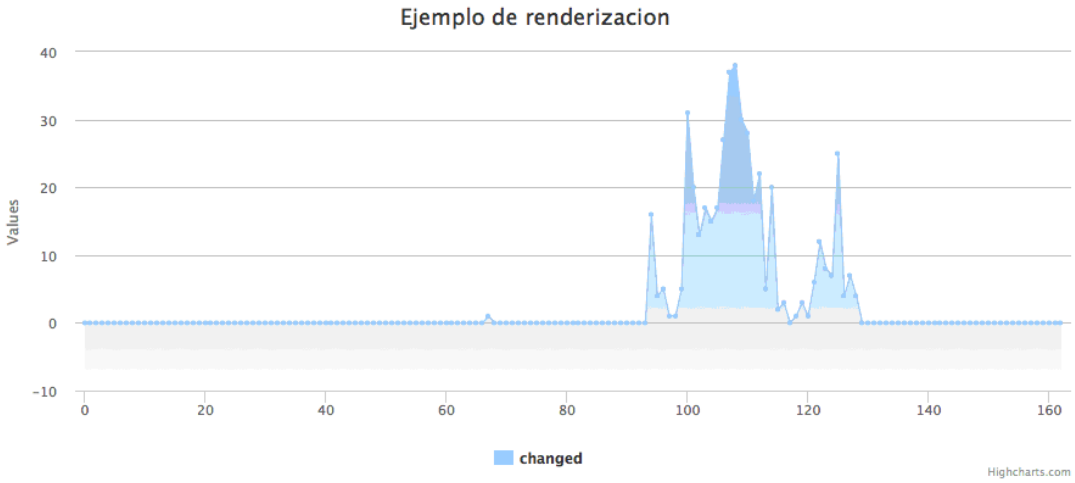

Figura 4.4: Ejemplo de widget de la biblioteca Highcharts.js con datos reales

Concluida la parte de búsqueda de biblioteca y seleccionada highcharts para nuestros propósitos el siguiente paso era poner en práctica lo aprendido en la red y buscar la forma de recrear una primera gráfica personalizada con el objetivo de empezar a crear nuestro DashBoard. Para ello es necesario trabajar con los ficheros JSON reales. En este punto era importante conocerlos a fondo debido a que su formato sería importante en cuanto a la selección de datos para la representación de los mismos.

Eso si, se debía tener en cuenta que a la hora de escribir el código las funciones de filtrado de datos debían estar debidamente divididas, siendo estas un ente independiente del resto de la funcionalidad del código. Esto debió hacerse de este modo debido que ya en este punto se barajaba con la posibilidad de la creación de nuestro propio formato de seguimiento de datos debido a que no existía un estándar equitativo entre todos los ficheros JSON existentes de distintas empresas.

Navegando por internet se descubrió que la mayoría de las empresas que hacen uso de esta clase de recursos para hacer estudios de mercado sobre sí mismas y saber los beneficios que obtienen durante un periodo de tiempo hacían caso omiso a los datos de otras empresas más allá de sus resultados finales dónde se podían comparar directamente y saber en qué posición

de la competición se encontraban. Esta situación daba lugar a lo ya mencionado de la existencia de ficheros JSON de distinto formato para la creación de gráficas que nos permitan hacer un seguimiento exhaustivo acerca de un proyecto.

Por otro lado, dada la complejidad de los datos con los que se estaba trabajando suponía que la representación de nuestra gráfica no dependerá de un único fichero, si no que dependerá de varios, los cuales pueden proceder de distintas fuentes, y que además el programa debe saber administrar para ofrecer al usuario una experiencia sencilla y amena.

En la siguiente figura se puede observar un primer prototipo de DashBoard el cual permite la representación personalizada de los datos de tres empresas distintas. Permite la selección personalizada de la representación de los propios datos. Es aquí dónde hablábamos de la flexibilidad de highcharts para permitir representar datos distintos de maneras distintas sobre el mismo widget. Por otro lado y pese a que en la figura no se muestre highcharts ofrece algunos extras para poder descargar la gráfica, imprimirla o exportarla en el formato que nosotros deseemos.

Por último, mencionar que permite la interactuación con los datos permitiendo seleccionar dentro de los datos ya existentes en la gráfica cuál queremos tener visible.

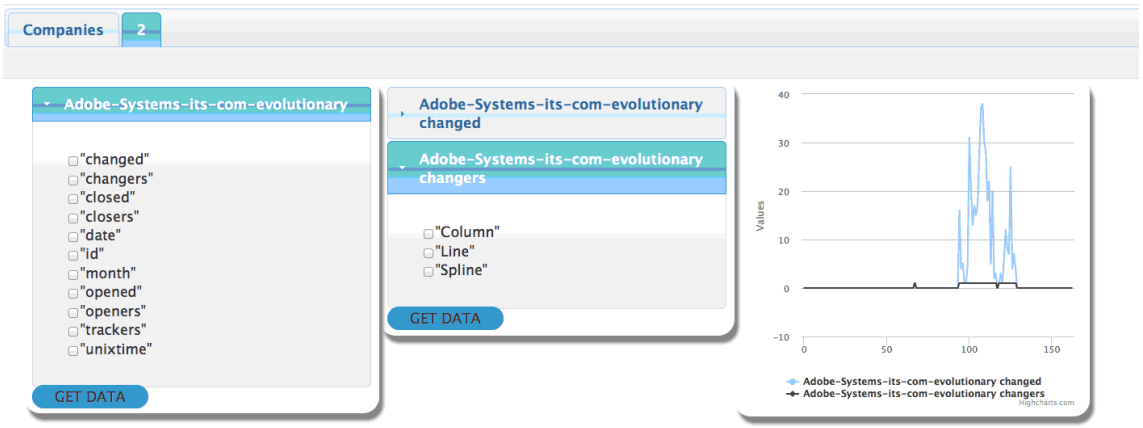

Figura 4.5: Primer prototipo de DashBoard

4.2.3. Bibliotecas y aplicaciones exploradas

- Higthcharts.js :

  Gran flexibilidad, ofrece la posibilidad de realizar dashborads personalizados simplemente cogiendo los elementos de un JSON o añadiéndolos en el formato que desee el usuario.

Es la biblioteca que nos ha ofrecido hasta la fecha mejores resultados en cuanto a adaptación de datos se refiere.

- Charts.js :

  No ofrece tanta flexibilidad como high charts, tiene un formato de inclusión de datos único que no se puede modificar, y es poco flexible. La adaptación de los datos es prácticamente distinta en cada gráfica, lo que implica más líneas de código a la hora de implementar un nuevo widget. Entre otras cosas Chart.js no incluye leyendas a diferencia de Hightcharts. Para poder incluirlas hay que añadir un nuevo widget aparte del existente para crear la gráfica.

- d3.js :

  Ofrece una gran variedad de gráficas con las que trabajar, pero a nivel de código es poco flexible como Charts.js. Además de eso está programado a muy bajo nivel, con lo que puede no ser cómodo para nuestros objetivos. d3.js es el origen de casi todas las bibliotecas de representación gráfica. En el caso de no poseer una biblioteca que se adapte a nuestras necesidades siempre podemos recurrir a d3.js para crar nuestras propias gráficas que se adapten a nuestros datos.

- Jslate :

  Produce errores a la hora de modificar a nuestro gusto el DashBoard, es poco flexible en este aspecto, pero ofrece el mejor de los formatos quesito a la hora de representar por su flexibilidad, facilidad e intuición. Permite el filtrado de datos mediante un pequeño cuadro de texto, lo cual permite distintos análisis del usuario.

- Dashing :

  Biblioteca muy original y muy llamativa. Permite la representación y seguimiento de datos en tiempo real aplicable al mundo de las finanzas, o a la programación en equipo a gran escala. En este caso la representación de los datos en cuanto a código refiere es bastante ambigua y poco intuitiva.

- Dashku :

Código abierto, pero no existen ejemplos ni referencias para tocar el código del autor. Habría que investigarla más.

- dc.js :

  Biblioteca de gran flexibilidad aparente. Posee una gran comunidad de usuarios que han aportado su granito de arena y han creado sub-bibliotecas que pueden ser aplicadas a distintos ámbitos. Pese a todo la biblioteca no ofrece un código abierto claro más allá del código subido en GitHub por el autor, lo que la hace al igual que con dashku una biblioteca difícil de abarcar en primera instancia. Esta biblioteca está especialmente diseñada para el uso de la bilioteca crossfilter, la cual viene incluida dentro de esta. Es especialmente buena para el diseño de widgets gráficos que permitan la modificación dinámica de varias gráficas simultáneamente al ser una modificada. Es un punto a favor y bastante interesante para el estudio de los datos desde varios puntos de vista y de forma sencilla.

- Crossfilter :

  Se trata de un algoritmo de agrupamiento de datos, el cual es relativamente útil si hacemos uso de dc.js. Esta biblioteca nos facilita el masajeo de los datos con ficheros JSON relativamente grandes procedentes de empresas, realizando las operaciones en unos pocos segundos. Incluye las funciones ”map” y ”reviews”.

- nvd3.js :

  Viendo el código y los ejemplos que vienen, y haciendo unas pequeñas pruebas con ella sobre el mismo código que ofrece en su página he podido ver que dicha biblioteca se asemeja mucho en uso a Charts.js. Lo que ofrece básicamente es una biblioteca basada en d3, que saca de los ficheros JSON de manera directa su formato a la hora de aplicarla en el JavaScript, y que modifica su formato dependiendo de la gráfica que se quiera usar. Sin embargo es mucho más flexible que Chart.js, permitiendo añadir o quitar datos representados sobre el widget gráfico. Otro punto a su favor es que la representación de las gráficas temporales es bastante destacable con respecto a otras bibliotecas del mismo ámbito.

- c3.js :

De origen japonés se asemeja mucho a nvd3 en cuanto a la flexibilidad a la hora de jugar con los datos que están representados dentro del mismo widget, pudiendo quitar un dato o añadirlo a una gráfica ya pintada. Como añadido podemos decir que es aún más flexible porque no sólo nos permite jugar con los datos de manera dinámica si no que además nos permite cambiar su forma de representación de una forma mucho más simple que otras bibliotecas saltándonos el paso de tener que repintar el widget. Es decir, con el mismo widget dibujado podemos pedir al objeto situado ya en el árbol DOM que se transforme a otro tipo de widget distinto, véase de un tipo ”pie” directamente a un tipo ”bar”.

- Stashboard :

  Biblioteca que hace uso de python para subir, bajar y deletear datos. Su documentación es prácticamente nula salvo dos ejemplos sueltos situados en su página principal. Para ver en más en profundidad la biblioteca hay que pegarse directamente con el código del autor. En este caso la biblioteca a diferencia de otras es muy pequeña.

- Grafana :

  Código abierto para descargar, pero carece de documentación que explique sus funciones. Lo único que posee es una pequeña aplicación de ejemplo dónde se pueden modificar unas gráficas ya existentes. Si accedemos al contenido en GitHub, veremos que la propia aplicación de ejemplo es la que nos descargaremos, en otras palabras, se trata de un DashBoard ya terminado. Puede ser útil a la hora de coger ideas para le nuestro, pero debido a que ya tiene un modo de representación creado hace que lo situemos en los descartes.

- Freeboard :

  Código complejo solo disponible desde el repositorio fuente donde fue escrita. No tiene documentación suficiente, ejemplos o tutoriales. Lo que hemos podido ver sobre este mismo es que se trata de un DashBoard ya acabado, con varias opciones dónde crear nuestras propias gráficas a partir de ficheros JSON o similares. Como contrapartida debemos añadir que tiene una serie de bibliotecas gráficas de representación de datos que están orientadas a la representación de datos en tiempo real. Debido que tiene ya una serie de biblotecas gráficas establecidas añadir una nueva se convierte en un trabajo algo tedioso aun que realizable.

- Graylog2 :

  Se trata de una biblioteca no abierta destinada al ámbito empresarial. No hay forma de estudiarla en primera instancia ni tampoco forma de descargarla, pues se trata de una biblioteca que no es de código libre, si no que requiere de pago para acceder a sus prestaciones, pero no tendremos acceso de manera directa a su código.

- Logstash :

  Biblioteca de alto nivel de código abierto pero nula documentación. La única forma de aprender sobre esta es meter mano directamente al código del autor el cual es altamente amplio.

- Kibana :

  Utiliza una implementación semejante a Logstash, pero de la misma forma que la anterior carece de ejemplos, tutoriales o documentación accesible para aprender sobre el código y cómo usarlo. Lo que ofrece Kibana es simplemente un DashBoard ya implementado que nos permite modificar a nuestro gusto con las gráficas que nos interesen, guardarlo y compartirlo con otra gente. Básicamente lo que estamos buscando nosotros para crearlo pero de una manera más cerrada.

