UNIVERSIDAD
REY JUAN CARLOS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Curso Académico 2014/2015

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

WEB DEBIAN COUNTING

Autor : Roberto Rodríguez Alonso

Tutor : Dr. Gregorio Robles

Dedicado a

mi familia y mi novia

por todo el esfuerzo y el apoyo

II

Resumen

Este trabajo desarrolla una aplicación web que muestra estadísticas y números reunidos por SLOCCount. SLOCCount cuenta el número de líneas de código fuente en software y utiliza el modelo COCOMO para el cálculo de las estimaciones de lo previsto, el esfuerzo en personas-meses, los desarrolladores necesarios y el coste total del desarrollo.

El objetivo principal del proyecto es renovarlo con las nuevas tecnologías implementadas en estos últimos años, ya que este proyecto se realizó por el Grupo de Sistemas y Comunicaciones (GSyC) de la Universidad Rey Juan Carlos en 2003. El proyecto se ha realizado desde cero con el único uso de las bases de datos aportadas por el proyecto anterior.

Para el desarrollo de este proyecto hemos utilizado recursos de HTML5 y CSS3 para crear una interfaz más atractiva. Por otro lado, cabe destacar la utilización de caché para proporcionar rapidez en la respuesta al usuario. Además pasa a ser una web dinámica, gracias a la interacción AJAX del servidor con el cliente. Finalmente, la aplicación se termina llevando a producción con un servidor Apache y una maquina virtual proporcionada por el grupo Libresoft de la Universidad Rey Juan Carlos.

Summary

This paper describes a web application that shows statistics and numbers gathered by SLOCCount software. SLOCCount counts the number of lines of the software source code and uses the COCOMO model to estimate Project management factors such as the effort measured in Person-Months, number of developers needed and the total cost of development.

The main objective of this Project is to renew this application, applying state of the art technologies. This work was carried out by the Systems and Communications Group (GSyC) at Rey Juan Carlos University, in 2003. The application proposed is a completely new software, which only shares databases provided by prevous projects.

For the development of this Project HTML5 and CSS3 have been used to create a more attractive interface. On the other hand, we should underline the use of cache providing a quickly response to the user. Moreover, it has now become a dynamic web, thanks to the AJAX server interaction with the client. Finally, the application is carried out production with Apache server and virtual machine provided by Libresoft group at Rey Juan Carlos University.

Índice general

Índice de figuras

Capítulo 1

Introducción

Está aplicación ofrece estadísticas y datos obtenidos con la herramienta SLOCCount [8], a través de una interfaz web dinámica. SLOCCount cuenta el número de líneas de código fuente (por definición una línea que termina en un salto de línea o indicador de fin de archivo, y que contiene al menos un carácter que no está en blanco o que no sea un comentario) en software. Se realizan estimaciones estadísticas con el modelo COCOMO que pueden ser utilizados para tener una idea económica de lo que implica un proyecto de desarrollo software. SLOCCount-web[1] ha sido implementado como parte del proyecto de desarrollo de software libre de Ingeniería de la Universidad Rey Juan Carlos [9] [10].

Debido a los grandes cambios en la web en los últimos años, el usuario está viviendo en un entorno más dinámico y con una interfaz más vistosa. Este proyecto está orientado a analistas y desarrolladores de software, ya que es una web que muestra de forma ordenada los datos obtenidos de las distintas versiones del sistema operativo Debian a través de la herramienta SLOCCount.

En este apartado detallaremos los conceptos fundamentales en los que se basa la aplicación web y el entorno de trabajo.

[1] es una interfaz web para los datos generados por SLOCCount

1.1. ¿Qué es SLOCCount?

SLOCCount (*Source Lines of Code Count*) [8] es un conjunto de herramientas que analiza proyectos de software y se utiliza para el recuento de líneas de código fuente físicas en un gran número de lenguajes. Además, es un proyecto de software libre y está liberado bajo la Licencia Pública General (*GPL*), por lo que puede ser descargado gratuitamente y modificado para adaptarlo a las necesidades de cada usuario.

SLOCCount puede detectar automáticamente 27 lenguajes de programación e incluye una serie de heurísticas para que pueda detectar los tipos de archivos, incluso de aquellos que no utilizan las extensiones más corrientes. Los contadores de SLOC[2] tienen inteligencia suficiente para manejar rarezas de varios idiomas, examinan los archivos en lenguaje ensamblador, determinan cuando se utiliza un comentario y luego cuentan las líneas correctamente de forma automática. Utiliza el modelo COCOMO con el que es capaz de medir el esfuerzo, número de desarrolladores...

El proyecto fue implementado por el estadounidense David A. Wheeler, debido a la necesidad de analizar el proyecto Red Hat Linux y sus resultados fueron publicados en el año 2001.

[2]Líneas de código fuente

1.2. COCOMO

COCOMO o MOdelo COnstructivo de COstes (*COnstructive COst MOdel*) es un modelo matemático empírico que se utiliza para la estimación del coste de un software y está orientado a la magnitud del producto final. Existen tres submodelos, cada uno con un nivel de detalle, y en nuestro caso hemos utilizado el modelo básico que proporciona una impresión general del proyecto. Además, dentro de este existen tres y el elegido ha sido el modo orgánico.

Ecuaciones utilizadas:

- Esfuerzo (Personas-Mes) = $2,4 * KSLOC^{1,05}$
- Tiempo de desarrollo (Meses) = $2,5 * KSLOC^{0,38}$
- Personas = $\frac{Esfuerzo}{Tiempodedesarrollo}$

Finalmente, se estima que los costes del sueldo promedio de los desarrolladores a tiempo completo en el año 2000, según una encuesta salarial publicada por Computer World , se ha determinado que el sueldo era de 56.286 dólares al año.

Este modelo fue desarrollado por B. W. Boehm entre finales del año 1970 y comienzos de 1980, exponiéndolo detalladamente en su libro *Software Engineering Economics* (*Prentice-Hall, 1981*).

1.3. LAMP

El acrónimo LAMP fue originalmente acuñado para describir un conjunto de software Open Source[3] utilizado para propulsar muchos sitios web:

- **Linux** (*sistema operativo*)
- **Apache** (*servidor web*)
- **MySQL** (*base de datos*)
- **PHP** (*Lenguaje de programación*), en nuestro caso Django y Python.

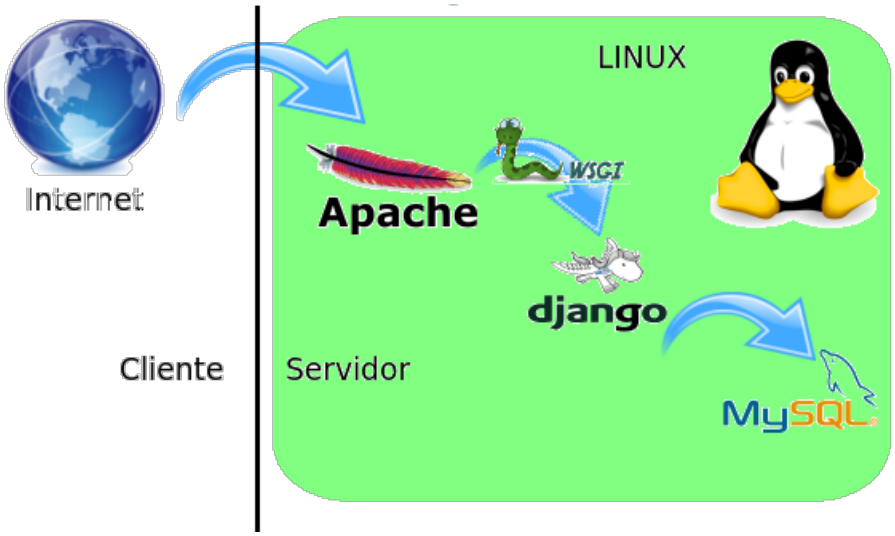

Figura 1.1: Estructura LAMP. (Elaboración propia)

El acrónimo se ha convertido más una referencia a este tipo de arquitectura software que a cualquiera de las tecnologías nombradas anteriormente. La tecnología LAMP utilizada por Django es la llamada shared nothing (*nada compartido*).

La filosofía *shared nothing* consiste en una arquitectura en la que cada nodo es independiente y autosuficiente, lo que proporciona al sistema una alta escalabilidad. Este sistema es muy efectivo para pequeñas empresas, ya que tiene un bajo coste y no necesita un servidor muy potente, que es lo que sucedía con el sistema monolítico. Algunas grandes empresas han basado su negocio en esta arquitectura, como Amazon, Facebook, Google, Wikipedia, Yahoo, Youtube...

[3] Código abierto

1.4. Estructura de esta memoria

Con el fin de facilitar la lectura de esta memoria, es necesario explicar brevemente la estructura de cada capitulo.

- Capítulo 1 **Introducción**: en este capítulo se realiza una descripción precisa del proyecto y del contexto tecnológico utilizado en él.
- Capítulo 2 **Objetivos**: Describe los objetivos y las tareas marcadas para el desarrollo.
- Capítulo 3 **Tecnologías utilizadas**: En este capítulo se detallan cada una de las distintas tecnologías utilizadas, para una mejor compresión de la implementación y el desarrollo de la aplicación.
- Capítulo 4 **Implementación y desarrollo**: En este capítulo se explica el funcionamiento y desarrollo del proyecto. Se describen los bloques implementados en la aplicación y las soluciones, a los distintos problemas encontrados.
- Capítulo 5 **Conclusiones**: El último capítulo de la memoria detalla las decisiones que se han tomado en la implementación del proyecto. Además se describen los conocimientos aprendidos y aplicados al proyecto. Finalmente, se plantean posibles mejoras en futuras líneas de trabajo.

Capítulo 2

Objetivos

2.1. Descripción del problema

Tenemos la necesidad de analizar los proyectos de software para sacar el máximo rendimiento. Por ello, tenemos la herramienta SLOCCount que nos realiza un pequeño análisis de un proyecto software.

Cuando obtenemos todos los datos necesarios para analizar el proyecto necesitamos mostrarlos de forma ordenada y detallada.

Debido a los cambios tecnológicos se necesitan mostrar los datos en una plataforma que pueda ser visionada por cualquier persona, por lo tanto el proyecto debe ser mostrado en el sitio más globalizado del mundo, la web.

2.2. Objetivo principal

El objetivo principal del proyecto es mostrar los datos (que se han obtenido con la herramienta SLOCCount), guardados en una base de datos en formato `".sql"`, de una forma analítica y que pueda ser entendida por el usuario. Además, debe ser una aplicación dinámica y con una interfaz atractiva y de fácil uso para el usuario.

2.3. Objetivos específicos

Los datos que se quieren mostrar se dividen en cuatro grandes estructuras:

- **Statistics**: en este apartado aparecen las macroestadísticas, estadísticas por el tipo de lenguaje y macroestadísticas COCOMO.
- **Packages**: aquí encontraremos una lista ordenada de todos los paquetes con sus estadísticas y cada paquete puede ser seleccionados por su nombre o número de ficheros.
- **Graphs**: podemos observar con gráficas las estadísticas mostradas en los apartados anteriores para comprenderlas en un formato visual.
- **Buscador de paquetes**: nos mostrará información analítica de cada paquete, diferenciando cada lenguaje de programación para ese paquete y un análisis COCOMO.

Finalmente, todo estará integrado en la web principal y solo se modificará uno de los contenedores del HTML, según la información que nos llega en formato JSON.

Capítulo 3

Tecnologías utilizadas

3.1. HTML5

HTML5 (*HyperText Markup Language, versión 5*) es la última versión del lenguaje básico de la World Wide Web (*www*), HTML.

Esta nueva versión nos ofrecerá nuevos elementos, atributos y comportamientos, que nos proporcionarán sitios y aplicaciones web más dinámicas y diversas.

Los nuevos recursos que aparecen son los siguientes:

- **Semánticamente**, aparecen nuevos elementos que permiten describir con mayor precisión cual es su contenido: `<section>`, `<article>`, `<nav>`, `<header>`, `<footer>` y `<aside>`, desaparece la etiqueta `<hgroup>`.

- **Conectividad**, ofrece al servidor una comunicación de forma permanente intercambiando datos en formato no HTML, permite al servidor colocar eventos en un cliente, en lugar de la antigua respuesta del servidor solo a petición del cliente, y conectar a personas por videoconferencia sin necesidad de una aplicación externa.