4.3. Iteración 1. Creación de un *DashBoard* genérico

4.3.1. Introducción

En este primer ciclo de iteración nos basaremos en el desarrollo de un DashBoard genérico que nos permita incluir un soporte con el que poder empezar a programar nuestro proyecto de manera fija. Sobre él estableceremos algunas pautas de la interfaz de usuario que puede que sean modificadas en un futuro para la mejora del diseño del mismo.

4.3.2. Desarrollo del ciclo

Llegada la finalización de la versión anterior, el siguiente paso dada la complejidad que se avecinaba en cuanto a la administración de los datos y la interfaz a proponer al usuario, se pensó en la creación de un segundo prototipo orientado a establecerse tanto el principio como el final en cuanto a apariencia.

La programación sobre una plantilla web establecida facilitaría el desarrollo de la aplicación de una forma más concreta, y nos permitiría poner los pies en la tierra en cuanto a los objetivos que buscábamos en un principio, pudiendo discernir ya de primera mano cuales de estos son factibles, o cuales de estos podrían estar limitados por la tecnología o la prestaciones pensadas en un inicio para el usuario.

En otras palabras, tener una primera apariencia de lo que podría ser un futuro DashBoard nos haría más conscientes de lo que teníamos hecho y de lo que no.

De este modo se decidió comenzar con los siguientes pasos:

- Creación de una interfaz, con objeto principal y destacado una hoja en blanco sobre la que trabajar.
- Traspaso de distintos DashBoards del prototipo anterior.
- Creación de las tres principales gráficas que representará el programa.
- Permitir la personalización de dichas gráficas como en el prototipo anterior
- Buscar los primeros eventos relacionales entre las gráficas que permitan escarbar entre los datos.

Bootstrap nos da la posibilidad de crear una plantilla simple en la que trabajar, gracias a la cual podemos crear fácilmente nuestro propio DashBoard en poco tiempo, para seguidamente añadir otras posibilidades más prácticas.

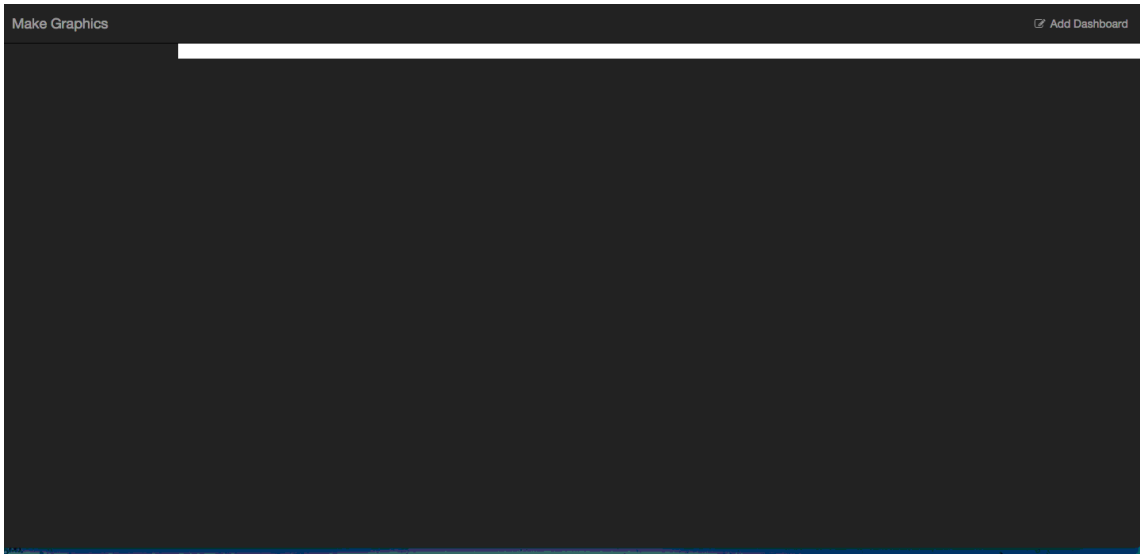

Figura 4.6: DashBoard vacío

En el mismo escenario hemos establecido la posibilidad de crear distintos DashBoards para poder trabajar de manera paralela con gráficas personalizadas que pueden estar orientadas a distintos ámbitos ofreciendo de este modo al usuario un método sobre el que organizarse a él mismo.

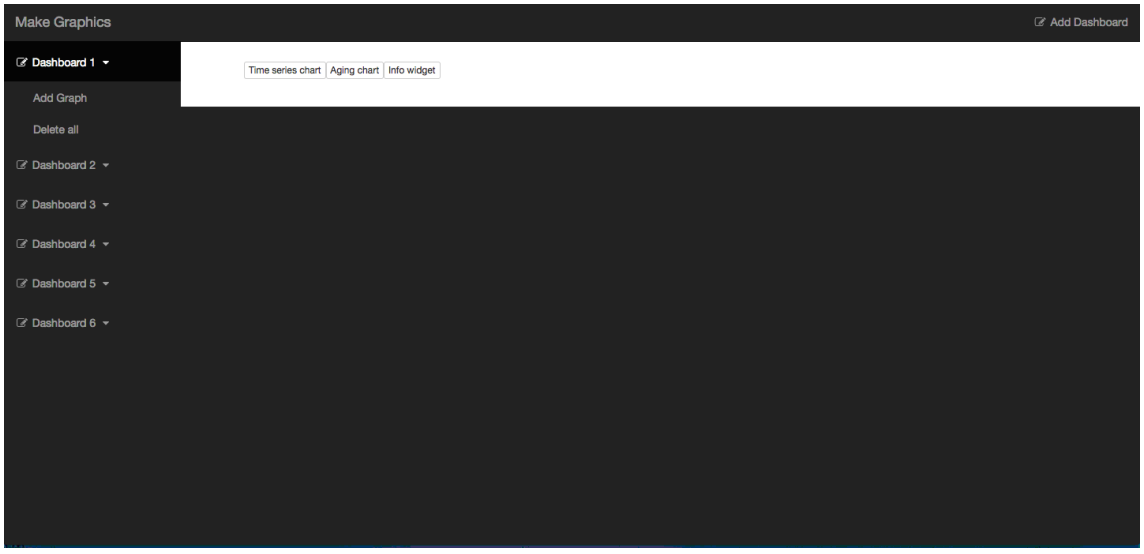

Figura 4.7: Primeros paneles creados

Como se observa en la figura anterior, se ha dado la posibilidad de crear distintos tipos de gráficas las cuales dependen de los JSON apropiados en cada una de ellas. Es decir, una gráfica toma datos de dónde los necesita de manera inteligente, y no tomándolos de un único JSON. Como podremos comprobar en las sucesivas imágenes se muestran las métricas de cada tipo de gráfica con el objetivo de dejar decidir al usuario cuál de ellas se desea representar y de qué forma. Cabe decir que hay métricas que no tienen demasiado sentido representarlas junto a qué otras pero es decisión del usuario saber qué es lo que desea ver en qué zona.

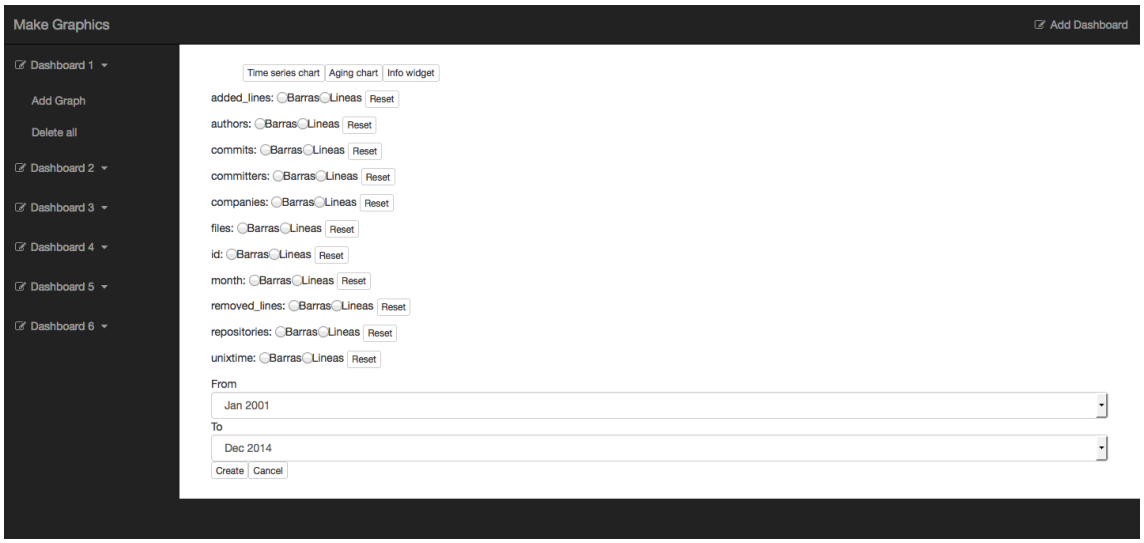

Figura 4.8: Primer prototipo de creación de una gráfica de tipo tiempo

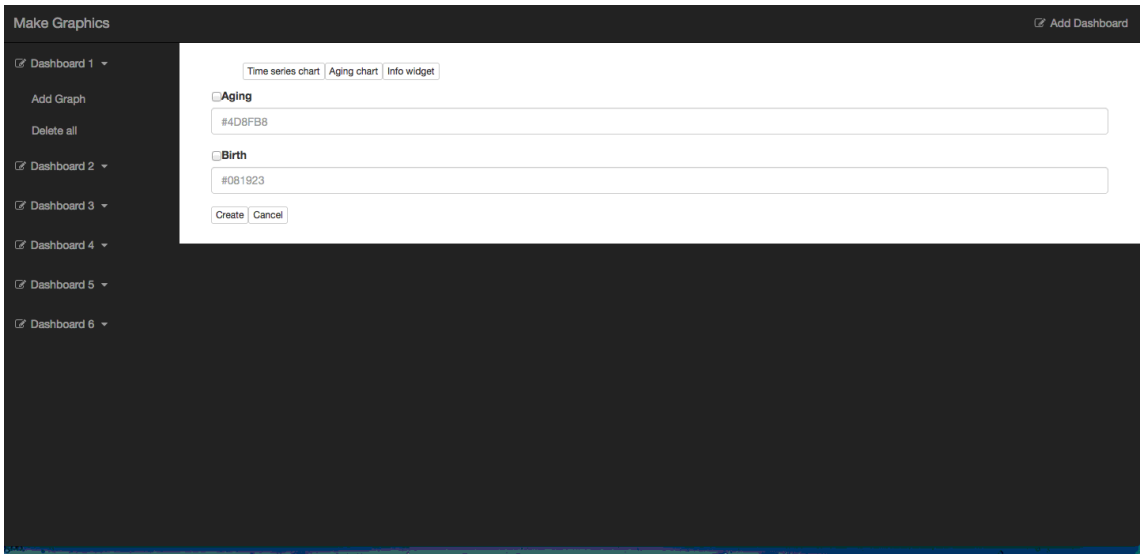

Figura 4.9: Primer prototipo de creación de una gráfica de tipo demografía

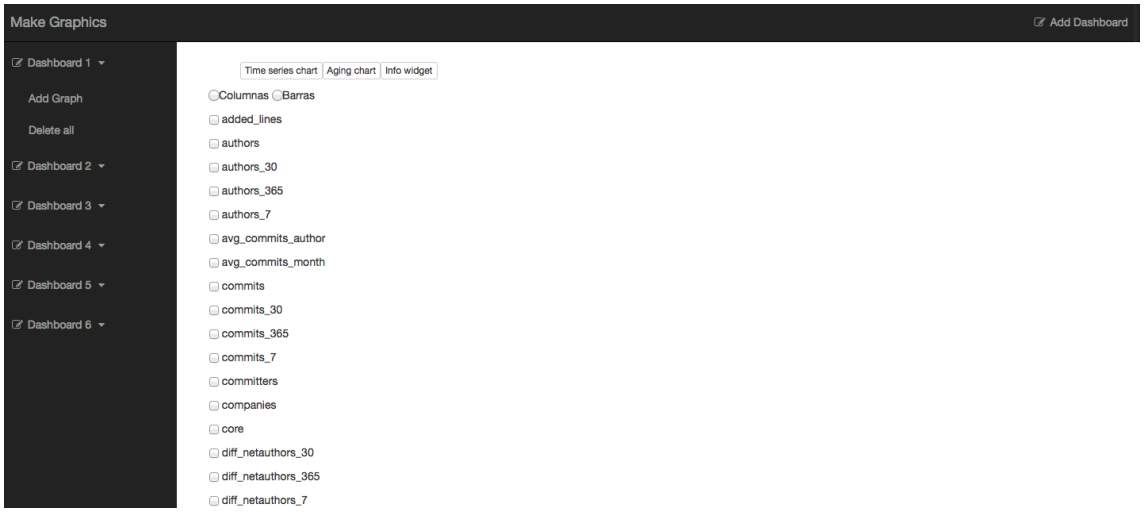

Figura 4.10: Primer prototipo de creación de una gráfica de tipo información

Cada DashBoard es personalizable en cuanto a organización y gráficas a utilizar dentro del mismo, permitiendo la movilidad de las gráficas entre ellas.