- **Desconectado y almacenamiento**: las páginas web pueden almacenar datos localmente, en el lado del cliente y operar fuera de línea de manera más eficiente.

- **Multimedia y gráficos**: incorpora etiquetas como `<canvas>` con el que podemos dibujar gráficos en 2D y 3D. Otras etiquetas son `<audio>` y `<video>` que pueden manipular

y almacenar imágenes de nuestra cámara del ordenador.

- Podemos acceder a los dispositivos de nuestro ordenador como la cámara y el micrófono, de esta forma conseguimos crear videoconferencias.

- Tenemos nuevos elementos como Drag y Drop, que nos permiten arrastrar objetos como imágenes o elementos de geolocalización, que nos proporcionan la situación geográfica de nuestro cliente.

- **Web workers** es una API[1] que delega la evaluación de Javascript para subprocesos en segundo plano, evitando que estas actividades ralenticen eventos interactivos.

Finalmente, el estilo de nuestra web puede ser modificado con CSS3, que detallamos más adelante.

3.2. Javascript

Hace unos años la velocidad de navegación era muy lenta y surgió la idea de crear un lenguaje de programación que se ejecutara en el navegador del cliente, de esta forma Brendan Eich (programador de Netscape[2]), pensó que podría realizar este lenguaje con otras tecnologías existentes como era el caso de ScriptEase (lanzamiento previsto para 1995), llamándolo LiveScript. Más tarde, Netscape firmó una alianza con Sun Microsystem para el desarrollo de este lenguaje, pero decidió cambiarle el nombre por el de Javascript, debido exclusivamente al marketing por la popularidad de Java.

Es un lenguaje de programación interpretado, a veces abreviado como `js`. Es un lenguaje script multi-paradigma orientado a objetos, basado en prototipos, dinámico, soporta estilos de programación funcional y débilmente tipado.

Para interactuar con una página web se dota al lenguaje Javascript del árbol DOM (*Document Object Model*). El DOM es una API (*Application Programming Interface*) que nos propor-

[1] Interfaz de programación de aplicaciones

[2] Navegador web

ciona un modelo estándar de objetos para representar elementos HTML y XML[3], de esta forma con Javascript podemos acceder y manipular el árbol DOM. El DOM puede ser representado como un árbol genealógico, en nuestro proyecto tenemos un caso claro del árbol que puede ser formado a partir del elemento `<table>`:

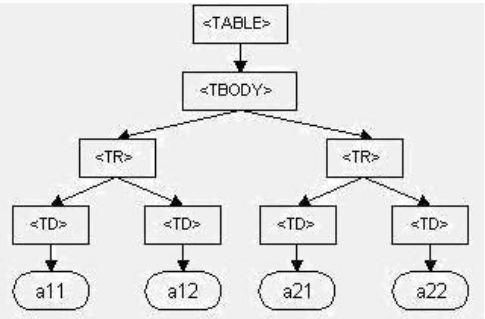

Figura 3.1: Árbol DOM de una etiqueta `<table>`. Fuente: `http://www.irt.org/articles/js143/319x209xtable.jpg.pagespeed.ic.90IRcn_9J7.webp`

De esta forma con Javascript podemos realizar cambios en la estructura DOM mediante un manejador de eventos, por ejemplo podríamos realizar una función que ingresando un número en `a11`, otro en `a12` y clickamos en el elemento table, el manejador captura el evento con onclick y este realiza una función que nos sume los elementos obteniendo el resultado en `a21` y eliminando el elemento `a22` y su padre `<td>`. De esta forma hemos cambiado la estructura del árbol DOM dejando el elemento tbody con 3 nietos en lugar de cuatro.

3.2.1. Jquery

Jquery [2] es una de las bibliotecas más utilizadas de Javascript. Podemos simplificar la forma de manejar eventos, manipular el árbol DOM, crear animaciones o interactuar con AJAX[4]. Es una biblioteca Open Source, por eso puede ser utilizada tanto para proyectos libres, como para proyectos privados.

Una de las características más importantes es la función `$()`, que permite manipular el árbol DOM y cambiar el contenido de la web con el uso de AJAX, sin necesidad de recargar la página. La forma de interactuar es simple introducimos dentro del paréntesis la estructura

[3] lenguaje de marcas extensible

[4] JavaScript asíncrono y XML

que queramos modificar, por ejemplo si introducimos `$("#principal")`, nos devolverá el elemento con `id = #principal;`. Una vez hemos establecido el elemento que queremos seleccionar le añadimos la función que queremos realizar, por ejemplo:
`$("#principal").append("Fin");`
Con esa sentencia hemos añadido al elemento HTML el texto ”Fin”.

Otra característica importante es que para ejecutar cualquier modificación en nuestra página, necesitamos que el árbol DOM se haya cargado completamente, para eso nos proporcionan una instrucción:
`$(document).ready(function() { //Instrucciones });`

Finalmente, tenemos que aclarar que Jquery es una librería de tantas otras que existen en Javascript y que es capaz de facilitarnos el desarrollo de un proyecto web. Se muestra un ejemplo de las facilidades que le da al desarrollador el uso de Jquery frente a Javascript.

Jquery:

```
$("#contenedor").append("<p>Titulo</p>");
```

Javascript:

```
document.getElementById("contenedor").innerHTML += "<p>Titulo</p>";
```

3.2.2. AJAX

AJAX (*Asynchronous JavaScript And XML*) es un conjunto de tecnologías que se unen de formas nuevas y sorprendentes, permitiendo la interactividad del cliente con el servidor en segundo plano mientras se mantiene una comunicación asíncrona. De esta forma, es posible realizar cambios en la página sin necesidad de recargarla, permitiendo al cliente una mejora en la interacción con el servidor.

Las aplicaciones construidas con esta tecnología eliminan la recarga constante de páginas en el servidor. Además, el usuario nunca tiene una ventana vacía y puede mirar otras zonas de

la web mientras recibimos la información que hemos pedido.

Una de las aplicaciones más importantes basadas en esta tecnología es Google Maps. Esta aplicación permite moverse por un mapa sólo con el movimiento del ratón. Cuando movemos el ratón en el mapa, detecta el evento y este manda una petición AJAX para recibir los frames y mostrarlos en la nueva posición del mapa.

AJAX no es una tecnología basada exclusivamente en Javascript, como hemos dicho anteriormente; es un conjunto de tecnologías entre las que se encuentran HTML, CSS, árbol DOM y XML. XML es el formato generalizado para la transferencia de datos, pero pueden utilizarse otros formatos como HTML preformateado, texto plano o el que se ha utilizado en este proyecto, JSON.

3.2.3. JSON

JSON (*JavaScript Object Notation*) es un formato para el intercambio de datos. Nació como una alternativa a XML, pero su fácil uso ha generado un gran número de seguidores.

Se basa en una estructura de datos del lenguaje de programación Javascript. Una gran ventaja de JSON es que puede ser leído por cualquier lenguaje de programación y parseado de una forma sencilla.

Este tipo de estructura de datos pueden asumir la forma de un objeto con clave y valor o un array. En este proyecto, en el lado del servidor se ha utilizado simplejson para el lenguaje de programación Python. Simplejson es un codificador/decodificador de JSON para Python, simple, rápido y extensible.

3.3. Jquery UI

JqueryUI [3] es una biblioteca de componentes creada a partir de la librería Jquery. Jquery añade interacción con la interfaz del usuario, widgets, efectos y temas. Al usuario le proporciona un diseño visual atractivo y una interacción intuitiva.

En la página oficial de JqueryUI existe una herramienta llamada Themeroller que permite diseñar los estilos online de una forma sencilla. De esta forma conseguimos que se adapte a nuestras necesidades.

La biblioteca se divide en distintos módulos:

- **Interacciones**:
  - **Draggable**: hace que el elemento pueda ser arrastrado.
  - **Droppable**: permite que el elemento responda a elementos que han sido arrastrados hacia él.
  - **Resizable**: permite redimensionar el elemento.
  - **Selectable**: permite seleccionar entre una lista de elementos.
  - **Sortable**: ordena una lista de elementos.
- **Widgets** *utilizados en el proyecto*:
  - **Accordion**: menú desplegable con efecto acordeón.
  - **Autocomplete**: permite a los usuarios autocompletar y seleccionar de una lista a medida que escribe, utilizando el filtrado.
  - **Dialog**: muestra el contenido en una superposición interactiva, también llamada ventana modal.
  - **Tooltip**: muestra información sobre herramientas que son útiles para rellenar formularios y para señalar contenido adicional de cada campo.
  - Existen otros tipos de widgets que no hemos utilizado como button, datepicker, menu, progressbar, selectmenu, slider, spinner y tabs.
- **Efectos**: aplica animaciones a los elementos seleccionados.

Es compatible con los navegadores Internet Explorer 6.0, Mozilla Firefox 3, Safari 3.1, Opera 9.6 y Google Chrome.

3.4. CSS3

CSS (*Cascading Style Sheets*) u hojas de estilo en cascada es un lenguaje utilizado para definir la presentación de documento HTML o XML. La hojas de estilo pueden ser utilizadas en un documento HTML añadiendo el parámetro style dentro de la etiqueta que queramos modificar. Otra forma de modificar el estilo de nuestra etiqueta es añadir a nuestro documento HTML, dentro del elemento `<head>`, la etiqueta `<style>`. Esto nos proporciona el beneficio de separar el estilo de la página del documento HTML, a este tipo se le llama hoja de estilo interna.

Conseguimos que los estilos sean reconocidos antes de que la página sea cargada completamente, pero este tipo de definición del estilo pierde las ventajas que ofrecen las hojas de estilo al mezclarse el contenido con la presentación.

Las hojas de estilo externas están vinculadas a un documento HTML a través del elemento `<link>`, el cual debe ser situado en la sección `<head>`. Con esto conseguimos separar completamente en archivos distintos las reglas de estilo del código HTML, proporcionando al desarrollador una mayor abstracción. Un documento CSS consta de un conjunto de reglas que se aplican al estilo de una página. La regla está formada por una declaración que indica que tenemos qué hacer y un selector que nos indica a quién debemos hacérselo. A un mismo selector se le pueden aplicar múltiples declaraciones y a su vez varios selectores pueden compartir las mismas declaraciones. Los selectores son nombrados según su nombre (*name*), identificador (*id*), clase (*class*), posición en el DOM...
La declaración se define entre llaves y tiene un formato `"{propiedad:valor;}"`.

CSS3 es la tercera generación de CSS y totalmente compatible con HTML5. Los nuevos selectores dan más flexibilidad a la hora de seleccionar unos u otros elementos. CSS3 está dividido en varios documentos llamados *"módulos"* y cada modulo añade nuevas funcionalidades a la anterior generación. Con esta última generación podemos conseguir cajas con bordes redondeados, sombras, animaciones, cualquier tipografía, colores gradientes y pseudo-clases para agregar estilo a elementos HTML.

El gran inconveniente de CSS es que está formado por muchos módulos que siguen desarrollándose y todos los navegadores no pueden dar soporte a algunas funciones de CSS.

3.5. Highcharts

Highcharts [4] es una biblioteca de gráficos escrita en Javascript, que ofrece de una forma sencilla agregar gráficas a un sitio web. Esta biblioteca está muy bien documentada y es posible utilizarla en todos los navegadores modernos. Para utilizar Highcharts no necesitamos licencia sino va a ser comercializado, además es un sistema de código abierto, por lo que puedes modificarlo a las necesidades de cada usuario.

Se basa únicamente en las tecnologías de navegación nativa y no requiere plugins secundarios. Además, no es necesario instalar nada, solo con descargarnos el núcleo `highcharts.js` podemos empezar a crear gráficas dinámicas. Para la creación de gráficas no necesitamos grandes conocimientos de programación, las opciones están estructuradas como objetos Javascript. Podemos crear gráficas en vivo con la tecnología AJAX, añadir múltiples ejes, etiquetas tooltip, exportación de gráficas a nuestro ordenador, zoom en gráficas con múltiples datos, rotación del texto para las etiquetas.

Con esta biblioteca podemos crear gráficas muy diversas, a continuación detallamos las que se han utilizado en el proyecto.

- **Splines**: el tipo de gráfico de spline es un gráfico de líneas que traza una curva ajustada a través de cada punto de datos de una serie, este tipo se ha utilizado para mostrar el modelo COCOMO.