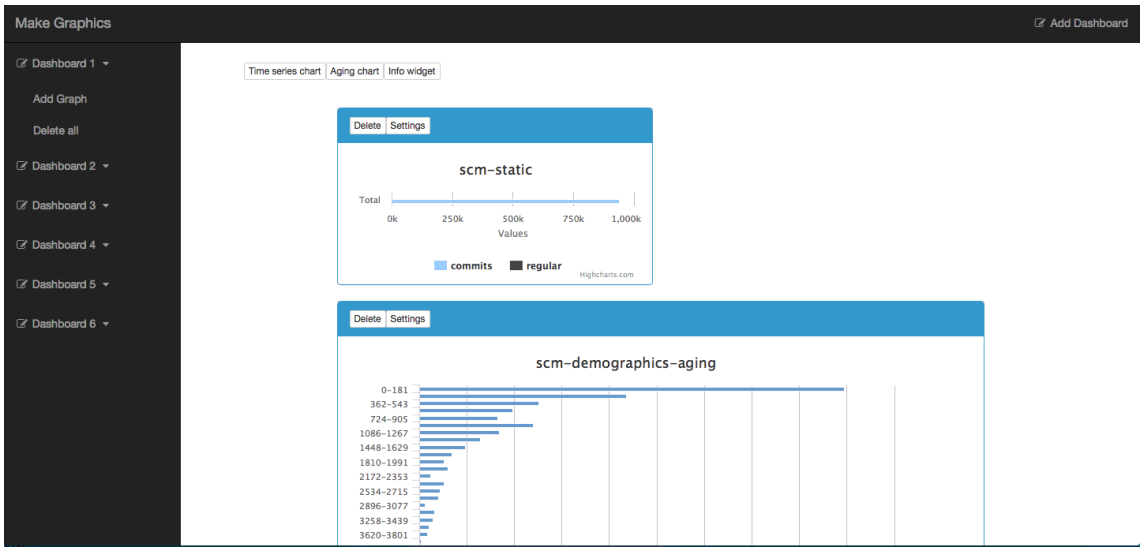

Figura 4.11: Primeros widgets creados

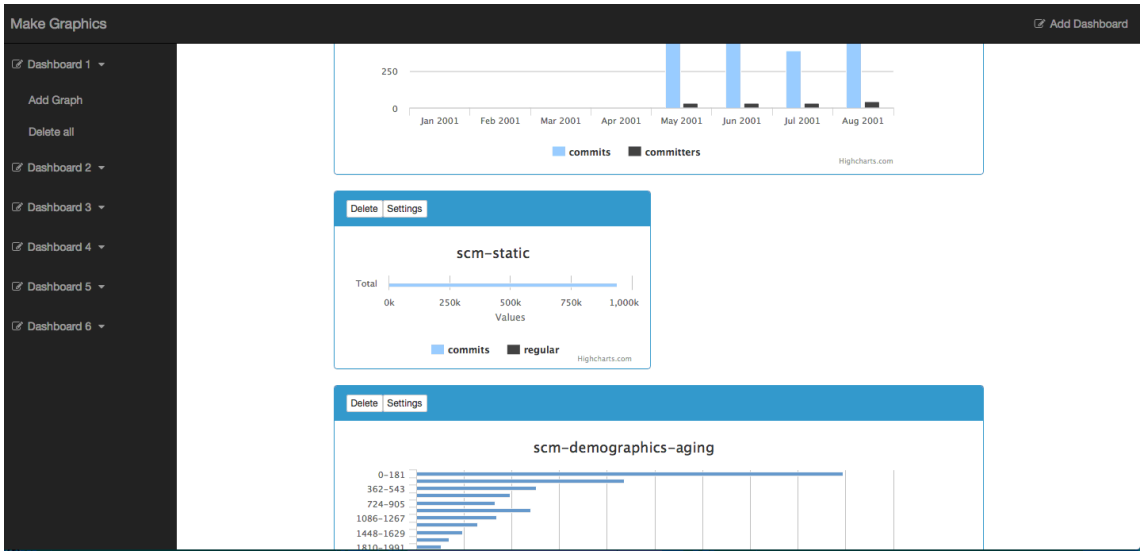

Figura 4.12: Primeros widgets creados

Cada gráfica del mismo modo ofrece la posibilidad de personalización a posteriori de la misma. Es decir, podemos de la misma gráfica seleccionar los datos más interesantes y eliminar o descartar los que hemos comprobado que no nos son de utilidad. Highcharts.js como ya dijimos nos ofrece esta posibilidad de manera sencilla[1].

De esta manera podemos observar cómo el dato deseado posee un acceso relativamente sencillo, teniendo posibilidad de conocer todas y cada una de sus características o cambiarlas para redibujar la gráfica si fuera necesario.

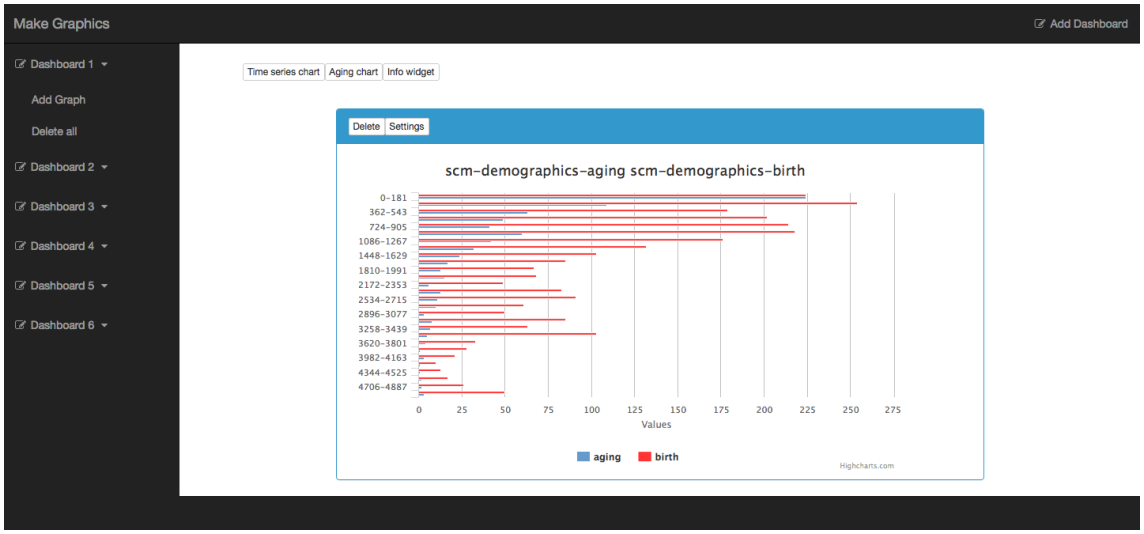

Figura 4.13: Ejemplo de widget demográfico modificado

[1]Véase el anexo B dónde se muestra un ejemplo de la estructura de un objeto de Highcharts

Del mismo modo en este prototipo que aquí se presenta se ofrece los primeros eventos entre datos que nos permiten escarbar en los datos y hacer comparaciones más profundas en cuanto a lo que buscamos. Por ejemplo, en la figura anterior mostramos un gráfico de edades de un proyecto de desarrollo de software. Tanto edad de nuestros trabajadores (age de color azul) dentro del proyecto como iniciados a lo largo del tiempo en esa época (birth de color rojo). Las unidades de la tabla en el eje y indican la cantidad de días que han estado dentro de la empresa.

En este caso la evolución durante el 2014 como se puede observar ha sido bastante notorio, pero por ejemplo nos puede interesar conocer la gente que ha entrado en el proyecto comparándola con la gente que esta en él, es decir, buscamos sus nombres. La gráfica ofrece eventos de clicado que permiten la creación de nuevas gráficas para escarbar con mayor profundidad en los datos.

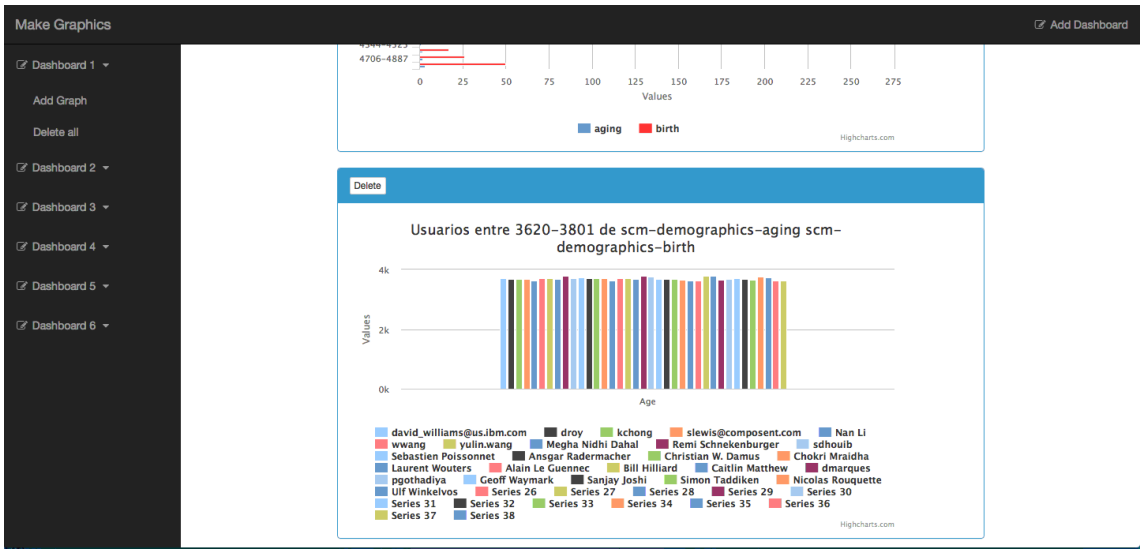

Figura 4.14: Gráfica resultante de clicar sobre una gráfica de demografía

La comparativa de los datos puede ser un elemento interesante sobre el que trabajar en los próximos ciclos. Por otro lado, el extrapolar dicha aplicación a la conexión online la puede hacer aún más interesante, pudiendo hacer comparativas o seguimientos de las gráficas a lo largo del tiempo, convirtiéndose en una herramienta notoriamente útil para la organización de un equipo.

En los próximos ciclos se explorarán las posibilidades de compartir DashBoards, permisos de modificación mediante urls únicas, sincronización durante la creación de un nuevo tablero personalizado, y se pondrán medidas a los posibles errores que puedan existir en cuanto a la obtención de ficheros y los errores que estos puedan provocar. Del mismo modo se buscará una

mejora de la interfaz de usuario.

4.4. Iteración 2. Creación de un sistema de guardado

4.4.1. Introducción

Conociendo ya una primera visualización de nuestra aplicación, y viendo que la creación de las gráficas funcionaba correctamente se procedió a la ampliación del número de funcionalidades que aportaba el entorno.

De esta manera se mejoró ligeramente la visualización del gridster y los widgets, para intentar pasar al siguiente ciclo, la implementación de un servidor, guardado y aplanamiento de las gráficas y la reproducción de las mismas si se quiere acceder de nuevo a ese entorno de trabajo ya existente. Este pequeño paso intermedio era importante haciendo de él un uso semejante al de una traza, con el que discernir durante su programación si el funcionamiento era correcto o no.

Para lograr esto se procedió al uso de Django y Python. Un servidor a través del cual se añadirían las urls y del cual se haría uso de sus bases de datos en primera instancia para poder hacer las pruebas pertinentes. Véase que si en un futuro se quisiera subir la aplicación a un entorno web permanente existen herramientas que nos pueden hacer la vida más fácil pero la mayoría de ellas no hacen uso de las bases de datos sqlite de las que hace uso Django por defecto.