- **Basic bar**: un gráfico de columnas es una forma de representar gráficamente una serie de datos, y está conformado por barras horizontales. Se ha utilizado para obtener los histogramas.

- **Pie chart**: el gráfico circular se utiliza para representar porcentajes y proporciones. Con este se ha representado el porcentaje de lenguaje de programación que se ha utilizado en cada versión de Debian.

3.6. Django y Python

3.6.1. Django

Django [1] es un framework web de código abierto escrito íntegramente en Python que permite construir aplicaciones web de forma rápida. Fue desarrollado inicialmente para gestionar páginas webs orientadas a noticias de World Online y en julio de 2005 se liberó bajo licencia BSD (Berkeley Software Distribution).

Se basa en la filosofía **Modelo-Vista-Controlador** (*MVC*), pero no sigue estrictamente el paradigma MVC, porque lo que se llamaría *Controlador* en los típicos frameworks MVC en este se llama *vista* y lo que se llamaría *vista* se llama *plantilla*. El objetivo principal de Django es la creación de sitios web complejos de una forma rápida, poniendo énfasis en el reúso, la conectividad y el principio *No te repitas* (**DRY**, *Don't Repeat Yourself*). La arquitectura del framework se puede dividir en cinco grandes bloques (tres bloques de proyecto y dos bloques de aplicación):

- **settings.py**: es el fichero de configuración formado por un conjunto de variables donde se configura la base de datos, aplicaciones instaladas, caché, zona horaria, codificación, directorio de plantillas...

- **urls.py**: usa expresiones regulares para asociar URLs de nuestro proyecto con las funciones que controlan la creación de nuestros templates, es decir, las vistas de cada aplicación.

- **manage.py**: es un wrapper del django-admin.py que se encarga de realizar varias tareas antes de delegar el comando al django-admin.py: pone el paquete del proyecto en el sys.path, establece la variable DJANGO_SETTINGS_MODULE de modo que apunte al settings.py de su proyecto y llama a django.setup() para inicializar algunas partes internas de Django. Con manage.py podemos interactuar desde la consola con el proyecto Django de varias formas con los comandos descritos en la documentación.

- **models.py**: aquí crearemos las clases que forman el modelo de nuestra aplicación. Cada aplicación tiene su models.py y contiene los campos básicos y el comportamiento de los datos que serán almacenados en código Python. Dentro crearemos una clase por cada ta-

bla del modelo y un campo indicando el tipo de dato por cada columna de la tabla. Django usa un modelo para ejecutar código SQL y retornar las estructuras convenientes en Python representando las filas de tus tablas de la base de datos. Con el comando `syncdb` podemos generar lo que sería la sentencia `CREATE TABLE` automáticamente, verificando la base de datos para ver si las tablas apropiadas ya existen, y creándolas en caso de no existir.

- **views.py**: es el código invocado por una URL o conjunto de URLs. Debe ser un método o un objeto. Este código son funciones de Python que se ejecutarán en el servidor, para satisfacer la petición de un cliente en forma de template, respuesta HTML, JSON, imagen... Las vistas tienen acceso a la información que pide el cliente para así actuar en consecuencia. Además, pueden distinguir entre el método **GET**[5] y **POST**[6].

Django's Cache Framework

El gran inconveniente de las aplicaciones web dinámicas son la excesiva carga de trabajo en el servidor, por toda la clase de cálculos y consultas. Esto es mucho más costoso que leer un fichero estático del sistema.

Para mejorar la eficiencia del sitio web se usa la caché que es un almacenamiento rápido, normalmente en la memoria RAM de nuestro servidor, evitando rehacer cálculos matemáticos muy pesados. Django contiene un framework de caché que abstrae las operaciones básicas sin que nosotros tengamos que preocuparnos por el motor de caché que usemos, ya que, por defecto, Django soporta Memcached[7], caché en base de datos, caché en el sistema de archivos y memoria local.

[5] obtener información del servidor

[6] enviar información al servidor

[7] sistema distribuido de propósito general para caché de objetos basado en memoria

3.6.2. Python

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 y cuyo nombre está inspirado en el grupo de cómicos ingleses *"Monty Python"*. Tiene una sintaxis muy limpia y que favorece un código legible. Se trata de un lenguaje interpretado o de script, dinámica y fuertemente tipado y multiplataforma. La ventaja de este tipo de lenguajes es que son más flexibles y portables, en cambio son más lentos que los lenguajes compilados. Un lenguaje compilado es traducido de un lenguaje fuente en uno hablado en lenguaje máquina empleando un compilador. Esto permite que al ejecutar el programa se guarde el código binario en la memoria del ordenador e inicie la ejecución desde la primera instrucción. Sin embargo, un lenguaje interpretado como Python cuando es ejecutado, convierte el código fuente en una forma intermediaria llamada bytecodes (*archivos .pyc*), después los traduce en lenguaje máquina y son ejecutadas las instrucciones. Esto permite que sea portable como hemos dicho anteriormente despreocupándote de enlazar, cargar librerías, etc.

Es un lenguaje multiparadigma debido a que soporta orientación a objetos, programación imperativa y programación funcional. Esta orientado tanto a procedimientos como a objetos, esto quiere decir que está construido sobre funciones, las cuales son piezas de programa que pueden reutilizarse, y sobre objetos, los cuales combinan datos y funcionalidades.

Es un lenguaje con una sintaxis muy sencilla y es muy aconsejable para iniciarse en la programación, aunque esta sintaxis es muy estricta:

- Se diferencia entre mayúsculas y minúsculas.
- No tenemos llaves para encerrar bloques, lo que indica el comienzo y el final de un bloque es el nivel de sangrado.
- La declaración de variables es implícita, es decir las variables aparecen cuando se les asigna un valor y desaparecen cuando se sale de su ámbito.
- Una sentencia termina cuando acaba la linea, salvo en expresiones entre paracentesis, corchetes o llaves.

3.7. Apache

Apache [5] es un servidor web HTTP de código abierto multiplataforma, ya que puede ser utilizado en Windows, UNIX, Macintosh y otros que utilicen el protocolo HTTP1.1 y maquinas virtuales. Apache es uno de los servidores más usados debido a que es fácil conseguirlo y proporciona mucha documentación.

¿Por qué utilizamos Apache? Django nos proporciona un servidor para probar nuestra aplicación, pero si queremos llevar la aplicación a producción necesitamos un servidor que pueda servir grandes cantidades de tráfico.

En el proyecto se ha utilizado Apache con mod_wsgi, debido a que es actualmente la configuración más aconsejada para usar Django en un servidor en producción. `mod_wsgi` es un módulo de Apache que puede albergar cualquier aplicación Python WSGI. Puede funcionar de dos modos, modo demonio o en nuestro caso el modo incrustado. En el modo incrustado el código permanece en memoria a lo largo de la vida del proceso Apache, lo que repercute en aumentos significativos de desempeño comparado con otros arreglos del servidor.

Capítulo 4

Implementación y desarrollo

4.1. Crear e Importar base de datos

El primer paso para implementar el proyecto fue instalar MySQL en el ordenador, para ello necesitamos instalar los paquetes mysql-server y mysql-client. Una vez instalados los paquetes tenemos que crear la contraseña del administrador y podemos conectar con el servidor MySQL. Ahora, necesitamos importar las bases de datos que se han proporcionado. Tenemos archivos con formato `".sql"` y estos tienen los mismos nombres en sus tablas (*"files"* y *"packages"*). Si importáramos todos los archivos sin tener en cuenta ese problema no podríamos diferenciar entre las distintas versiones, por eso lo que hemos hecho es lo siguiente:

- Creamos la base de datos: `create database Nombrebasededatos;`
- Entramos en la base de datos: `use Nombrebasededatos;`

  Los siguientes pasos se repiten con todos los archivos formato `".sql"`.
- Cargamos el archivo en la base de datos: `source path/nombre_archivo.sql;`
- Renombramos las tablas files y packages con los nombres de las versiones:
  `rename table packages to packagesnombreversion;`
  `rename table files to filesnombreversion;`

4.2. Creación y estructura del proyecto Django

La creación de un proyecto Django es sencilla, una vez instalados Python y Django tenemos que ejecutar el comando:

```
django-admin.py startproject debiancount
```

Se ha creado el directorio debiancount, ahora se debe crear la aplicación desde ese directorio con el comando:

```
python manage.py startapp debianapp
```

Con estos comandos se han creado los siguientes archivos:

- **__init__.py**: un fichero vacío que debe ser considerado como un paquete Python.
- **manage.py**: herramienta para interactuar con el proyecto Django.
- **settings.py**: configuración del proyecto.
- **urls.py**: URLs de las aplicaciones del proyecto.
- **models.py**: definición de las clases del modelo de datos.
- **views.py**: se encarga de invocar el código para cada recurso, mostrando la página al usuario.
- **templates**: en este directorio creamos los ficheros de texto que pueden ser generados en cualquier formato de texto (HTML, XML, CSV[1]...)
- **static**: en este directorio se guardan los ficheros estáticos (imágenes, código Javascript, librerías, CSS, etc).

[1] comma-separated values: documento para representar datos en forma de tabla

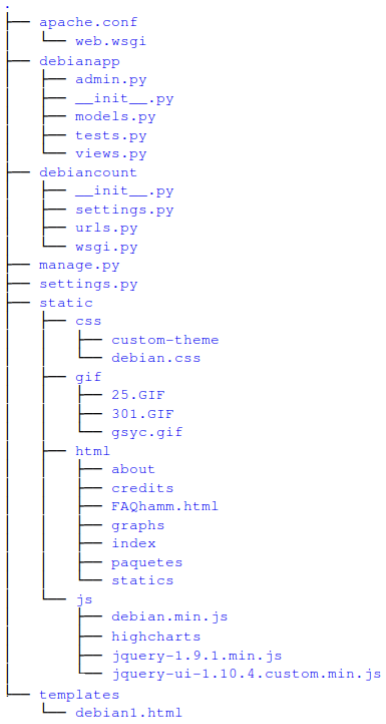

Figura 4.1: Árbol con ficheros Javascript comprimidos.

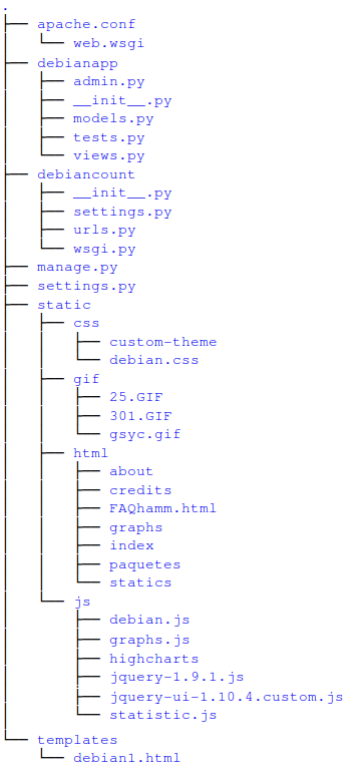

Figura 4.2: Árbol con ficheros Javascript sin comprimir.

Se han creado dos capturas del árbol de ficheros de la aplicación, en la Figura 4.2 tenemos los archivos Javascript sin pasar por el proceso de compresión. En la Figura 4.1 tenemos los archivos Javascript acabados en `.min.js`, esto significa que los archivos han sido comprimidos y por lo tanto el tiempo de respuesta al cliente sera menor. En el caso de los ficheros *jquery-ui-1.10.4.custom.min.js* y *jquery-1.9.1.min.js* pueden ser descargados directamente comprimidos desde sus respectivas páginas web. En cambio, los ficheros que han sido programados se han comprimido desde `http://jscompress.com/`, los tres ficheros se han comprimido en un solo fichero *debian.min.js*. Se ha comprobado con el complemento Firebug que la aplicación tiene un tiempo de descarga con los ficheros `"min"` de 125 ms, mientras que los ficheros originales tienen un tiempo de descarga de 691 ms.

4.2.1. Interacción Django, Model-Template-View

La creación de un proyecto Django nos mostrará un mensaje en la web *It worked!*, no obstante si se quiere trabajar de forma dinámica necesitamos modificar los archivos anteriormente nombrados en este apartado. En el apartado 3.6.1 se ha descrito cada archivo detalladamente, pero no la interacción entre ellos para el perfecto funcionamiento del framework.
Cuando arrancamos el servidor -*Se puede arrancar el servidor Python con la línea de comandos:* `python manage.py runserver` *para un entorno de pruebas o como detallaremos más adelante, con un servidor Apache para un entorno de producción*- se importa el archivo *settings.py*.