4.4.2. Desarrollo del ciclo

Así es cómo buscamos en este ciclo:

- Ligera mejora gráfica de la apariencia de los widgets
- Guardado mediante urls únicas
- Sincronización con la creación de las gráficas
- Aplanamiento simple de las mismas

Y es de este modo que se presenta la siguiente organización de urls que nos permitirán interactuar con nuestra aplicación:

A vista del usuario:

- ../ : DashBoard vacío, página home desde el que empezar a utilizar nuestra aplicación
- ../d/+: DashBoard ya creado que se carga para seguir modificándolo a nuestro gusto

Desde el punto de vista del servidor:

- ../:

  Get: DashBoard vacío, página home desde el que empezar a utilizar nuestra

- /db/:

  Get: Listado de ids de los DashBoards creados

  Post: Creación de un nuevo DashBoard.

- /db/d+:

  Get: Carga de un DashBoard ya creado, devuelve el JSON de configuración del mismo.

  Put: Sobreescritura de un DashBoard ya creado con una nueva configuración.

- /db/default:

  Get: Presentación del DashBoard por defecto con el que mostrar al usuario la funcionalidad.

Con esta nueva estructura tendríamos algo semejante a esto:

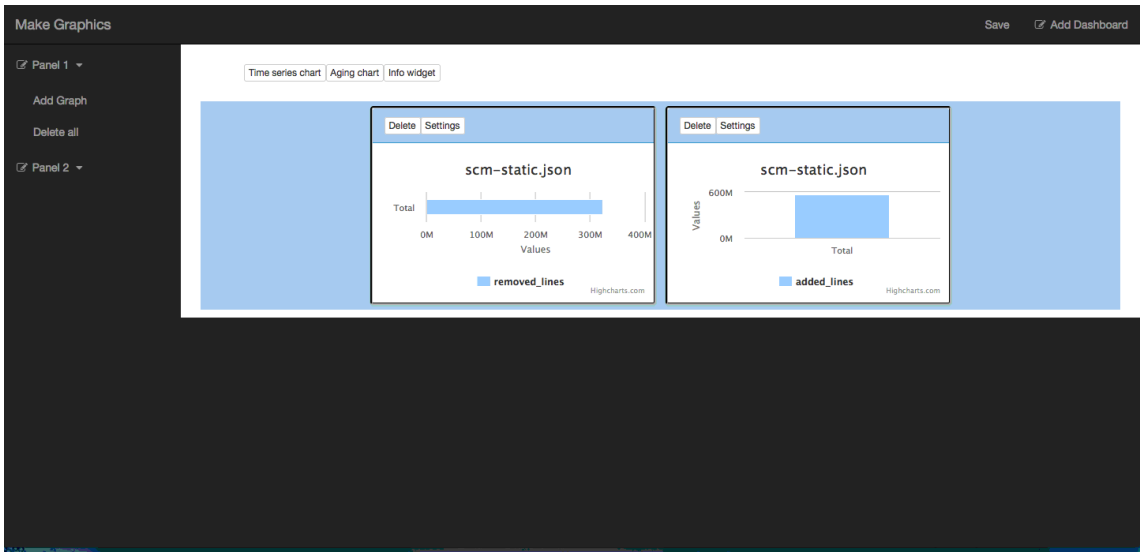

Figura 4.15: Widgets recien creados

Como observamos en la figura anterior hemos creado dos gráficas básicas sobre un DashBoard vacío aún no creado. Las aplicación permite el guardado de las mismas mediante el botón ”Save” situado arriba a la derecha. Al pulsarlo...

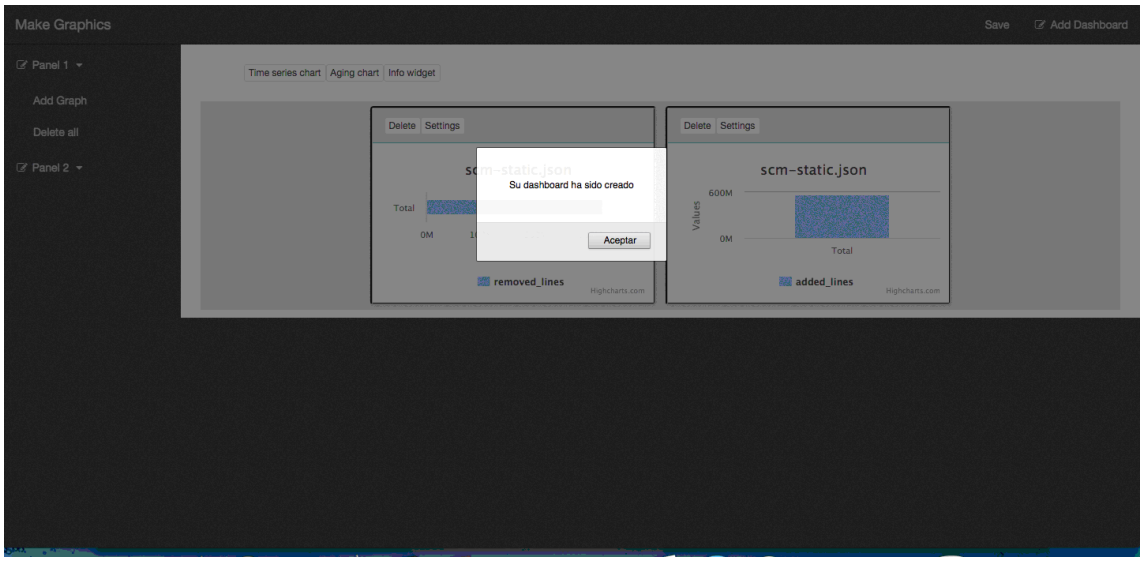

Figura 4.16: Guardado completado

Se creará un nuevo DashBoard en nuestro servidor, un mensaje aparecerá en pantalla para avisarnos de tal evento.

Como podemos ver en la segunda imagen la url en la que estamos posicionados ha cambiado conforme hemos guardado, situándonos ya en el lugar correspondiente de nuestro entorno de trabajo.

Si entramos en el funcionamiento del fichero de configuración tendríamos actualmente algo de este formato.

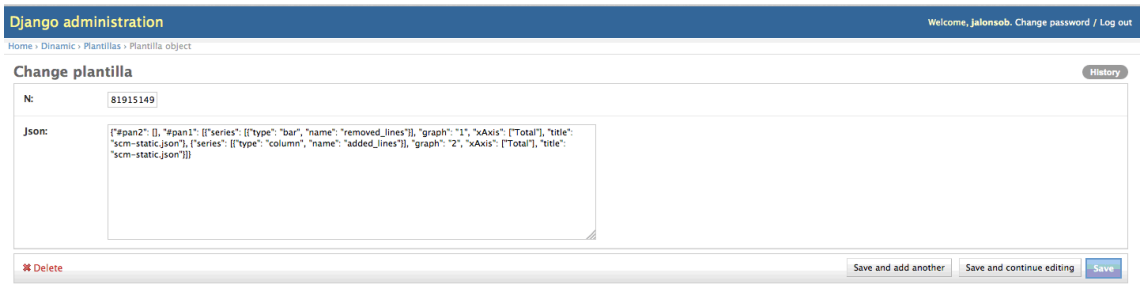

Figura 4.17: Visión del administrador del servidor

Cada DashBoard está identificado por un número único N y por un JSON de configuración. Dicho JSON es un objeto cuyas claves son los paneles que dibujamos, y cuyo contenido son arrays. Dentro de cada posición de los arrays tenemos objetos con las características necesarias para guardar debidamente una gráfica. El título, las métricas y su formato así como el identificador de gráfica son elementos importantes a la hora de guardarlo dentro del servidor para posteriormente reproducirlo debidamente.

Uno de los mayores problemas que existen al hacer uso de una aplicación que necesita de peticiones de datos de gran envergadura es el tiempo de carga con respecto al tiempo de espera de los datos.

Por eso fue necesaria una reestructuración completa del código orientada a la espera de los datos para después hacer uso de los mismos. Es decir, pido los datos si son necesarios, si están pedidos espero a recibirlos, si no los recibo lanzo un evento orientado a un error en la descarga, y si ha resultado correcta la petición pinto la gráfica sobre la que estaba trabajando y las subsiguientes que tal vez dependían del mismo tipo de JSON.

Establecido nuestro servidor, nuestro guardado y reproducción de charts, pasamos al siguiente ciclo donde puliremos algunos defectos que actualmente posee nuestra aplicación.

4.5. Iteración 3. Implementación mediante descomposición por objetos

4.5.1. Introducción

Con el avance de la aplicación nos vimos en la situación de que una vez concluida esta podría ser de utilidad a alguien desde el punto de vista externo para darle unos usos propios y añadir una serie de widgets personalizados. De esta manera en este ciclo se ha buscado una conversión total del código de centralizado a dividido en secciones.

4.5.2. Desarrollo del ciclo

Visto desde un punto de vista iterativo, los objetivos a cumplir en esta parte fueron:

- División del código y las gráficas en objetos
- Reestructuración de los paneles en formato objeto
- Recreación de los menús por parte de cada widget de manera individual y propia, totalmente ajena al código general del DashBoard
- Aplanamiento de manera individual de cada widget, olvidándonos de cómo se estructura y cómo se recrea.
- Comprobación de dicha estructura creando dos nuevos widgets

La estructura a establecer de este modo siguió la siguiente jerarquía:

```
Panel  {
var id;
var widgets;
var color;
var settings;

this.getSettings;
```

```
this.getId;
this.getWidgets;
this.flatten;
this.changeColor;
this.pushElement;
this.deleteElement;
this.deleteAll;

}
```

Lista de variables:

- id: Identificador propio del panel
- widgets: Array de los widgets que posee. Se almacenan como objetos completos no aplanados
- color: Color de fondo del widget por si se deseara modificar.
- settings: Barra de botones de creación del widget.

Lista de funciones:

- this.getSettings: Devuelve la barra de botones
- this.getId: Devuelve la id
- this.getWidgets: Devuelve la lista de widgets
- this.flatten: Función que devuelve el panel aplanado con todos sus elementos
- this.changeColor: Cambia el color
- this.pushElement: Incluye un nuevo elemento al panel
- this.deleteElement: Elimina un elemento del panel
- this.deleteAll: Elimina todos los elementos del panel

```
Widget {
this.id;
this.color
this.panel;
this.content;
this.buttons;
this.square;


this.gridsterWidth;
this.gridsterheight;
this.toDraw;
this.changeContent;
this.changeColor;
this.drawErrorWidget;
}
```

Lista de variables:

- this.id: Número de identificación del widget
- this.color: Color del widget
- this.panel: Id del panel al que pertenece
- this.content: Contenido del panel
- this.buttons: Botones del panel
- this.square: div de botones y contenido unido para implementarlo directamente en el gridster
- this.gridsterWidth: Ancho del widget
- this.gridsterheight: Alto del widget

Lista de funciones:

- this.toDraw: Función para dibujar el widget en el gridster
- this.changeContent: Cambia el contenido
- this.changeColor: Cambia el color
- this.drawErrorWidget: Dibuja un widget de error

A partir de esta primera estructura genérica cualquiera puede aplicar las bibliotecas que necesite para la inclusión de nuevos widgets que se pueden aplicar a diversas tareas. En términos generales, lo más apropiado es que al aplicar una nueva biblioteca creemos un nuevo widget padre del que sacar los hijos para cada una de sus aplicaciones. Esto es debido a que por regla general en el entorno de las bibliotecas gráficas de representación las funciones para distintas gráficas suelen ser las mimas, y aun que su representación sea distinta, su forma de modelarlas es semejante.

Con respecto a esto veamos un pequeño ejemplo con lo que tenemos entre manos. Tomando nuestra biblioteca predilecta HighCharts.

En primer lugar debemos crear la clase padre con la que crear las subclases para la realización de los widgets de distintos tipos. Esto es lo que sigue.

```
HighWidget {

this.getSerie;
this.existLabel;

}
```

Como podemos comprobar apenas tenemos dos funciones, pero no necesitamos más. Estas funciones son las que unen a todos los widgets que hacen uso de esta biblioteca y que dando igual la naturaleza de lo que dibujemos las necesitaremos para formar los widgets a posteriori.

- this.getSerie: Obtiene la característica ”series” de un widget ya pintado
- this.existLabel: Comprueba si una métrica existe dentro de un widget gráfico ya pintado.