El archivo settings.py se usa para configurar todas las opciones que nos permite Django, sin embargo la que nos interesa es `ROOT_URLCONF`. Esta variable informa del módulo que debe utilizar para esa aplicación web. Si llega una petición web se carga `URLCONF` apuntada por la variable `ROOT_URLCONF`. `URLCONF` realiza un mapeo entre los patrones del archivo *urls.py* (`URLCONF` permite expresiones regulares, ya que son una forma compacta de especificar patrones en un texto). Cuando encuentra una coincidencia llama a la función vista asociada, pasando un objeto `HttpRequest` como primer parámetro de la función. Si no encuentra el patrón Django tiene un middleware que lanza una excepción como respuesta.

La petición se ha enlazado con una función programada en la vista. Esta función dará una respuesta al cliente, pero si necesitamos obtener información de una base de datos tenemos que importar los modelos que se usaran en la aplicación. Una vez creados los modelos como se detalla en el apartado 4.3, Django provee automáticamente una API en Python de alto nivel para trabajar con estos. Como en el caso de la `URLCONF` la base de datos se enlaza con el framework en el archivo de configuración *settings.py*, de esta forma el modelo validara el acceso a la base de datos que ha sido asociada. En el caso de tener **Django's Cache Framework** no llegara acceder al modelo, ya que la caché nos facilitara la información. Finalmente, la función vista es responsable de retornar el objeto `HttpResponse`, aunque este archivo tiene un *View Middleware* que se encarga de las excepciones lanzadas por la vista.

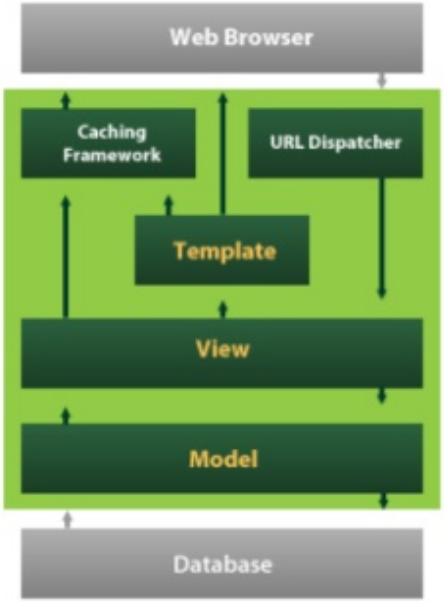

Figura 4.3: Procesamiento de una petición Django. (Fuente: `http://ch-blog.s3.amazonaws.com/2014/01/Django_mvc.png`)

4.3. Modelos de Django

Todo proyecto Django tiene un fichero models.py en el cual se definen la clases del modelo de datos. En el fichero settings.py se debe conectar el proyecto Django con la base de datos MySQL que hemos creado. En este caso se tenían las bases de datos importadas en MySQL, por lo tanto con el siguiente comando se nos crearán las clases de forma automática: `python manage.py inspectdb >path/models.py`
Tenemos 7 versiones de Debian y estas formarán las clases diferenciando por las tablas de tipo Files y Packages.

```
class Fileshamm(models.Model):
    id = models.BigIntegerField(primary_key=True)
    file = models.CharField(max_length=255)
    language = models.CharField(max_length=16)
    sloc = models.IntegerField()
    class Meta:
        managed = False
        db_table = 'fileshamm'
```

Clase **Files**: esta clase distingue entre ficheros por cada paquete (tendrá un nombre distinto según la versión: fileshamm,fileslenny,filessarge...).

- **id**: identificador del fichero (se puede tener el mismo identificador para distintos nombres de ficheros).
- **file**: nombre del fichero.
- **language**: lenguaje de programación con el que se ha desarrollado el fichero.
- **sloc**: número de lineas de código del fichero.

```
class Packageshamm(models.Model):
    id = models.BigIntegerField(primary_key=True)
    name = models.CharField(max_length=64)
    version = models.CharField(max_length=32)
    slocs = models.BigIntegerField()
    files = models.IntegerField()
    class Meta:
        managed = False
        db_table = 'packageshamm'
```

Clase **Packages**: esta clase distingue entre los paquetes (tendrá un nombre distinto según la versión: packageshamm, packageslenny, packagessarge...).

- **id**: identificador del paquete.
- **name**: nombre del paquete.
- **version**: versión del paquete.
- **sloc**: líneas de código del paquete.
- **files**: número de ficheros que tiene el paquete.

Ahora que tenemos todas las clases formadas podemos realizar consultas a la base de datos con la API que nos proporciona Django.

4.4. Llamadas AJAX

En este proyecto uno de los cambios más importantes respecto a la antigua web, es realizar todas las consultas al servidor con llamadas AJAX. En la web tenemos dos tipos de contenidos; aquellos que están en la base de datos y que se obtienen de forma dinámica y los contenidos en formato `".html"` que son estáticos y están almacenados en la carpeta `/static/html` del servidor. Por lo tanto, en el proyecto se realizaran dos tipos de llamadas AJAX:

Llamadas AJAX a ficheros estáticos:

En el servidor tenemos una carpeta llamada `"static"` en esta carpeta se almacenan todos los ficheros estáticos del programa. Aquí se ha creado un directorio llamado `"html"` en el se guardan los ficheros `".html"` con la respuesta de las secciones index, about, FAQ y credits de cada versión Debian de la web.

Para mostrar el contenido de los ficheros estáticos `".html"` con llamadas AJAX desde Javascript, solo tenemos que usar el procedimiento .load que nos ofrece Jquery.

```
$( "identificador" ).load( "ruta del fichero" );
```

Identificador en el caso de esta web es `"#Principal"` que es nuestra página principal como se mostró en la figura que muestra la estructura de la web. Ruta del fichero, como hemos dicho anteriormente nuestro ficheros están en `/static/html`, además se han separado los ficheros HTML por sección. Por lo tanto la ruta para el fichero HTML de la sección about de la versión Hamm sería `/estatico/html/about/abouthamm.html`.

Además en el fichero urls.py se añade la siguiente linea para surtir al cliente los ficheros estáticos:

```
url(r'^estatico/(?P<path>.*)$', 'django.views.static.serve',
{'document_root':'/home/rober/Descargas/debiancount/static'}),
```

En cambio, mostrar el contenido de forma dinámica con llamadas AJAX, requiere más elaboración debido a la arquitectura MVC. Desde el cliente ejecutamos el procedimiento AJAX que nos proporciona Jquery:

```
var request = $.ajax({
    type:'GET',
    data:{"namepack":namepack},
    beforeSend: function(){
        $("#Principal").html("")
        $("#Principal").addClass("loading");
    },
    success: function(json) {
        paquetes(json,name[1],"name");
    },
    complete: function() {
        $("#Principal").removeClass("loading");
    },
    url: "/paquetes"
});
```

- **type**: El tipo de llamada que se realiza será GET, ya que pedimos información.
- **data**: Son los datos que le damos al servidor para que sepa lo que queremos recibir. En este caso le damos la sección que queremos y la versión en la variable namepack.
- **url**: Es la dirección donde el servidor tendrá preparada la función que recogerá la llamada AJAX.
- **success**: Recibe la información en la variable json y luego se utiliza una función para mostrar la información.
- **beforeSend**: En este caso mostramos un gif de carga para mostrar al cliente que se esta procesando su petición.
- **complete**: Cuando termina de procesarse la petición el gif se elimina.

El servidor recibe la petición en la url que se ha decidido, en este caso, `/paquetes`. Por lo tanto, en el fichero *urls.py* se añade la siguiente línea:

```
url(r'^paquetes$', 'debianapp.views.paquetes')
```

Ahora que tenemos enlazada la dirección en la que se pide la respuesta con la función que vamos a utilizar en el servidor. En el fichero views.py se tiene que crear una función llamada paquetes que buscara la información que nos pide el cliente y se la enviara.

```
if request.method == u'GET':
    namepack = request.GET[u'namepack']
    namepack = namepack.split('_')
    listaobj = busquedaobjs(Fileshamm,Packageshamm)
    listapacks = tablaspackages(Packageshamm, listaobj[1])
    json = simplejson.dumps(listapacks)

    return HttpResponse(json, mimetype='application/json')
```

En el código se mira si se ha realizado una llamada de tipo GET. Como es así, el siguiente paso es extraer la información que nos manda el cliente. Sacamos la información y llamamos a la función que nos busca la información (listaobj y listapacks). Cuando la búsqueda se ha realizado se convierte la lista de información a formato JSON y se envía con HttpResponse. El cliente recibe la respuesta en formato JSON y como se ha dicho anteriormente con la función success se puede mostrar la información de la forma que quiera el desarrollador.

4.5. Arquitectura General de la Aplicación

El funcionamiento de la aplicación se basa en la interacción entre las diferentes tecnologías presentadas en el capitulo 3, por lo que en la siguiente sección detallamos la arquitectura de la aplicación. El framework Django es la herramienta principal del proyecto, ya que es el elemento de enlace entre las diferentes tecnologías.

El primer paso es tener un servidor que reciba las peticiones, en este proyecto se ha elegido Apache. Para servir el proyecto se necesita el modulo WSGI de Apache, con el podemos alojar y ejecutar el proyecto Django. Se debe configurar el fichero WSGI con el siguiente código para alojar el proyecto:

```
import os, sys
sys.path.append('PATH del proyecto')
os.environ['DJANGO_SETTINGS_MODULE'] = 'debiancount.settings'
import django.core.handlers.wsgi
_application = django.core.handlers.wsgi.WSGIHandler()
```

```
def application(environ, start_response):
    environ['PATH_INFO'] = environ['SCRIPT_NAME'] + environ['PATH_INFO']
    if environ['wsgi.url_scheme'] == 'https':
        environ['HTTPS'] = 'on'
    return _application(environ, start_response)
```

Esta variable es la que corresponde al proyecto: `os.environ['DJANGO_SETTINGS_MODULE']`

Además, se debe configurar el fichero del servidor Apache, en ese fichero se configura el número de procesos, hilos y los directorios que queremos servir en este proyecto.