A partir de este punto podemos crear los objetos hijos que se corresponden a dicha biblioteca. Dado que actualmente tenemos 3 tipos de gráficas cuyo código es bastante extenso, nos centraremos nuevamente en la más simple, la de tipo información estática.

```
HighInfo {
this.buttons;
this.square;
this.title;
var series;
this.flatten;

this.makeMenu;
this.takeData;
this.MakeWidget;
this.Parser;
this.Draw;
this.getSeries;
this.settings;
this.redraw;

}
```

Lista de variables:

- this.buttons: Variable con los botones nuevos adaptado al widget en concreto
- this.square: Variable con el recuadro nuevo adaptado al widget en concreto
- this.title: Titulo del chart

- series: Listado de series y su formato para la creación del widget gráfico

Lista de funciones:

- this.flatten: Función de aplanamiento para guardar el widget en un formato JSON
- this.makeMenu: Función para recrear el menú propio de este tipo de widget gráfico
- this.takeData: Función para obtener los datos de menú creado por el mismo widget
- this.MakeWidget: Función que permite al widget dibujarse a sí mismo
- this.Parser: Función de parseo necesario para los datos propia de este widget
- this.Draw: Función para dibujar sobre un canvas una gráfica con la biblioteca que estamos usando
- this.getSeries: Devuelve las series guardadas del gráfico
- this.settings: Reproduce un menú con las opciones que el widget está representando marcadas
- this.redraw: Redibuja el widget con las nuevas opciones

Este nuevo modelo simplifica notoriamente el funcionamiento de nuestro DashBoard a la hora de introducir nuevos tipos de widgets, así como el aplanamiento de las gráficas para posteriormente ser capaces de reproducirlas nuevamente.

Por otro lado el tratar cada objeto de manera independiente nos facilita el manejo de las distintas gráficas, no solo para aplanarlas y guardarlas, si no también para ser capaces de cambiar las gráficas de lugar. Dicho esto veamos de manera más gráfica los resultados de esta iteración.

En el primer ejemplo veremos el movimiento de las gráficas del que hablamos. Como podemos ver en la imagen que hay a continuación tenemos presentadas dos gráficas, una de evolución temporal y otra de información estática.

Figura 4.18: Ejemplo de los resultados con esta nueva arquitectura

Creados 4 paneles distintos decidimos desplazar nuestro gráfico estático al panel 3.

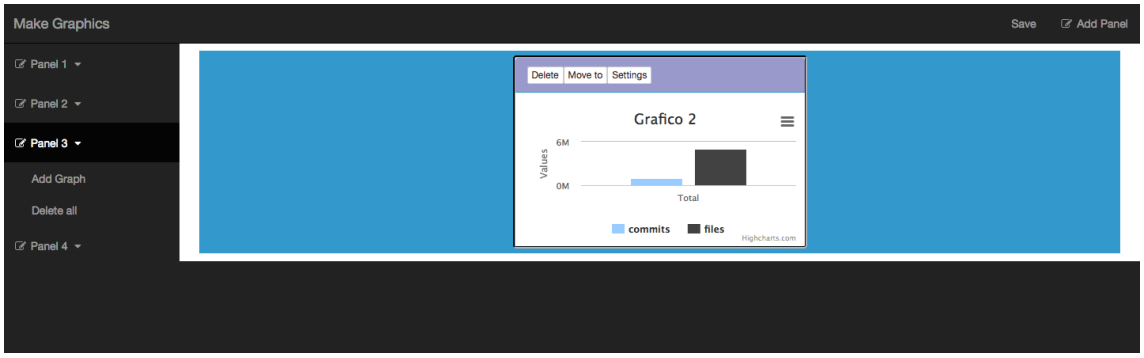

Figura 4.19: Chart de panel 1 en panel 3

Este se desplaza automáticamente una vez seleccionado el lugar donde queremos moverlo. Mantiene la apariencia, desde el color del widget, pasando por el tamaño y el formato de los datos.

Viendo que la jerarquía funciona de cualquier modo decidimos crear en este panel 3 un widget de video, que como hemos explicado anteriormente es hijo del widget principal, dado que en este caso no tenemos uso de nuestra biblioteca gráfica.

Figura 4.20: Ejemplo de widget con video

Por último decidimos ahora realizar un guardado del DashBoard sobre el que estamos trabajando. Al guardar, se almacena del mismo modo que se hacía en el ciclo anterior, con la diferencia de que ahora cada widget se aplana por sí solo.

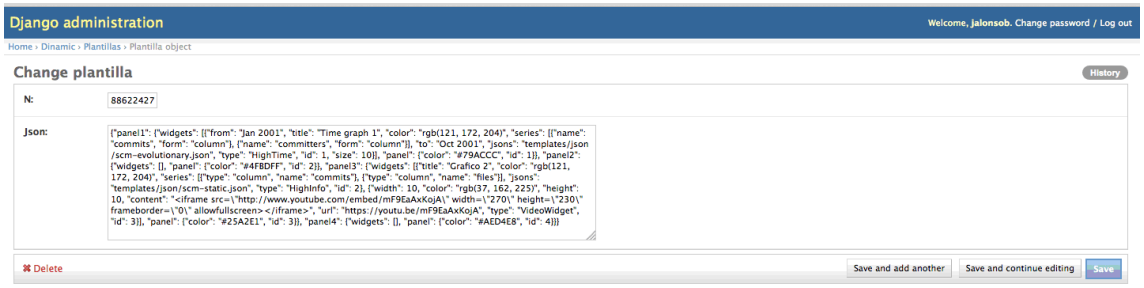

Figura 4.21: Ejemplo de nuevo formato de JSON de configuración

4.6. Iteración 4. Ficheros de configuración complejos, independencia en representación y múltiples empresas

4.6.1. Introducción

En esta sección vamos a hacer un análisis de la evolución de los ficheros de configuración a lo largo del desarrollo del proyecto. Estos ficheros constituyen una parte notoriamente importante debido a que son los responsables del prediseño de cada widget en cuanto a tipo gráfico a la hora de ser representado. Por otro lado los eventos de representación de los widgets gráficos de Highcharts.js que dependen de una serie de datos que deben ser descargados han tenido que ser modificados en vista a este nuevo planteamiento.

4.6.2. Desarrollo del ciclo

Como ya hemos visto con anterioridad, es necesaria de la lectura de una serie de ficheros de datos situados en una localización que viene definida en nuestro archivo de configuración. Esta lectura de datos debe ser lo más eficiente posible y establecerse mediante un orden de llamadas orientada a la captura y lanzamiento de eventos de descarga.

En secciones anteriores vimos un ejemplo de esto, donde se establecía una serie de eventos que estaban orientados a la creación de las gráficas de manera simultánea una vez los archivos estuvieran descargados (en este caso por simultáneo hablamos de gráficas que dependen del mismo JSON). Para poder dibujar las gráficas y saber las métricas que queremos representar en el widget debemos ofrecer al usuario la posibilidad de representarlas a su gusto. La mejor manera de hacer esto es realizar un pequeño estudio de las métricas de las empresas y ver cuales de ellas tienen en común. Para verlo desde un punto de vista más directo podemos expresarlo como:

- Reconfiguración de las métricas en la configuración del DashBoard.
- Descarga asíncrona de los ficheros de datos.
- Rediseño del modo de dibujar los widgets.

En las primeras pruebas se descargaban los archivos de manera directa cuando se quería representar algunas de las gráficas para posteriormente mostrar las métricas, y una vez estos

están guardados en caché hacer uso de los datos que poseen, sin embargo esta idea cae por su propio peso cuando nos damos cuenta de que existen proyectos en los que participan más de 50 empresas, dónde cada empresa posee entre 3 y 5 archivos de distinta naturaleza de datos, y que dentro de estos mismos existen a su vez 5 tipos. El número de descargas crece considerablemente haciendo nuestra aplicación lenta, tediosa y para nada útil.

A partir de la creación de un pequeño script en JavaScript pudimos ver cuales eran las métricas que tenían en común las empresas (lógicamente no lo íbamos a hacer a mano) y a partir de ellas crear un objeto que podría constituir nuestro archivo JSON de configuración. Si bien es cierto, el fallo en la descarga de un único fichero tiene más posibilidades que el fallo en la descarga de 750 ficheros. Es poco probable que fallen los 750 de manera simultánea en comparación a uno o dos, sin embargo este modelo corresponde al más eficiente debido a que la memoria pese a ser un recurso actualmente abundante y relativamente barato es algo que debemos saber gestionar con cuidado, sobre todo si se trata de movimientos en red por la lentitud que pueda producirse en nuestra aplicación.

Pese a que la lentitud parezca un factor de segunda en cuanto a importancia se refiere, en este tipo de asuntos, se convierte en uno de primera cuando realizamos un programa que está orientado al su uso por parte de usuarios no acostumbrados al ámbito de la programación ni al manejo de datos de manera masiva. Cuando queremos entrar en una página web siempre gusta ver que las cosas funcionan, que no se quedan pensando y que si tocamos algo todo se mueva ¿no?

Dicho esto, y dado que el fichero de configuración abarca unas 500 líneas, vamos a exponer un pequeño fragmento del mismo con el objetivo de hacernos una pequeña idea de qué es lo que buscamos implementar. Por otro lado decir, que este fichero de configuración está orientado a posibles cambios en posibles futuros ciclos conforme vayamos evolucionando la aplicación.

```
"static-mls":{
"values":[{
"nameUser":"companies",
"realName":"companies",
"periods":[""]
},{
```

```
"nameUser":"diff net senders",
"realName":"diff_netsenders",
"periods":["_30", "_365", "_7"]
},{
"nameUser":"diff net sent",
"realName":"diff_netsent",
"periods":["_30", "_365", "_7"]
},{
"nameUser":"percentage senders",
"realName":"percentage_senders",
"periods":["_30", "_365", "_7"]
},{
"nameUser":"percentage sent",
"realName":"percentage_sent",
"periods":["_30", "_365", "_7"]
},{
"nameUser":"repositories",
"realName":"repositories",
"periods":[""]
},

...

}],
"periods":["","_365","_30", "_7"]
},
```

Como podemos ver en el código anterior, en el fichero de configuración tenemos los diversos datos representados conforme a su naturaleza. En este caso son datos estáticos de mls. A la hora de representar una gráfica de correspondiente a este tipo de dato el propio widget hace uso de este fichero cuya descarga se realiza nada más arrancar la aplicación mostrando las métricas

para el usuario. Cada métrica tiene un periodo asignado, de manera que a la hora de representar el dato para el usuario y para leer el dato que el usuario desea se nos muestra un sistema mucho más simple y compacto que si hiciéramos un único objeto o sección dentro del array de valores para cada métrica individual.

Por otro lado debido a que debemos adaptarnos al tipo de archivo que se quiera representar, el cual vendrá definido por la empresa, pasamos de tener una referencia como la que se tenía al principio, propia para cada dato, a realizar una referencia genérica en el archivo de configuración de la situación de todos los archivos con sus métricas.

```
"reference":"templates/JSON/"
```

Siguiendo por esta vertiente debemos tener en cuenta como ya hemos dicho que en un solo proyecto pueden llegar a trabajar 50 empresas. Es aquí donde debemos tener en cuenta la posibilidad de que se desee tener varias representaciones de datos de una misma empresa por cada uno de los paneles, lo que nos lleva a la creación de un nuevo menú con el que especificar qué empresas entran dentro de qué tipo de datos, ya que no todas tienen código fuente, o no todas hacen uso de irc.

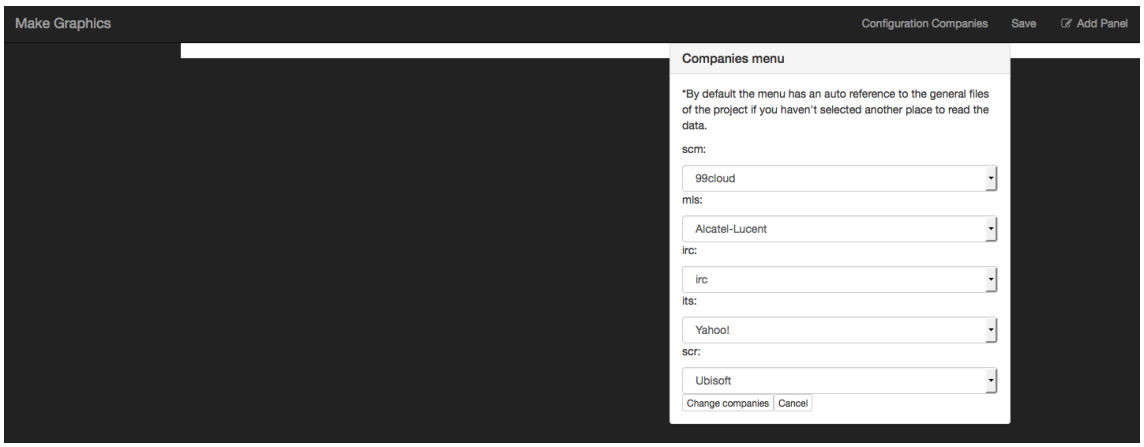

Figura 4.22: Menú de compañías

Un nuevo problema que se plantea con esto es que en la descarga de datos ahora deben existir eventos individuales por cada fichero de datos nuevo descargado. Esto es debido a que pese a existir ficheros de datos de la misma naturaleza tales como scm, irc, its... etc, etc cada uno

puede provenir de una empresa totalmente distinta, lo que nos obliga a tener un mayor control sobre el manejo de los ficheros.

4.7. Arquitectura general

El primer objetivo que planteabamos satisfacer era la creación de un DashBoard básico cuya función sea extraer los parámetros a estudiar a partir de un fichero JSON. Esta utilidad analizadora recibe por parte del usuario argumentos necesarios para su funcionamiento. El principal es la URL que direcciona el documento a examinar en el caso de que el DashBoard ya esté creado, junto con diversos parámetros que configuran aspectos relacionados con la base de datos donde depositar los resultados, véase métricas o aspectos.

El funcionamiento de la aplicación consiste en recoger la petición compuesta por los parámetros citados, solicitar la información correspondiente a las métricas del JSON en cuestión, analizarla para obtener los datos cuantificables de interés, representarlos y agruparlos con la finalidad de que permanezcan alojados en una base de datos para su posterior representación cuando se quiera volver a acceder al mismo entorno de trabajo.

Su composición interna puede dividirse, por tanto de la siguiente forma:

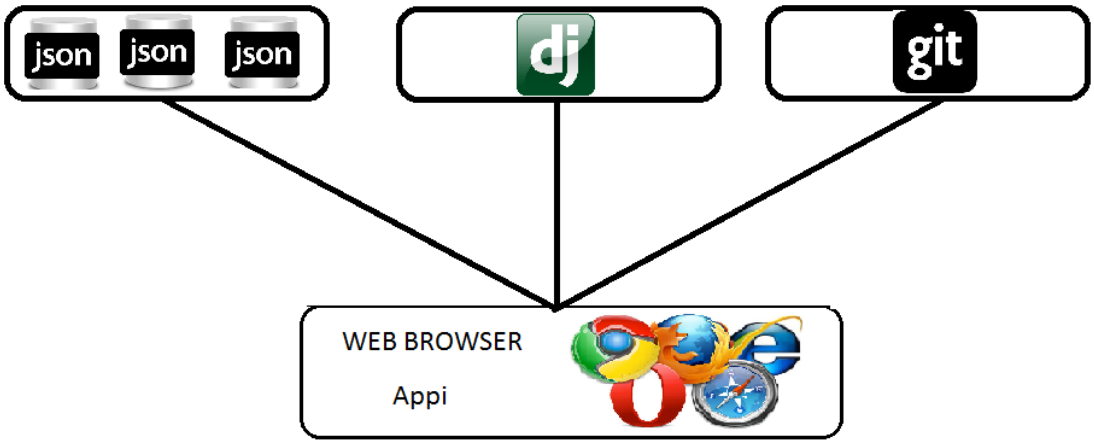

Figura 4.23: Arquitectura general

Como se puede observar en la imagen, nuestra aplicación puede hacer uso de diversas fuentes para su correcto funcionamiento. La aplicación dependiendo de si está siendo presentada desde GitHub o Django presentará los archivos necesarios para su funcionamiento de una manera totalmente distinta.

Basada o no en GitHub, las peticiones de los archivos de configuración y datos se realizarán de la misma forma (con un ”getJSON” de Jquery). Una vez se obtengan los datos se procederá

a su representación. Cuando se desee acabar esta será almacenada en un fichero en GitHub en el caso de su uso a través de la biblioteca Hello.js ó haremos un envío a Django para que realice su almacenamiento en su correspondiente lugar.

Las peticiones que se realicen de los ficheros de datos tendrán su catalizador en los propios objetos widgets que procederemos a crear cuando interactuemos con la aplicación. Muchos de estos objetos pueden depender de una misma fuente de datos, y puede darse el caso de no haber concluido con la descarga de los datos necesarios cuando se quiere crear un nuevo objeto que requiere de lo mismo.

Mediante peticiones asíncronas y la organización de la descarga a través de eventos conseguimos una descarga eficiente de los datos y una reducción de la ralentización que se pueda suceder. De esta manera un objeto realiza la petición, la descarga aún no se ha concluido pero la aplicación no se detiene, permitiendo a la máquina continuar generando nuevos objetos y nuevas peticiones, las cuales al concluir producirán un efecto de dibujado sobre el DashBoard simultáneo.

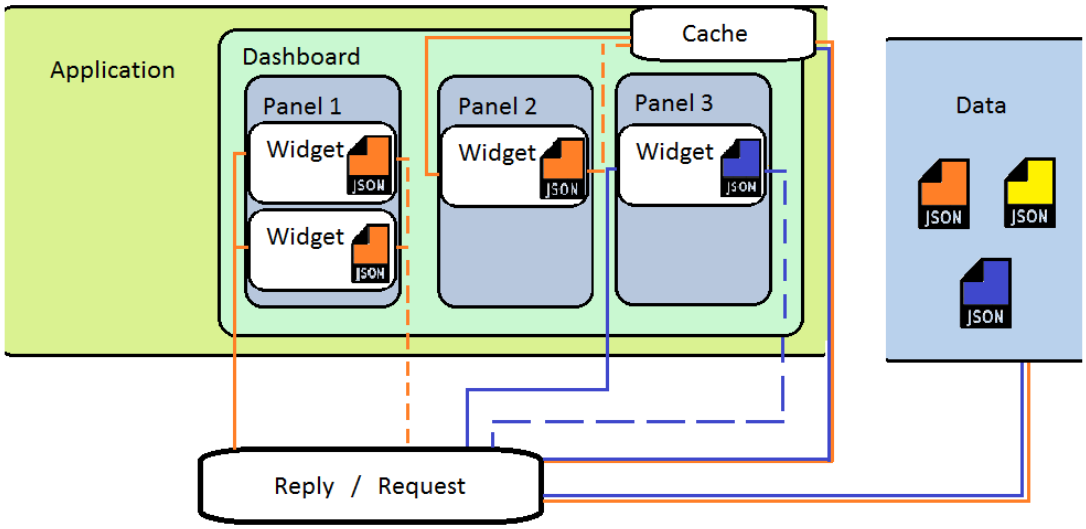

Figura 4.24: Arquitectura en objetos

Por último para poder realizar la petición de los datos que deseamos necesitamos tener conocimiento sobre la existencia o al menos el nombre de dichos datos. Al comienzo del arranque de la aplicación, anterior a la descarga de los ficheros correspondientes a los dashbaords de ejemplo (denominado 0.JSON) se realiza la descarga del fichero de configuración, a través del cual

se presentan los datos y empresas de los que podemos hacer uso. De esta manera la aplicación presenta una arquitectura en la que tenemos:

-JSON de configuración: JSON que posee las métricas clasificadas y adaptadas a la visión de un usuario. Estas métricas se encuentran en configuration.JSON

-JSON de guardado: JSON personalizado por el usuario, en el que se guarda la configuración de su DashBoard.

Y por otro lado y no por ello menos importante

-JSON de compañías: JSON que posee un objeto con las compañías de las cuales poseemos información en el directorio correspondiente. Este fichero se denomina como companies.JSON.

Capítulo 5

Conclusiones

El proyecto supone un intento por construir un conjunto de herramientas capaces de realizar la recogida de datos de distintos mecanismos de colaboración que ofrece el desarrollo abierto característico del software libre.

Podemos decir que su realización se completó con éxito, ya que han sido alcanzados todos los objetivos propuestos en un principio.

Así, se ha logrado crear una herramienta genérica para la extracción de datos desde fuentes de información disponibles públicamente, de la que pueden derivarse (siguiendo su interfaz) utilidades que desempeñan una idéntica función para fuentes específicas.

Por otra parte, los datos resultantes de este proceso son alojados en una base de datos relacional, que es una de las alternativas como formato intermedio e independiente que se requería.

El uso de las tecnologías más recientes para su desarrollo ha otorgado a la aplicación un aspecto flexible y dinámico para el usuario. Entre las tecnologías usadas habría que destacar el uso de AJAX que ha trabajado en equipo con JqueryUI permitiendo realizar una serie de operaciones de descarga y visionado de datos lo suficientemente eficientes como para evitar grandes lapsos temporales en las descargas de los distintos ficheros JSON. Estas bases de datos contenidas en dichos ficheros poseían un tamaño considerable, lo que producía el congelamiento de la aplicación si no se hubiera hecho de este modo, debido al enorme tamaño de los datos manejados.

El proyecto por otro lado puede ser aplicado del mismo modo para otros usos que no sean el seguimiento de desarrollo de un proyecto software. Esto podría ser por ejemplo el seguimiento anual de la evolución del aprendizaje de los alumnos en asignaturas como SO's, SAT o DAT

simplemente cambiando el nombre de de las variables principales de las que se van a nutrir los datos (scm, its, etc)

Como colofón, hemos de recordar que se consiguió desarrollar el proyecto exclusivamente usando software libre, incluyendo documentación y elementos gráficos.

5.1. Lecciones aprendidas

Fueron muchas las lecciones aportadas a través de la ejecución del proyecto, por lo que puede considerarse una experiencia formativa enriquecedora. Aquí se presentan algunas de ellas:

- La profundización en el lenguaje JavaScript. La programación del proyecto ha sido orientada a objetos, aumentando nuestras habilidades en la organización de la estructura del código.
- El manejo de Ajax ha sido crucial para el desarrollo del proyecto, en concreto para dar pie al almacenamiento de los datos.
- El uso de bibliotecas gráficas como Highcharts.js para la representación de los gráficos de datos.
- Refrescación del lenguaje Python, y aprendizaje del mismo orientada al uso masivo de ficheros.
- La profundización en la tecnología Django.
- Se ha aprendido también a manejar LaTeX, el sistema de procesamiento de documentos empleado en la realización de esta memoria y de otros tantos documentos en el futuro, con total seguridad.

5.2. Conocimientos aplicados

Para el desarrollo de la aplicación se han aplicado conocimientos sobre Python y JavaScript, especialmente este último, ya que suponen los lenguajes principales en los que ha sido escrito el código que hace funcionar la aplicación.

Del mismo modo se han aplicado conocimientos del framework Django y el protocolo http para la implementación de una base de datos que nos permitiera almacenar nuestros entornos personalizados.

Por último se han aplicado técnicas de búsqueda y análisis de información en distintos tipos de plataformas aprendidas durante el grado de Ingeniería de las Tecnologías de la Telecomunicación.

5.3. Trabajos futuros

Como en todos los proyectos de software, siempre existen elementos a mejorar e investigar por los participantes de este proyecto. Algunos de los que podemos comentar para aumentar las prestaciones de nuestra aplicación son:

- Optimización de tiempos: Es cierto que actualmente la aplicación funciona relativamente rápido pero el aumento del tamaño de los JSON y la aplicación a entornos de Big Data puede provocar ralentizaciones considerables en la misma.
- Seguridad: Habría que solventar problemas tales como:
  - Acceso a la base de datos
  - Modificaciones del código del sitio web
  - Ataques de denegación de servicios
- Uso en equipo: Es posible que una aplicación de este tipo sea aplicada a reuniones online entre miembros de una misma empresa o proyecto. La aplicación de tecnologías como Meteor permitiría añadir chats, videollamadas o un entorno interactivo de cambios instantáneos sincronizado entre cuentas de usuario.

Apéndice A

Instalación y uso

El modo de empleo de la aplicación es sumamente sencillo. Una vez situada en GitHub ó arrancado un servidor Django o su alternativa colocándola en algún lugar que se valga de esta misma tecnología lo primero que debemos hacer es una llamada a la url:

```
http:// ... /companies/
```

Dicha url analizará (con el servidor Django en marcha) el directorio donde se encuentran los archivos de empresas para posteriormente devolver un formato tipo objeto JSON que puede usarse para modificar el fichero de configuración y simplificar el análisis de múltiples empresas.

Una vez modificado esto solo queda arrancar la aplicación. En el caso de no poseer un fichero tipo 0.JSON indicará que no posee un fichero de configuración y que por tanto no se mostrará un DashBoard de ejemplo.

En el caso de que se quiera utilizar GitHub como base de datos hay que asegurarse de que los tokens que se usan son los adecuados conforme al directorio que se relacionará con la aplicación (línea 161 de la script DashBoard.js):

```
hello.init({
  GitHub : "CLIENT_ID"
},{
  redirect_uri : 'redirect.html',
  oauth_proxy : "https://auth-server.herokuapp.com/proxy",
  scope : "publish_files",
```

```
});
```

Por último mencionar que si se desea analizar algo externo a un desarrollo de software, se debe modificar la variable actualReadingData y sus semejantes con los nombres de los ficheros básicos que se vayan a utilizar:

```
var actualReadingData={
  "scm":"scm",
  "its":"its",
  "mls":"mls",
  "scr":"scr",
  "irc":"irc"
};
```

A.1. Requisitos

Los requisitos previos a la puesta en marcha de las herramientas son:

- *Intérprete de Python*.
  Debería funcionar con la versión 2.7.6 o posterior. Durante el desarrollo se usó la versión 2.7.6.