Una vez el servidor esta en marcha, Django necesita ser programado para recibir las diferentes peticiones, para ello se utiliza el lenguaje de programación Python. La petición se recibe según la dirección que hayamos introducido en el fichero urls.py y se enlaza con una función que será programada en el fichero views.py. Si la petición necesita tener información de la base de datos, primero tenemos que crear e importar la base de datos como se indica en el apartado 4.1. Tenemos las bases de datos importadas en MySQL, pero no en Django. En el fichero de configuración de Django debemos rellenar el siguiente campo para tener enlazado el proyecto con MySQL.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'Nombre base de datos',
        'USER': 'Nombre de usuario en MySQL',
        'PASSWORD': 'Contraseña de MySQL',
        'HOST': '',
        'PORT': '',
    }
}
```

En este momento realizamos una llamada desde la línea de comandos a inspectdb, este nos inspecciona la base de datos y nos crea los modelos que se han detallado en el apartado 4.3.

Hasta este momento, hemos lanzado el servidor Apache y se ha conectado con la base de datos para proporcionarnos la información. Como este proyecto es un poco especial, ya que se quieren proporcionar las páginas web con AJAX, como se detalla en el apartado 4.4 las peticiones se realizan desde el cliente con la tecnología AJAX y el servidor responde con la información en formato JSON.

Para interactuar con AJAX necesitamos tener programados los ficheros Javascript que interactuaran con el cliente, proporcionándole más dinamismo a la web.

Cada sección que describiremos en el apartado 4.8 tiene un fichero Javascript que interactuará con Django para proporcionarnos las diferentes secciones web.

- El fichero *debian.js* nos proporciona todas las páginas HTML estáticas que ya nombramos en el apartado 4.4 y la interacción de la sección packages. En este fichero también se crea la interfaz web con la librería JqueryUI y el buscador de paquetes.

- El fichero *statistics.js* nos ofrece la sección statistics.

- El fichero *graphs.js* esta programado para ofrecernos las gráficas que se obtienen con la biblioteca Highcharts.

En todos los ficheros Javascript nombrados anteriormente, se utiliza la biblioteca Jquery que proporciona al desarrollador; simplificar el código, realizar las llamadas AJAX y añadir efectos.

Todo lo anterior se muestra en una plantilla HTML localizada en la carpeta `/templates` del proyecto Django y para que visualmente sea atractiva para el usuario, se debe modificar con las hojas de estilo *(CSS3)*.

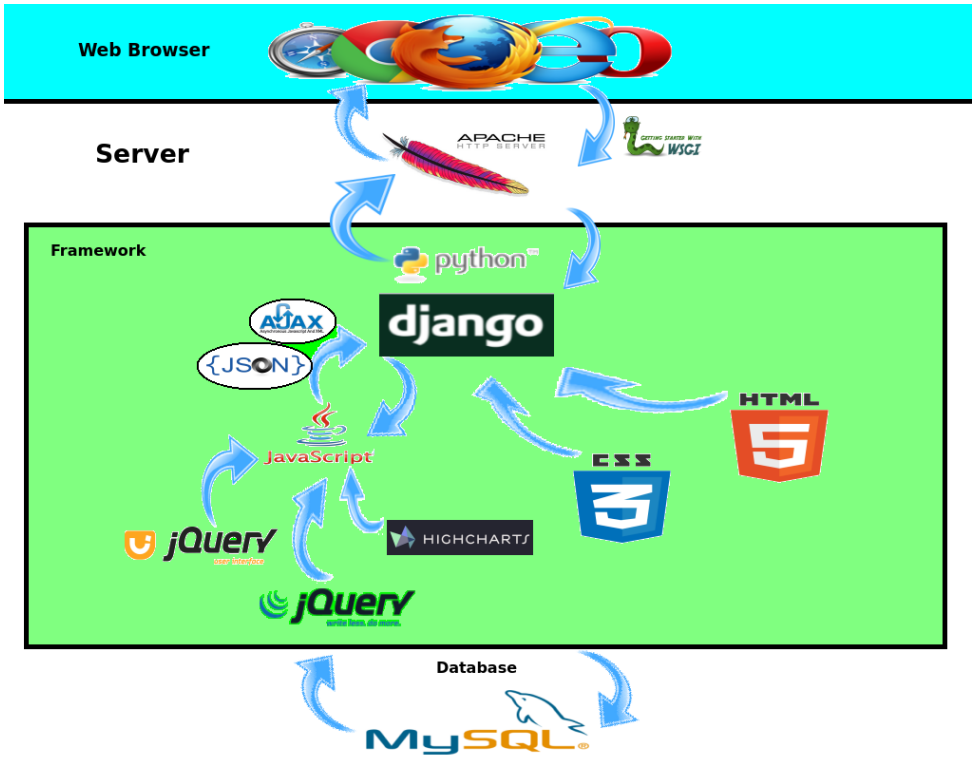

Figura 4.4: Arquitectura general de la aplicación. (Elaboración propia)

4.6. Diseño de la página

En todos los proyectos desarrollados en Django tenemos una página web principal y luego podemos movernos en otras direcciones por debajo de la web principal cambiando la plantilla. En cambio, en este proyecto se ha decidido que todo se mueva bajo la web principal con llamadas AJAX. La web tendrá la siguiente estructura:

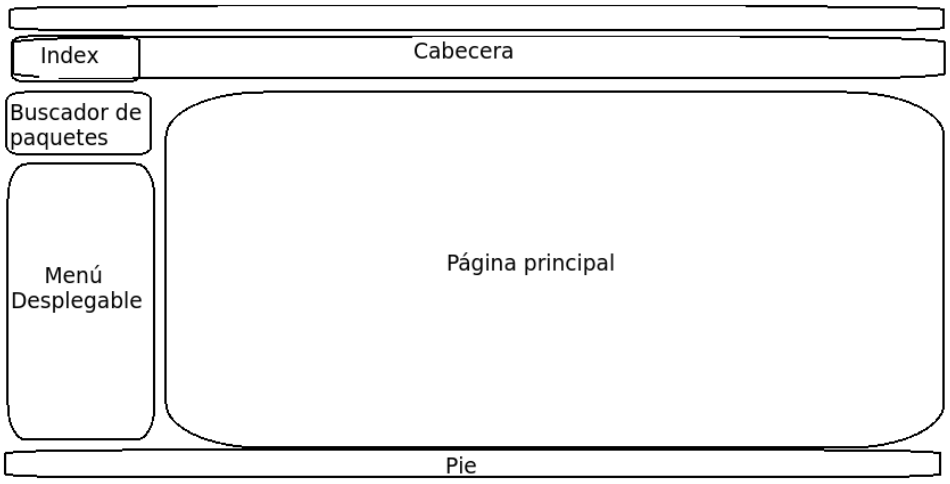

Figura 4.5: Estructura de la web. (Elaboración propia)

La cabecera cambia su nombre dependiendo de la versión de Debian en la que hemos clickado en el menú desplegable. El menú se despliega según la versión de Debian, cada versión tiene diferentes secciones que se podrán mostrar cuando clickemos sobre ella. Las secciones que se pueden mostrar son:

- **Index**: muestra una introducción a la página.
- **About**: permite profundizar en los objetivos de SLOCCount.
- **Statistics**: muestra las estadísticas macroeconómicas e incluye una lista ordenada de los lenguajes de programación que se utilizan y un modelo COCOMO.
- **Packages**: contiene una lista de todos los paquetes y sus estadísticas. Los paquetes pueden ser seleccionados para ver los ficheros diferenciados por el lenguaje de programación, las estadísticas y modelos COCOMO.
- **Graphs**: muestra figuras que pueden ayudar a entender la naturaleza del proyecto.

- **FAQ**: aquí se encuentra respuestas a preguntas sobre el modelo COCOMO, formulas y variables.
- **Credits**: podemos obtener información de las personas vinculadas con el proyecto.

En la página principal se mostrarán las diferentes secciones detalladas anteriormente. El buscador de paquetes nos mostrará información en una ventana modal que se desplegará. La caja Index nos vuelve a mostrar la página de inicio en el contenedor Página principal. Esta caja tiene posición fixed, por lo tanto siempre estará visible para el usuario.

4.7. API QuerySet Django

La API de la base de datos de Django es el complemento a la API de los modelos definidos en el apartado 4.3. Una vez definidos los modelos, se usara la API para acceder a la base de datos.

En este apartado, se hará referencia a los modelos usados en el proyecto:

El primer paso es importar los modelos para tener nuestros modelos visibles en las vistas y que estos puedan ser referenciados.

```
from models import *
```

Ahora que tenemos los modelos importados pueden ser usados para acceder a la base de datos desde las vistas.

La forma más simple de recuperar objetos de una tabla es obtenerlos todos. Para ello utilizamos el método `all()`.

```
allobjsF = Files.objects.all()
```

Obtenemos todos los objetos del modelo Files.

El mayor problema que encontramos con este método es que para bases de datos extensas, como las que se tienen en este proyecto, hay que tener mucho cuidado porque aumentaría el tiempo de ejecución. Lo mejor para este tipo de base de datos si queremos obtener todos los objetos es realizar una única búsqueda y guardarla en una variable por si la necesitamos en otra acción.

Sin embargo, normalmente lo que necesitamos es un conjunto de objetos del total. Para eso, hacemos uso de `filter()`.

```
objsleng = fich.objects.filter(language=lenguaje)
```

Filtra por el tipo de lenguaje en ese fichero.

Para obtener el número de objetos de un fichero, necesitamos utilizar el método `count()`, realiza lo que seria en MySQL `SELECT COUNT(*)`, de este modo obtenemos el número de objetos sin tener que cargar todos los registros en objetos Python y luego usar el método `len()`.

```
totalfiles = objs.count()
```

Cuenta los objetos de la variable objs.

Si lo que se quiere obtener es algo concreto, lo que hay que utilizar es el método `get()`, con este obtenemos el objeto que queramos según los parámetros que se introduzcan.

```
Packages.objects.get(name=nombre)
```

Obtenemos el objeto, buscándolo por el parámetro nombre.

En el desarrollo del proyecto se tenia la necesidad de obtener la suma de todos los valores de una lista, lo primero ocurrencia que se puede tener es sumar todos los objetos con un bucle for, pero con el método `aggregate(Sum())` proporcionado por Django disminuimos el tiempo de ejecución de la suma.

```
fich.objects.aggregate(Sum('sloc'))
```

Obtenemos la suma de todos los *SLOCs* de la lista de objetos *fich*.

Para diferenciar el tipo de lenguajes de cada versión de Debian, lo que se puede pensar en un primer momento es recorrer la lista de objetos con todos los lenguajes y después eliminar los duplicados. Aunque, la API nos proporciona `values_list()` con el que obtenemos los valores según el key[2] que introducimos.

El siguiente código nos da una lista de los diferentes lenguajes:

```
lenguajes = fich.objects.values_list('language',flat=True)
lenguajes = list(set(lenguajes))
```

[2]clave que se usa en la base de datos

4.8. Secciones de la web

En todas las aplicaciones de un proyecto Django tenemos un fichero llamado views.py, en el que el servidor va a recibir peticiones web y este las contestará en formato de texto. El fichero views.py enlaza sus funciones con las direcciones del fichero urls.py. En este proyecto las urls se enlazan a llamadas AJAX que serán recibidas en las funciones de las vistas y estas las procesarán enviando la información en formato JSON. Cuando el cliente reciba la información en formato JSON la parseará y mostrará en la web. En los siguientes apartados obtenemos con más detalle una visión de cada sección.

4.8.1. Statistic

En esta sección se detallan las macroestadísticas de la versión Debian que hallamos seleccionado, tiene tres subsecciones macroestadísticas, estadísticas por lenguaje de programación y modelo COCOMO. En un primer momento se pensó diseñarla de tal forma que mandabas una petición AJAX cada vez que clickabas en una de las subsecciones, pero finalmente se optó por pedir las tres peticiones AJAX de las subsecciones a la vez cuando clickamos en la sección. ¿Por qué se ha decidido hacerlo de esta forma? Una buena forma de explicarlo es con una metáfora, cuando pedimos el menú en un restaurante el camarero nos toma nota del primer plato, el segundo plato y el postre. No tenemos que llamar al camarero por cada plato que pedimos, ya que demoraría demasiado el servicio.
En la web se utiliza la misma fórmula, se piden las tres subsecciones a la vez y de esta forma podemos visualizar las que se vayan cargando. Por lo tanto, cuando se clicka en la sección statistic mandamos tres peticiones AJAX, el servidor las recibe, las procesa y envía la información en formato JSON. El cliente recibe la información, la prepara en el formato deseado con Javascript (en este caso son tablas) y la muestra en el navegador.

Macroestadísticas muestra los siguientes datos

- **Número de paquetes.**
- **Número de ficheros.**

- **Número de SLOC.**
- **Media del número de SLOC/ficheros.**
- **Media del número de ficheros/paquetes.**
- **Media del número de SLOC/paquetes.**

Para obtener estos resultados la API de Django nos proporciona filtros y sumatorios, que nos hacen los cálculos más rápidos y sencillos. La siguiente línea de código nos sumaría todos los SLOC: `numtotalsloc=fich.objects.aggregate(Sum('sloc'));`

Estadísticas por lenguaje de programación (muestra una tabla con 9 columnas)

- **Índice**
- **Lenguaje de programación**
- **Número de SLOC**
- **Porcentaje de SLOC**
- **Número de paquetes**
- **Porcentaje de paquetes**
- **Número de ficheros**
- **Porcentaje de ficheros**
- **Número de SLOC/ficheros**

En la última fila se realiza la suma de todas las filas obteniendo los resultados totales de las columnas anteriormente mencionadas.

Modelo COCOMO

Muestra una tabla con el número de líneas de código de la versión, el esfuerzo estimado personas-mes, el tiempo estimado en realizar la versión tanto en meses como en años, el número de desarrolladores que son necesarios para realizarla y el coste total del desarrollo. Para realizar los cálculos hemos utilizado el modelo básico de COCOMO, a continuación se detallan las fórmulas.

- **Esfuerzo (Personas-Mes)** = $(2,4 * KSLOC^{1,05})$
- **Tiempo de desarrollo (Meses)** = $(2,5 * KSLOC^{0,38})$
- **Personas** = $(\frac{Esfuerzo}{Tiempodedesarrollo})$
- **Coste total** = $((Salariomedio) * (Personas - mes) * (Tiempoestimado))$

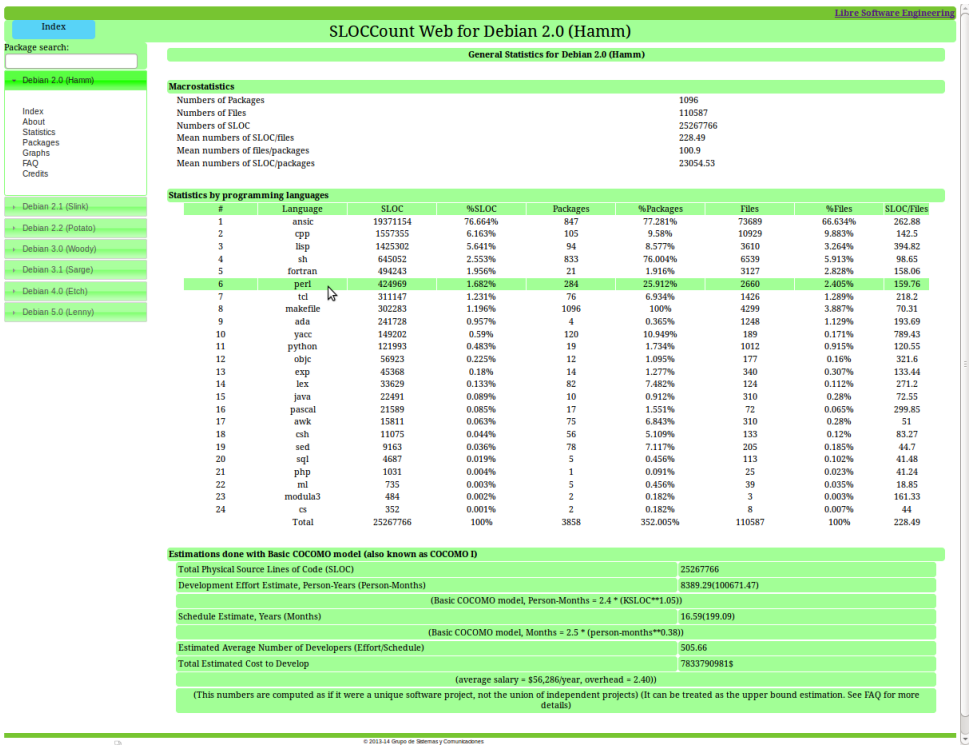

Figura 4.6: Captura de la sección statistics (versión Hamm).

Una vez comprendida la estructura de esta sección, se va a proceder a la explicación del código para llegar a obtener esa estructura.

Cuando pinchamos en *Statistics* de cualquiera de las versiones Debian, tenemos un evento Jquery.

Lo primero que se realiza es un cambio en la cabecera de la web:

`$(".cabecera").html("SLOCCount Web for Debian 2.0 (Hamm)");`

El segundo paso es cargar un HTML con la estructura de la pagina principal de la web:

`$( "#Principal").load(/estatico/html/statics/staticshamm.html")`

El tercer paso es guardar el identificador con el cual sabremos en que sección de la web estamos `id="statics_hamm"`.

Finalmente, realizamos las tres llamadas AJAX nombradas anteriormente. Cuando el servidor recibe la petición comprueba que es una llamada de tipo GET, separa la variable (el identificador nombrado anteriormente) que recibe en una lista (el primer elemento de la lista es la sección y la segunda la versión de Debian) y comprobamos si la información esta en caché. Con el segundo elemento de la lista (la versión de Debian) comprobamos de cual modelo de la base de datos debemos sacar la información. Cuando tenemos toda la información la introducimos en cache y respondemos al cliente encapsulando la lista en formato JSON.