- *Sistema gestor de base de datos MySQL*.
  Debería funcionar con la versión 3.8.5 en adelante. Durante el desarrollo se usó la versión 3.8.5.

- *Cuenta GitHub*.
  En caso de querer dar uso a la aplicación en GitHub se debe poseer una cuenta.

Apéndice B

Formato de objeto Highchart

```
{

    _cursor : ""
    angular : undefined
    animation : true
    axes : [Object { horiz=true, isXAxis=true, coll="xAxis",
    m\'as...},Object { coll="yAxis", side=3,
    options={...},...}]
    axisOffset : [0, 0, 22, 58.416666030883796]
    bounds : Object: { h={...}, v={...}}
    btnCount : 0
    callback : undefined
    callbacks : [function(b)]
    cancelClick : false
    chartHeight : 177
    chartBackground : Object: { element=rect.highcharts-background,
    renderer={...}, x=0,...}
    chartWidth : 350
    clipBox : Object: { x=0, y=0, width=272,...}
    clipOffset : [0, 0, 0, 0]
    clipRect : Object: { element=rect, renderer={...},
```

```
parentGroup={...},...}
colorCounter : 2
container : div#highcharts-2.highcharts-container
containerHeight : 177
containerWidth : 350
credits : Object: { element=text, renderer={...},
textStr="Highcharts.com",...}
detachedinit : null
detachedredraw : null
exportDivElements : []
exportSVGElements : [Object { element=g.highcharts-button,
renderer={...}, styles={...}, mas...}, Object { element=path,
renderer={...}, d="M 6 6.5 L 20 6.5 M 6 11....11.5 M 6 16.5 L 20
16.5",...}]
hasCartesianSeries : true
hasRendered : true
hasUserSize : null
index : 1
inverted : undefined
isDirtyBox : false
isDirtyLegend : false
isResizing : 0
jQuery : 111104296875597222908 Object { events={...},
toJSON=function(), handle=function()}
legend : Object { options={...}, itemStyle={...}, itemHiddenStyle={..
margin : [undefined, undefined, undefined, undefined]
marginBottom : 75
marginRight : 10
maxTicks : null
mouseDownX : 33
mouseDownY : 46
```

```
mouseIsDown : false
oldChartHeight : null
oldChartWidth : 350
options : Object { colors=[10], symbols=[5], lang={...},
...}
plotBorderWidth : 0
plotBox : Object { x=68, y=47, width=272,...}
plotHeight : 55
plotLeft : 68
plotSizeX : 272
plotSizeY : 55
plotTop : 47
plotWidth : 272
pointCount : 2
pointer : Object { options={...}, chart={...}, zoomX=false,
...}
polar : undefined
reflowTimeout : 23
renderTo : div#2.panel-body
renderer : Object { style={...}, isSVG=true, box=svg,
...}
series : [Object { chart={...}, userOptions={...},
tooltipOptions={...},...}, Object { chart={...},
userOptions={...}, tooltipOptions={...},...}]
seriesGroup : Object { element=g.highcharts-series-group,
renderer={...}, zIndex=3,...}
spacing : [10, 10, 15, 10]
spacingBox : Object { x=10, y=10, width=330, ...}
symbolCounter : 0
title : Object { element=text.highcharts-title, renderer={...},
textStr="Grafico 2",...}
```

```
    titleOffset : 22
    tooltip : Object { chart={...}, options={...}, crosshairs=[0],...}
    userOptions : Object { chart={...}, xAxis={...}, title={...},
    ...}
    xAxis : [Object { horiz=true, isXAxis=true, coll="xAxis",
    ...}]
    yAxis : [Object { coll="yAxis", side=3, options={...},
    ...}]
}
```

Apéndice C

Implementación, funciones importantes (Iteración 2)

Debido a que en esta parte tocamos el código más en profundidad es necesario abrir una segunda sección desde la cual explicar cómo está estructurado el código.

Todo nuestro código se basa en 3 partes distintas correspondientes a cada uno de los widgets. Cada parte a su vez se subdivide en una serie de funciones de funcionamiento semejante pero especializada en cada widget, lo que nos permite ser más claros en el código y traducirlo a un formato de objetos si fuera necesario en un futuro. Para verlo más claro partiremos del widget de funcionamiento más simple que tenemos ahora mismo, el estático y estudiaremos sus funciones principales y más importantes que hacen que pueda ser dibujado.

showInfoSettings():

Función que permite mostrar el menú de creación de una gráfica estática. Símplemente crea en el árbol DOM un html destinado a ser un formulario con el que permitir al usuario que use la aplicación seleccionar lo que le interesa representar.

```
function showInfoSettings(dash){

    keys=configuration.static

    $("#settings"+dash).slideUp("slow");
```

```
$("#making").slideDown("slow")

if((Object.getOwnPropertyNames(takeinfo).length === 0)
|| (Object.getOwnPropertyNames(configuration).length === 0)){


    $("#conten").append('<div id="actualMenu"></div>')

    $("#actualMenu").append('<p>Los ficheros de configuracion
    no han sido cargados con normalidad. Compruebe su
    conexion<p>');

    $("#actualMenu").append('<button onclick="deleteCreation()"
    type="button" class="btn btn-xs btn-default">
    Continue</button>')

}else{

    if($("#actualMenu")){

        $("#actualMenu").remove();

    }

    $("#conten").append('<div id="actualMenu">
    <p id="list" style="height: 200px; overflow-y: scroll;">
    </p></div>')

    $(''#actualMenu'').append('<p><input type=''radio''
    name=''toSeeOptions'' class=''radios'' value=''column''>
```

```
        Columnas <input type=''radio'' name=''toSeeOptions''
        class=''radios'' value=''bar''>Barras </p>');

        keys.forEach(function(element){


            $("#actualMenu #list").append('<p><input
            type="checkbox" value="'+element+'"> '+element+'</p>')



        })

        $("#actualMenu").append('<button onclick="takeDataInfo()"
        type="button" class="btn btn-xs btn-default">
        Create</button>')

        $("#actualMenu").append('<button onclick="deleteCreation()"
        type="button" class="btn btn-xs btn-default">
        Cancel</button>')
    }
}
```

takeDataInfo():

Función que permite obtener los datos que necesitamos del menú creado en la función anterior, ligado al botón ”Create” situado en la parte inferior del menú.

```
function takeDataInfo(numGraph){

    var forma=$("#actualMenu input[type='radio']:checked")
    .attr("value");
```

```
if(forma!=undefined){

    var selected = [];

    var forma=$("#actualMenu input[type='radio']:checked")
    .attr("value");

    $("#actualMenu input[type='checkbox']:checked")
    .each(function() {

        objaux={

            name:$(this).attr('value'),
            form: forma

        }

        selected.push(objaux);

    });

    if(selected.length!=0){

        makeGraphInfo(actualDash,selected,numGraph)

    }else{

        alert("Â¿Que clase de gr\'afica
        pretende representar sin datos?")
```

```
        }

    }else{

        alert("Es indispensable que seleccione
        un tipo de forma a representar")

    }
}
```

makeGraphInfo():

Función que nos permite crear el widget y dibujar su contenido. Hagamos especial énfasis en esta parte del código la cual resulta más interesante. Como hemos mencionado en este mismo apartado, existen problemas de sincronicidad a la hora de dibujar una gráfica una vez obtengamos un dato en concreto del JSON necesario. Gracias a las funciones de jquery .on() y .trigger() este problema se vio rápidamente solucionado permitiéndonos dibujar varias gráficas de manera simultánea o continuar creando widgets hasta que la descarag de los ficheros se completara.

```
function makeGraphInfo(dash,selected,graph){

    if(isNaN(graph)){

        numGraph+=1;

        graph=numGraph;

        var inDash= "dash"+dash.toString()

        var gridster = $("#"+inDash+" ul").gridster().data('gridster');
```

```
        color=($("#dash"+dash).css('background-color'))

        gridster.add_widget('<div id= "graph'+numGraph+'"
        class="panel panel-primary" style="border-style:
        groove;border-color: black;border-width: 3px">
        <div class="panel-heading" style="background-color:'+color+'">
        <button id="deleteButton"onclick="deleteGraph('+dash+','+graph+')
        <button id="settingsButton"onclick="settingsInfoGraph('+numGraph+
        </div><div id="'+numGraph+'" class="panel-body"> </div></div>',
        12, 8);

    }else{

        var chart = $('#'+graph).highcharts();

        chart.destroy()

    }

    $("#actualMenu").remove();

    $("#making").slideUp("slow")

    $("#"+graph).on("DrawInfo",function(event,trigger,data){

        var serie= parserGraphInfo(selected,data)

        drawGraphInfo(graph,serie,takeinfo.static.inside)

        $("#"+this.id).off()
```

```
})

$("#"+graph).on("ErrorGraphInfo",function(){

    drawErrorWidget(this.id)

    $("#"+this.id).off()

})

if(takeinfo.static.state==0){

    takeinfo.static.state=1;

    $.getJSON("templates/JSON/"+takeinfo.static.inside).success(funct

        takeinfo.static.saveData=data

        takeinfo.static.state=2;

        $("*").trigger("DrawInfo",["DrawInfo",data])

    }).error(function(){

        $("*").trigger("ErrorGraphInfo")

        takeinfo.static.static=0;

    });
```

```
    }else if(takeinfo.static.state==2){

        var serie= parserGraphInfo(selected,takeinfo.static.saveData);

        drawGraphInfo(graph,serie,takeinfo.static.inside)

    }

}
```

drawGraphInfo():

Función que permite dibujar el widget una vez lo tengamos todo cargado y los datos estén parseados. Aquí vemos un ejemplo de aplanamiento de datos simple donde se puede comprobar la simplicidad que ofrece Highcharts para dibujar un gráfico de manera sencilla y rápida.

```
function drawGraphInfo(id,serie,title){

    var options={

        chart:{

            renderTo:id.toString(),
            width: 350,
            height: 177

        },

        xAxis: {

            categories: ["Total"]
```

```
        },
        title: {
            text: title
        },
        series: serie
    }
    var chart= new Highcharts.Chart(options);
}
```

Apéndice D

API

En esta sección pasaremos a realizar un ligero paseo sobre el formato de uso de nuestra aplicación. Aquí detallaremos el uso correcto de la arquitectura montada y la interfaz de los objetos, así como crear otros nuevos para mejora y ampliación de las prestaciones que ofrecemos.

D.1. Objeto panel

Situado en el programa principal, podemos hacer uso del botón situado arriba a la derecha para crear un nuevo panel.

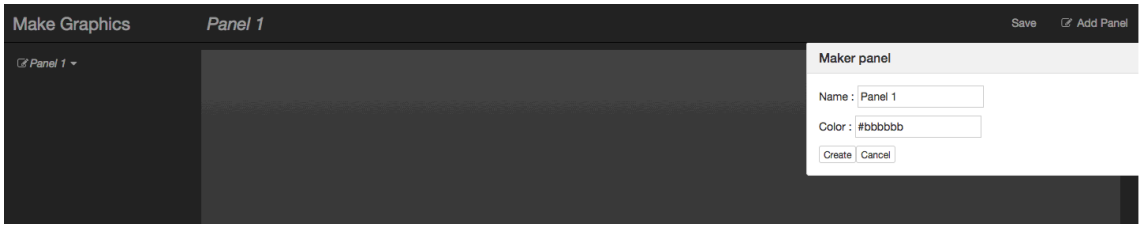

Figura D.1: Creación de un panel

Dicho botón hará uso de la siguiente función:

```
PanelCreation(color,name,reading)
```

Esta función aquí presentada creará y ajustará los parámetros necesarios para la creación de

nuestro panel sobre el árbol DOM (véase creación del gridster entre otros). El objeto creado durante el proceso tendrá la siguiente interfaz:

```
Panel(idPanel,color,name,reading,wid)
```

A la hora de crearlo necesitamos el identificador del panel (que no será otra cosa que el número del panel creado, el color que vamos a establecer de fondo, el nombre que vamos a darle, el formato por defecto del que vamos a leer y una última variable denominada ”wid” correspondiente a un posible array de creación con los widgets que posee dentro necesaria en el caso de cargado de un DashBoard situado en la base de datos. De esta manera tiene.