```
if request.method == u'GET':
    namepack = request.GET[u'namepack']
    namepack = namepack.split('_')
    valor = cache.get('stat_' + namepack[1])
    if (not valor):
        if (namepack[1] == "hamm"):

            \\AQUI VAN LAS FUNCIONES QUE NECESITAMOS DEPENDIENDO DE LA LLAMADA AJAX.

    \\ REALIZAMOS UN IF POR CADA VERSION DEBIAN.

        cache.set('stat_' + namepack[1], listastatic,None)
    else:
        listastatic = valor

    json = simplejson.dumps(listastatic[0])

    return HttpResponse(json, mimetype='application/json')
```

Cuando el cliente recibe la información, llama a una función que crea la tabla. Primero se crea la cabecera de la tabla y luego recorremos la lista proporcionada por el servidor introduciendo los datos con el formato tabla HTML. La suma de valores de cada columna de la tabla

se realiza en Javascript, por lo tanto cuando se recorre la lista se suman los valores y cuando termina el bucle, se crea la ultima fila de la tabla con el sumatorio. De esta forma conseguimos la sección Statistics, ya que las secciones Macrostatistics y COCOMO no necesitan un sumatorio de los datos, pero si se realiza una tabla para mostrar los datos. Cuando la tabla esta creada en una variable, se muestra en la web con la siguiente función: `$(ID).html(TABLA);`

4.8.2. Packages

Esta sección está dedicada a las estadísticas detalladas por el paquete de la versión que se halla seleccionado. Aquí encontraremos una tabla en la que nos encontraremos:

- **Índice**
- **Nombre del paquete**
- **Versión del paquete**
- **Número de lineas de código en ese paquete**
- **Número de ficheros que contiene el paquete**
- **Número de líneas de código fraccionado por el número de ficheros**
- **Número de personas necesarias para desarrollar el paquete cada mes**
- **Tiempo estimado que se necesita para desarrollar el paquete en meses**
- **Promedio del número de desarrolladores en un mes**
- **El coste total del paquete**

Al final de la tabla se muestra el número total de todas las columnas mencionadas anteriormente. Si se posiciona el ratón en una de las filas de la tabla, esa fila se resalta en otro color, de esta forma el usuario tiene un posicionamiento claro de la fila que observa. Como en la sección anterior, cuando el cliente clicka en la sección packages se envía una petición AJAX al servidor, este realiza los cálculos y los envía en formato JSON. Cuando el cliente los recibe los muestra en el formato tabla, la diferencia con la sección anterior es que en esta solo se envía una petición AJAX y las funciones serán similares al apartado Statistics, concretamente la sección estadísticas por lenguaje de programación.

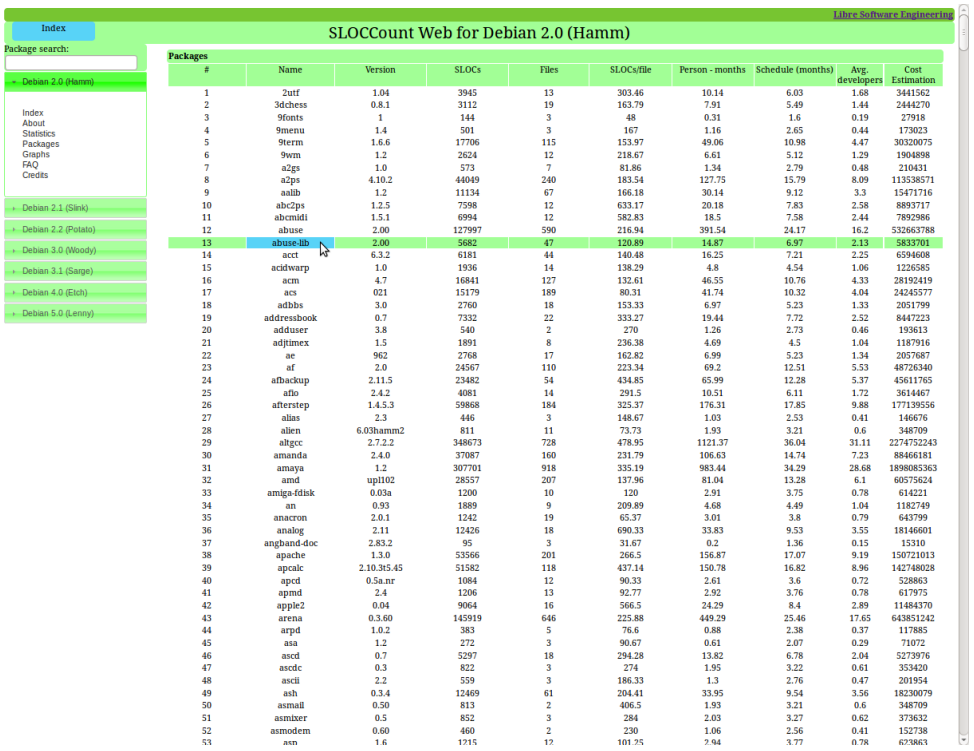

Figura 4.7: Captura de la sección packages (versión Hamm).

La tabla mostrada puede ser ordenada por Nombre, Versión, SLOCs o Número de ficheros. Cuando ordenamos la tabla no se envía una nueva petición AJAX, ya que se ha guardado en una variable la respuesta del servidor y con esta ordenamos en el cliente, gracias a Javascript.

Además, el nombre del paquete y el número de ficheros pueden ser seleccionados y nos mostrara información sobre ellos. Cuando se selecciona el nombre del paquete se despliega una ventana modal que nos muestra una tabla con:

- Los lenguajes usados en ese paquete.
- Número de líneas de código de cada lenguaje.
- El porcentaje de SLOCs.
- Número de ficheros de cada lenguaje de programación.
- Porcentaje de ficheros por lenguaje.
- SLOCs fraccionado por el número de ficheros.

La tabla se puede ordenar por lenguaje de programación, número de líneas de código o número de ficheros. También se muestra una tabla COCOMO para ese paquete como la mostrada en la sección macrostatistics. Con la ventana modal conseguimos una página superposicionada que nos proporciona flexibilidad, ya que tenemos nuestra tabla debajo para seguir buscando en ella lo que nos interesa, sin tener que volver a llamar al servidor para recibirla de nuevo. De esta forma el servidor recibe menos peticiones. En la Figura 4.8 se muestra una captura de la ventana modal cuando seleccionamos el nombre de uno de los paquetes.

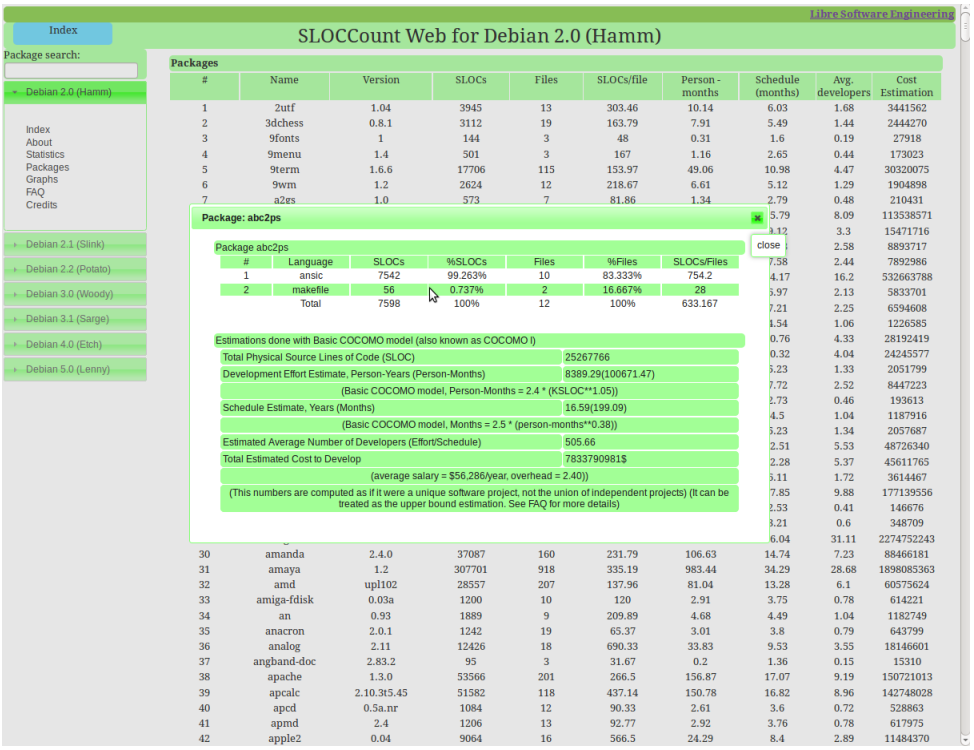

Figura 4.8: Captura de la sección packages después de presionar el nombre de un paquete (versión Hamm).

Por otro lado, podemos seleccionar el número de ficheros de un paquete, de esta forma se mostrará una ventana modal con una tabla que contiene:

- Índice.
- Nombre de los ficheros que contiene el paquete seleccionado.
- Lenguaje de programación utilizado en cada fichero.
- Número de líneas de código de cada fichero.

Al final de la tabla se muestra el número total de líneas de código usadas en ese fichero. Esta tabla también puede ser ordenada por nombre de fichero, lenguaje de programación o número de líneas de código. En la Figura 4.9 se muestra una captura de la ventana.

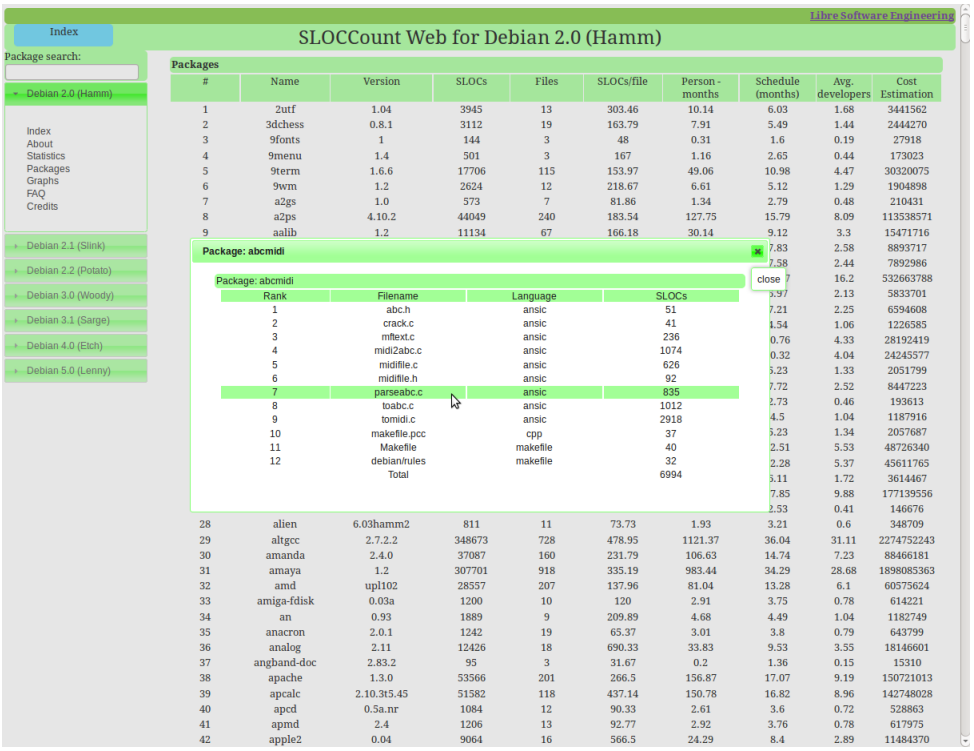

Figura 4.9: Captura de la sección packages después de presionar el número de ficheros de un paquete (versión Hamm).

4.8.3. Graphs

En este apartado de la web se muestran gráficas que representan las estadísticas de la versión de Debian que hallamos seleccionado. Para esta sección de la web se ha utilizado el selector hover de CSS, que nos resaltará la letra y cambiará el cursor cuando posicionemos el ratón en alguno de los nombres de las gráficas que queremos obtener. Cuando clickemos en uno de los nombres se desplegará la gráfica en la parte superior y se modificará si pulsamos en cualquier otro. Todas las gráficas tienen indicados sus respectivos ejes y el título, además al posicionar el cursor encima de alguna gráfica nos mostrará datos detallados. Como se explica en el apartado 3.5, las gráficas se han realizado con la biblioteca Highcharts y los módulos que se han utilizado son basic bar para mostrar los histogramas, pie chart y spline que muestra las estadísticas del modelo COCOMO en rectas.

Debido a que la biblioteca Highcharts no nos proporciona directamente una forma de calcular histogramas, se ha tenido que ingeniar la forma de realizarlos. La biblioteca nos ofrece gráficas de barras con el módulo basic bar, pero estas se obtienen introduciendo los datos en formato de lista en los ejes x e y. Por lo tanto, la forma de realizar histogramas es calcular con anterioridad sumatorios, para la obtención de dichos histogramas.

Lo primero que realizamos es una petición al servidor de los datos y estos se mandan directamente sin realizar ninguna operación en ellos.

Una vez recibidos los datos se ordenan y se realiza el sumatorio, en series que dependerán del número total de datos recibidos. Todos los datos se guardan en listas que serán posteriormente introducidas en el eje x e y. Finalmente, las listas se pasan a una función que crea el diagrama de barras.