- writeSettings(): Función de creación de los botones correspondientes a los widgets que es capaz de crear. Por defecto esta función pintará en la zona correspondiente del actual DashBoard.
- getSettings(): Función que devuelve las opciones del panel en cuestión.
- getContent(): Función que devuelve el contenido del gridster en sí.
- getId(): Función que devuelve la ID del panel.
- getWidgets(): Función que devuelve el array de los widgets del panel.
- flatten(): Función que permite aplanar el panel en sí, devolviendo un objeto con las características del panel y los widgets que estén dentro del mismo aplanados.
- pushElement(): Función que inserta dentro del array de widgets del panel un nuevo widget.
- deleteElement(): Función que elimina un widget del array de widgets del panel.
- deleteAll(): Función que elimina todos los widgets del array de widgets del panel.

La interfaz cara vista al usuario ofrece de este modo:

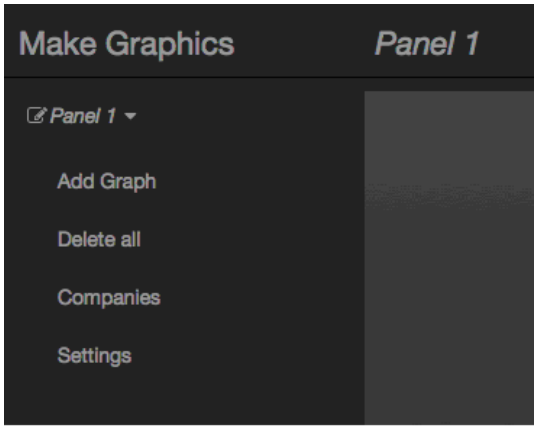

Figura D.2: Opciones del panel

- Add Graph: Abre el panel de widgets para añadir una nueva gráfica.
- Delete All: Elimina todos los widgets dibujados en los paneles.
- Companies: Despliega el menú de compañías para reconfigurar el panel.
- Settings: Permite cambiar algunas de las opciones del panel.

D.2. Objeto widget

Para poder diseñar un nuevo widget debemos abrir el menú relativo al mismo que podemos encontrar en la sección de opciones de cada panel, en el botón Add Graph:

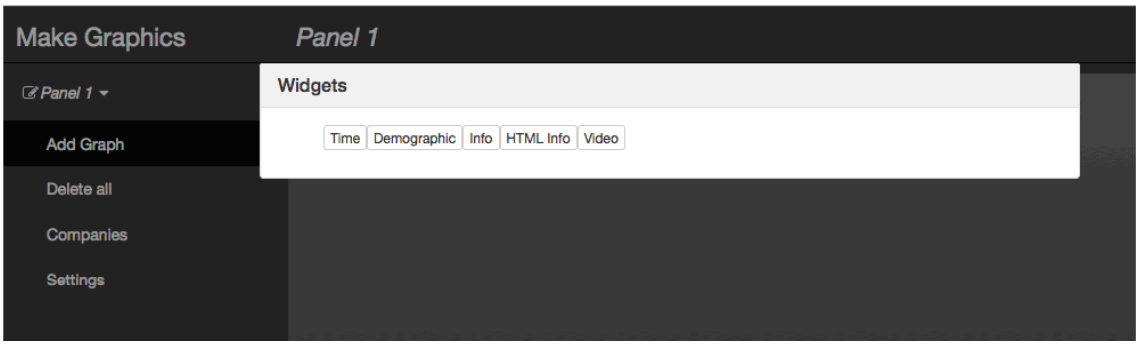

Figura D.3: Menú de widgets

Cada sección se subdivide en los posibles formatos de datos que puedan ser representados, a excepción de los widgets como los de video.

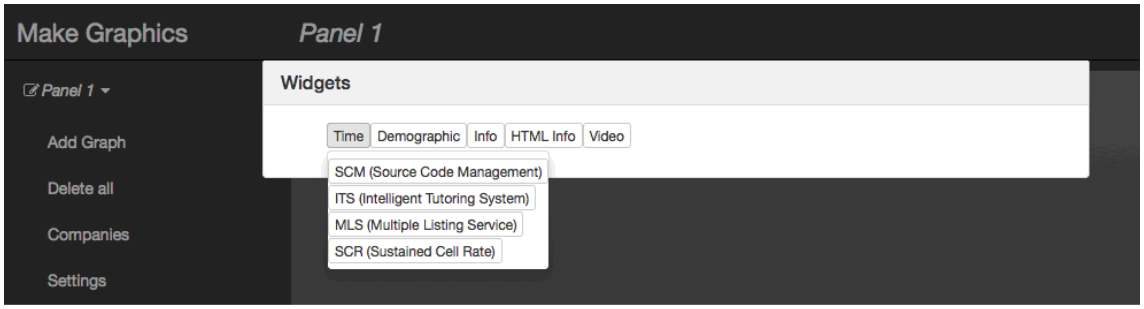

Figura D.4: Menú de cada widget

Cada sección se subdivide en los posibles formatos de datos que puedan ser representados, a excepción de los widgets como los de video.

Si seleccionamos una de las opciones se abrirá un menú con el que podremos diseñar dicho panel:

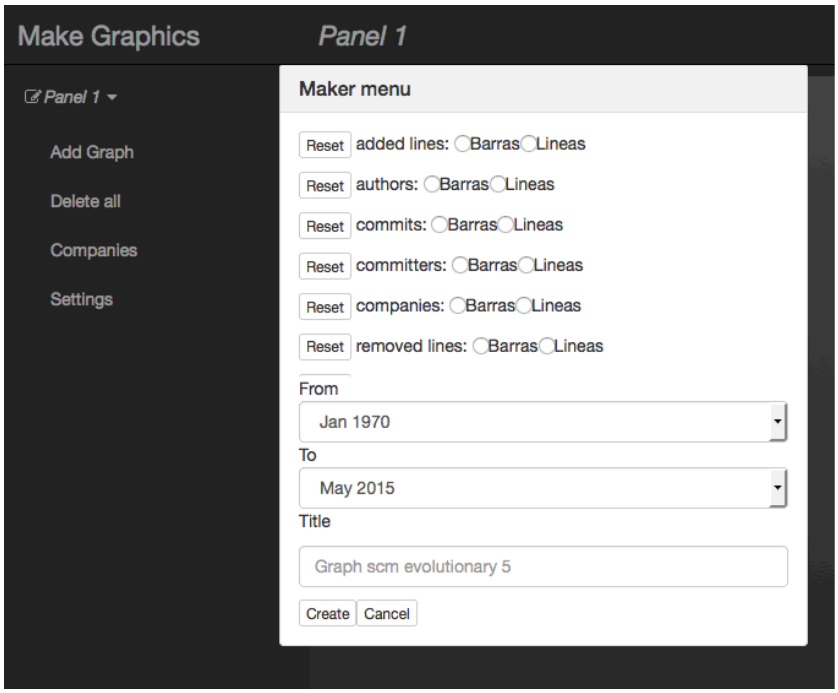

Figura D.5: Menú del wiget de datos de evolución temporal

Los objetos widgets de este modo se subdividen en:

Gráfico de representación demográfica del proyecto: HighDemo(id,panel,color,typeData,ficheros JSON,title,serie,x,y)

- Id: Identificador del widget

- Panel: Referencia al panel al que pertenece (identificador del panel)
- Color: Color del menú del widget
- typeData: Tipo de dato del que va a leer (scm,its,ircâ)
- ficheros JSON: JSON del que va a leer
- title: Título de dicho wiget
- serie: Serie aplanada sobre los datos que se van a leer
- x: tamaño de la altura del widget
- y: tamaño de la anchura del widget

*No posee sección de datos de lectura debido a que la gráfica demográfica corresponde a la demografía del proyecto general, no de cada empresa individual.

Gráfico de representación de datos estáticos:HighInfo(id,panel,color,typeData,readingData,JSON,title,serie

- Id: Identificador del widget
- Panel: Referencia al panel al que pertenece (identificador del panel)
- Color: Color del menú del widget
- typeData: Tipo de dato del que va a leer (scm,its,ircâ)
- readingData: Configuración de los datos de lectura por compañías
- ficheros JSON: JSON del que va a leer
- title: Título de dicho wiget
- serie: Serie aplanada sobre los datos que se van a leer
- x: tamaño de la altura del widget
- y: tamaño de la anchura del widget

Gráfico de representación de datos estáticos sin representación gráfica: HtmlInfoWidget(id,panel,color,type

- Id: Identificador del widget
- Panel: Referencia al panel al que pertenece (identificador del panel)
- Color: Color del menú del widget
- typeData: Tipo de dato del que va a leer (scm,its,ircâ)
- readingData: Configuración de los datos de lectura por compañías
- ficheros JSON: JSON del que va a leer
- title: Título de dicho wiget
- serie: Serie aplanada sobre los datos que se van a leer
- x: tamaño de la altura del widget
- y: tamaño de la anchura del widget

Gráfico de representación de datos temporales: HighTime(id,panel,color,typeData,readingData,JSON,title,s

- Id: Identificador del widget
- Panel: Referencia al panel al que pertenece (identificador del panel)
- Color: Color del menú del widget
- typeData: Tipo de dato del que va a leer (scm,its,ircâ)
- readingData: Configuración de los datos de lectura por compañías
- ficheros JSON: JSON del que va a leer
- title: Título de dicho wiget
- serie: Serie aplanada sobre los datos que se van a leer
- x: tamaño de la altura del widget
- y: tamaño de la anchura del widget
- from: Acotación de fecha inferior

- to: Acotación de fecha superior

Widget de inserción de video:VideoWidget(id,panel,color,direction,content,width,height,x,y)

- Id: Identificador del widget
- Panel: Referencia al panel al que pertenece (identificador del panel)
- Color: Color del menú del widget
- direction: Dirección del video
- content: Contenido html del video embebido
- width: Ancho del video
- heigh: Alto del video
- x: tamaño de la altura del widget
- y: tamaño de la anchura del widget

Una vez creados todos los widgets responden al siguiente formato de funciones:

- flatten(): Aplanamiento de la información del widget
- makeMenu(): Función de creación del menú en la sección correspondiente
- takeData(): Función de recogida de datos del formulario que crea la función ”makeMenu”
- makeWidget(): Función de representación del widget sobre el árbol DOM
- settings(): Función que abre el menú de opciones del widget ya creado, dando la oportunidad de modificarlo a posteriori.
- Redraw(): Función que redibuja un widget ya creado, normalmente utilizada después de haber cambiado sus opciones.

Estos widgets orientados como objetos heredan a su vez del objeto padre: Widget(id,panel,color,x,y)

- Id: Identificador del widget

- Panel: Referencia al panel al que pertenece (identificador del panel)
- x: tamaño de la altura del widget
- y: tamaño de la anchura del widget

En el caso de los gráficos que manejan datos provenientes de empresas y que pueden ser personalizados conforme a las métricas que se seleccionen, pueden existir como ya se han comentado, problemas de sincronicidad a la hora de realizar la descarga de los ficheros. Esto es descarga de un mismo fichero por parte de distintos widgets o espera a que el mismo complete su envío. Para evitar problemas se han ligado eventos a la descarga de los datos, que manejan el contenido una vez la descarga se ha completado, o se mantienen en espera si aún no hubo respuesta. Estas funciones se pueden encontrar en la documentación de Jquery.

- .trigger(eventType [, extraParameters]) : Ejecuta los manejadores y sus funciones ligadas a los elementos que coinciden con el evento que se acaba de dar.
- .on(events [, selector] [, data], handler) : Liga un evento y su manejador a los elementos seleccionados.

Bibliografía

[1] Mark Lutz. *Programming Python*. O'Reilly, 2001.

[2] Fredrik Lundh. *Python standard library*. O'Reilly, 2001.

[3] Referencia teórica web.

`http://es.wikipedia.org`

[4] Django.

`https://docs.djangoproject.com/en/1.8/`

[5] HTML5.

`http://www.w3.org/html/`

[6] CSS.

`http://www.w3.org/TR/CSS/`

[7] Bootstrap.

`http://getbootstrap.com/2.3.2/getting-started.html`

[8] JavaScript.

`http://www.ecmascript.org/docs.php`

[9] Jquery.

`http://api.jquery.com/`

[10] Python.

`https://docs.python.org/2/faq/`

[11] AJAX.

http://web.archive.org/web/20081012084441/http://sherekan.com.ar/blog/2008/04/19/introduccion-a-ajax/

[12] Highcharts.

http://www.highcharts.com/docs

[13] MetricsGrimoire.

http://metricsgrimoire.GitHub.io/

[14] vizGrimoire.

http://vizgrimoire.bitergia.org/

[15] OpenStack.

http://docs.openstack.org/

[16] Grupo de Sistemas y Comunicaciones - Universidad Rey Juan Carlos.

http://gsyc.urjc.es

[17] Web del proyecto Libre Software Engineering - GSYC.

http://libresoft.urjc.es