```
function graficas2(json,muestras,nombrehist,titulox,log){

    json.sort(function(a,b){return a - b})
    var counts = {};
    var count = []; //Lista eje x.
    var datos = []; //Lista eje y.
    json.forEach(function(x) { counts[x] = (counts[x] || 0)+1; }); //Ordena los datos.
    var i = 1;
    var cuenta = 0;
    var ultimokey = 0;
    muestras = json[json.length-1]/20;
    for (var key in counts){
        var cuenta = cuenta + counts[key];
        if (key >= i*muestras ){
            count.push(">"+ultimokey+"=<"+key);
            var ultimokey = key;
            datos.push(cuenta);
            cuenta= 0;
            i++;
        }
    }
    count.push(">"+ultimokey+"=<"+key);
    datos.push(cuenta);
    console.log(key);
    tickintervalo=datos[0]/20;
    if (log==1){crearhist(count,datos,undefined,nombrehist,titulox,"logarithmic");}
        //Si el diagrama es logarítmico en el eje y.
    else{crearhist(count,datos,tickintervalo,nombrehist,titulox);}
}
```

Con el módulo basic bar se muestran diferentes histogramas diferenciados por:

- Número de SLOCs en cada paquete.
- Número de ficheros en cada paquete.
- Número de SLOCs fraccionado por el número de ficheros.
- Número de lenguajes en cada paquete.
- Número de ficheros del mismo lenguaje de programación por paquete.
- Número de SLOCs por fichero de cada lenguaje de programación (Debian se ha programado con 24 lenguajes diferentes, por lo tanto se muestran 24 histogramas diferentes).

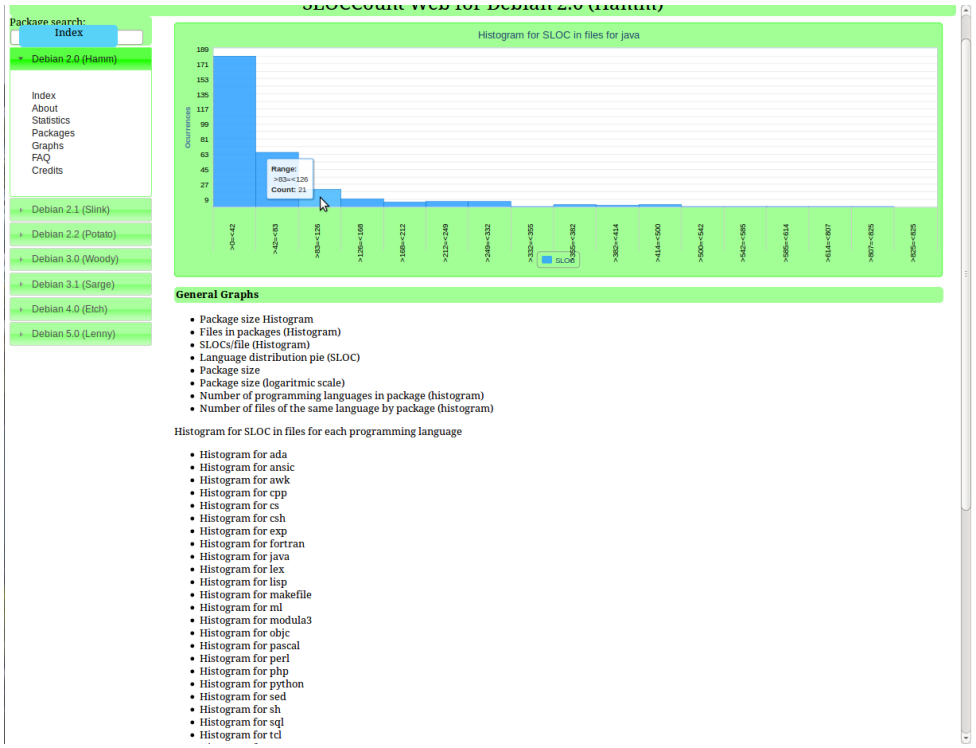

Figura 4.10: Captura de la sección graphs, histograma con el número de SLOCs por fichero del lenguaje Java (versión Hamm).

El módulo pie chart representa las proporciones de los lenguajes utilizados en un círculo. Debido a que se utilizan 24 lenguajes de programación y algunos son más utilizados que otros, se ha optado por representar la proporción de los cuatro lenguajes más utilizados y una proporción que indica el resto de lenguajes.

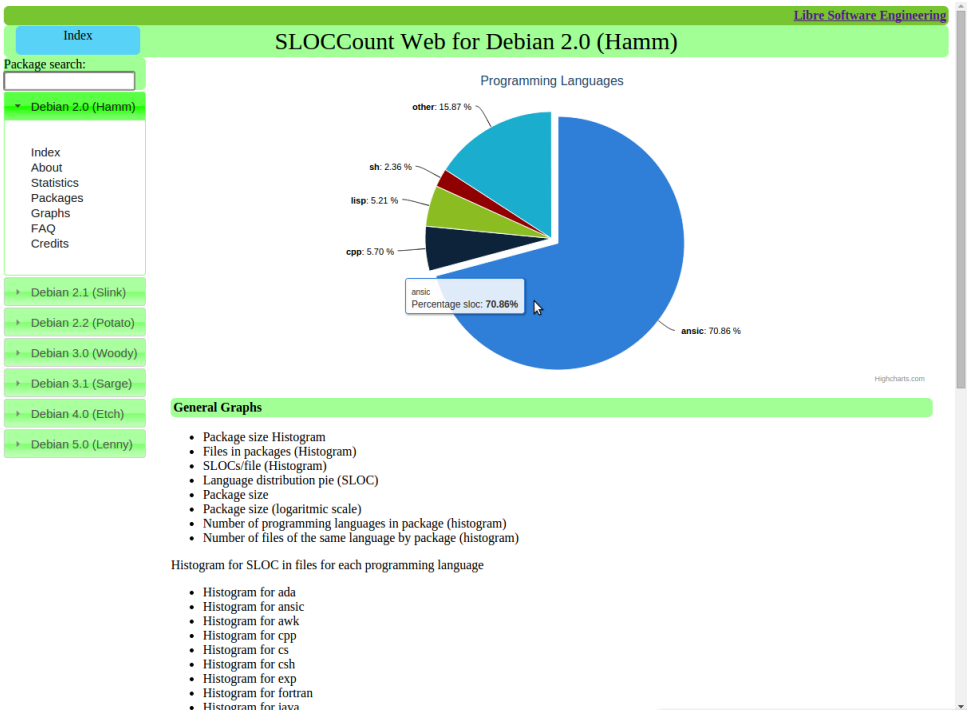

Figura 4.11: Captura de la sección graphs, pie chart de lenguajes de programación (versión Hamm).

Finalmente, en la web se utiliza el módulo splines con el que vamos a representar las siguientes estadísticas del modelo COCOMO. Este módulo representará las mismas gráficas en todas las versiones de Debian. Debido a que todos los datos que necesitamos para obtener estas gráficas no provienen del servidor sino de las fórmulas del modelo COCOMO, todo se ejecuta desde el cliente con Javascript. Tenemos tres grandes clases de rectas; de 1 a 1000 SLOC, de 1 a 1000 KSLOC y de 1 a 1000 MSLOC. Dentro de estas tres grandes clases tenemos las siguientes representaciones:

- COCOMO esfuerzo en personas-meses.
- COCOMO tiempo estimado de SLOC en meses.
- COCOMO tiempo estimado de esfuerzo en personas-meses.
- COCOMO promedio de desarrolladores.
- COCOMO estimación del coste total en miles de dolares.

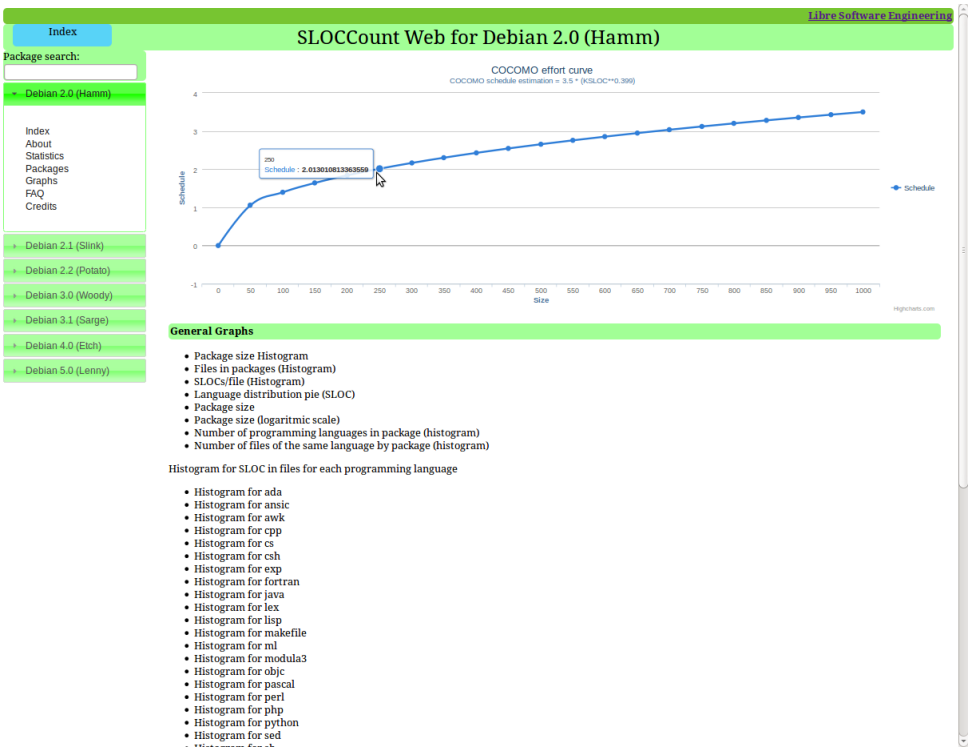

Figura 4.12: Captura de la sección graphs, splines curva de esfuerzo (Modelo COCOMO).

4.8.4. Buscador de paquetes

El buscador de paquetes se ha realizado con el elemento autocomplete de la biblioteca **JqueryUI**, de tal forma que mientras escribamos en el buscador este realizará llamadas al servidor y se desplegará una lista en la barra del buscador, proporcionándonos nombres de los cuales podemos seleccionar el que nos interesa. Una vez seleccionado el nombre del paquete, se realiza una llamada al servidor y reutilizando el código programado para la sección packages (cuando clickábamos el nombre de un paquete), se muestra en una ventana modal la respuesta del servidor, como ya describimos en el apartado 4.8.2. Hay que destacar que el buscador de paquetes funciona si estamos en una de las versiones de Debian (sabemos que estamos en una versión, porque el título de la web cambia y aparece el nombre de la versión en la que estamos), ya que buscará los paquetes de esa versión. En la Figura 4.13 se muestra un ejemplo de una lista desplegada del buscador de paquetes.

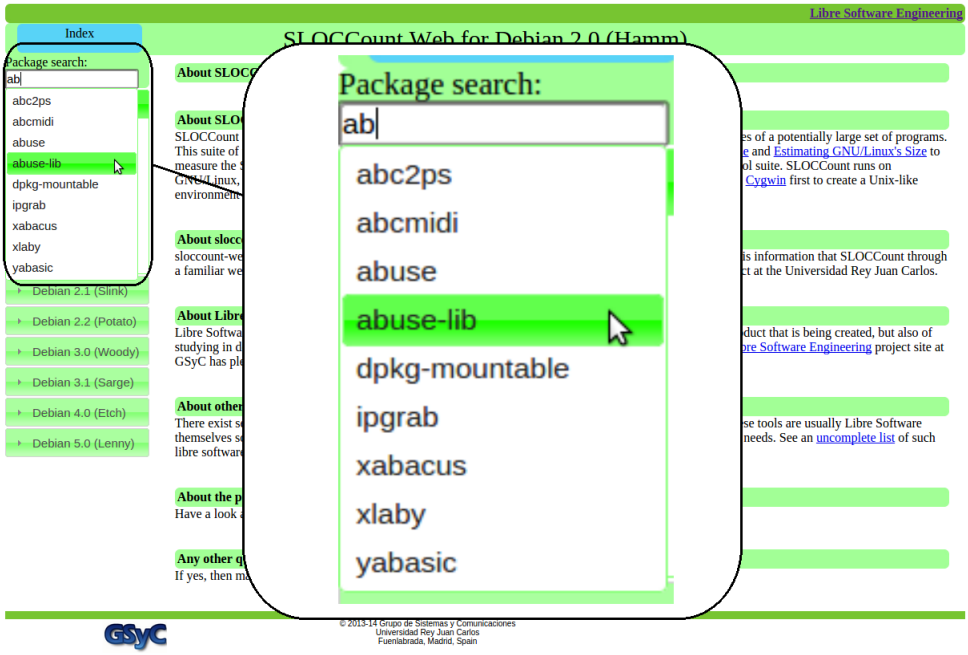

Figura 4.13: Captura del buscador de paquetes cuando tenemos *ab* en el buscador.

Capítulo 5

Conclusiones

Este proyecto se ha realizado con las tecnologías más recientes, lo que ha proporcionado que la aplicación web sea dinámica, flexible y vistosa para el usuario. Con el uso de la tecnología AJAX, el servidor no recibe llamadas que envíen contenido tan pesado como una página completa, ya que envía la información en formato JSON. Las bases de datos proporcionadas eran muy pesadas, por lo que al ser una web dinámica los cálculos que se hacen en el servidor necesitan de un tiempo razonable, pero con la introducción de la caché se consiguen disminuir esos tiempos concediendo al usuario una espera inapreciable. Con el uso de CSS3 y la biblioteca JqueryUI la web se convierte en un entorno mas dinámico e intuitivo para el usuario. Además, el usuario se encuentra en un entorno visual muy satisfactorio, debido al uso de efectos. Esta aplicación web proporciona un análisis sencillo del sistema operativo Debian y puede ser utilizada para resolver problemas analíticos de otros proyectos. Además, puede ser ampliada para nuevas versiones de Debian, ya que el código se ha encapsulado de tal forma que se le puedan añadir nuevas secciones y reutilizarlo para estas.

La idea del proyecto como una única página que solo envía y recibe información con peticiones AJAX, puede servir para futuros proyectos web de nuestra universidad. Algunos ejemplos pueden ser la concesión de trabajos fin de grado, becas, practicas externas o incluso el portal de servicios.

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 inmersión en el lenguaje Python, y con él en la programación orientada a objetos. Se adquirió cierta destreza en el manejo de querys.
- La profundización en la tecnología Django, en especial con la arquitectura MVC.
- El aprendizaje de los comandos básicos para poner en marcha y configurar un servidor de MySQL, además del modelado de objetos en Django.
- Se ha aprendido también a manejar LaTeX, el sistema de procesamiento de documentos empleado en la realización de esta memoria.
- El manejo de la tecnología AJAX, muy importante en el desarrollo del proyecto.
- El uso de una biblioteca de representación de gráficas dinámicas, como es el caso de Highcharts.

5.2. Conocimientos aplicados

Para el desarrollo del proyecto he aplicado conocimientos adquiridos durante la carrera.

El conocimiento más importante aportado a este proyecto es el desarrollo software con los lenguajes de programación Python y Javascript, ya que son las principales tecnologías utilizadas en el desarrollo de dicho proyecto.

Además, he aplicado conocimientos del framework Django y del protocolo HTTP, estudiados en la asignatura de Servicios y Aplicaciones Telematicas por la cual elegí este tipo de proyecto.

Por último, el conocimiento más importante que he adquirido durante el grado de Ingeniería en Tecnologías de la Telecomunicación es el de la búsqueda y análisis de información en cualquier tipo de plataforma, para obtener la solución a todo tipo de problemas.

5.3. Trabajos futuros

Como en todos los proyectos de software, se pueden mejorar e investigar para que el usuario disfrute en su totalidad del trabajo realizado.

1. **Optimizar tiempos**: El tiempo para los usuarios es un elemento muy importante cuando acceden a una web, ya que el usuario tiene la necesidad de recibir la información lo más rápido posible. Por ello, en este proyecto se podría realizar una investigación a fondo de toda la API que nos ofrece Django para realizar las operaciones de una forma más eficiente.

2. **Seguridad**: Algunos problemas de seguridad que se pueden encontrar en la web son:

   - Acceso a bases de datos y robo o corrupción de datos personales o confidenciales.
   - Modificación del código del sitio web.
   - Ataques de denegación de servicio (DoS) que deshabilitan la disponibilidad de los servicios.

   Por lo tanto, nuestra aplicación web debe ser protegida frente a los ataques de los usuarios. Añadir un certificado SSL sería una posible mejora.

3. **Aumentar Base de datos**: Nuestra base de datos no tiene las últimas dos versiones de Debian, que son Squeeze y Wheezy. Estas últimas dos versiones se podrían analizar con SLOCCount, almacenar en una base de datos SQL y posteriormente introducirse en la web.

4. **Cambio de SQL a NoSQL**: Esta posible mejora depende del pensamiento de cada desarrollador. Pienso que cambiando la base de datos a NoSQL mejoraría el acceso a los datos, ya que se accedería de una forma más rápida y sencilla entre los datos de la base Packages y Files.

Apéndice A

Instalación en la Maquina Virtual

La aplicación se ha llevado a producción en una maquina virtual de la Universidad Rey Juan Carlos proporcionada por el grupo Libresoft. Es conveniente enumerar los pasos que he seguido para ayudar a futuros alumnos.

- Copiar las bases de datos a la maquina virtual y la aplicación:

```
scp -r /ruta-fichero-origen nombreusuario@ipmaquinavirtual:/ruta-fichero-destino
```

- Instalar MySQL:

```
sudo apt-get install mysql-client
sudo apt-get install mysql-server
```

- Añadir las bases de datos a MySQL.
- Instalar Python:

```
sudo apt-get install python2.7
```

- Instalar Django:

```
sudo pip install django
```

- Instalar Python_MySQL:

```
sudo apt-get install python-mysqldb
```

Finalmente, necesitamos crear un VirtualHost en el servidor Apache. Entramos en la ruta `/etc/apache2/sites-available` donde se creara un fichero, en nuestro caso hemos elegido el siguiente nombre *debiancounting.conf* en el que introducimos lo siguiente:

```
<VirtualHost IP:80>
    ServerName Nombre del dominio
    DocumentRoot ruta aplicación
    <Directory ruta aplicación>
        Order allow,deny
        Allow from all
    </Directory>
    WSGIDaemonProcess nombreaplicación.djangoserver processes=4 threads=15 display-name=%{GROUP}
    WSGIProcessGroup nombreaplicación.djangoserver
    WSGIScriptAlias / rutaaplicación/apache.conf/web.wsgi
    Alias /static rutaaplicación/static
    <Directory rutaaplicación/static>
        Order deny,allow
        Allow from all
    </Directory>
</VirtualHost>
```

Además, se debe acudir al fichero *hosts* alojado en `/etc` y añadir el nombre del dominio que queremos utilizar. Por último, se enlaza y activa la aplicación (Hay que avisar al hosting para que nos proporcione el dominio):

```
cd /etc/apache2/sites-available && sudo a2ensite webproyecto\\
sudo /etc/init.d/apache2 restart
```

Bibliografía

[1] Web del proyecto Django.

`https://www.djangoproject.com/`

[2] Web del proyecto Jquery.

`https://jquery.com/`

[3] Web del proyecto JqueryUI.

`https://jqueryui.com/`

[4] Web del proyecto Highcharts.

`http://www.highcharts.com/`

[5] Web del proyecto Apache.

`http://httpd.apache.org/`

[6] Web de aprendizaje para desarrolladores.

`http://www.w3schools.com/`

[7] Comunidad para desarrolladores.

`http://stackoverflow.com/`

[8] Web del proyecto SLOCCount.

`http://www.dwheeler.com/sloccount/`

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

`http://gsyc.urjc.es`

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

`http://libresoft.urjc.es`

[11] Web de la aplicación.

http://debian-counting-new.libresoft.es/

[12] Repositorio Github.

https://github.com/roberro/DebianCounting