Universidad
Rey Juan Carlos

MÁSTER UNIVERSITARIO
EN SISTEMAS TELEMÁTICOS E INFORMÁTICOS

Curso Académico 2012/2013

Trabajo de Fin de Máster

Web de Debian Counting en Django

Autor: Meilin Xu

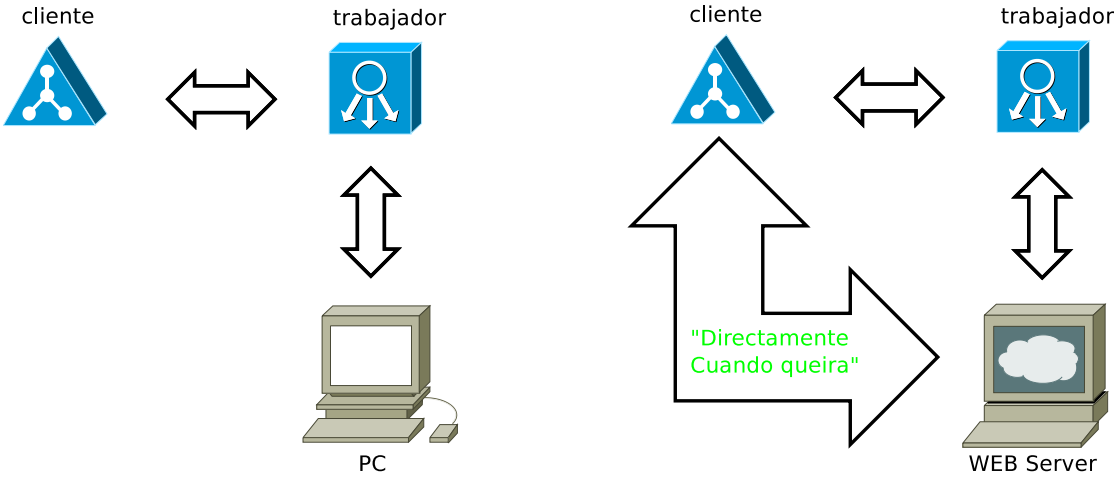
Tutor: Dr. Gregorio Robles

Resumen

Una vez trabajé en una distribuidora mayorista[1] de productos electrónicos. La empresa vende principalmente baterías portátiles. Todo estaba inventariado en un documento con diferentes marcas y modelos de los productos. El proceso de negocio era muy ineficiente. Si un cliente quería alguna información de las baterías, nos llamaba y buscábamos en el documento, después le enviábamos la información que nos había pedido. Si tuviéramos una web con una base de datos, el proceso podría ser más eficiente, ya que los clientes podrían buscar los productos cuando quisieran, como figura abajo.

Desde la creación de la WWW[2], la tecnología de desarrollo del web ha ido evolucionando mucho, y hay una variedad de técnicas. Entre las técnicas hay una tendencia común: más económico, más rápido y menos trabajo. Tienen características como las siguientes: usan software FLOSS, usan componentes LAMP, usan Web Frameworks.

Así, el presente trabajo presenta una de estas técnicas y realiza un sitio web soportado por una base de datos usando LAMP software, un web framework que se llama Django, y con una base de datos existentes de la web Debian Counting[3]del grupo de investigación GSyC/LibreSoft de la Universidad de Rey Juan Carlos.



[1]Distribuidor mayorista: es un componente de la cadena de distribución, en que la empresa o el empresario, no se pone en contacto directo con los consumidores o usuarios finales de sus productos, sino que entrega esta tarea a un especialista.
[2]WWW: World Wide Web
[3]Debian Counting: http://debian-counting.libresoft.es

Índice

1. Introducción

En este apartado se realiza una breve introducción a temas importantes para crear una web dinámica, explicando sus conceptos básicos, componentes e historia.

1.1. FLOSS[1]

Free/libre/open-source software (FLOSS) es un término que se utiliza para no tener que entrar en la discusión entre free software y open source software, ya que engloba a ambas. Sus licencias conceden a los usuarios el derecho de usar, copiar, estudiar, cambiar y mejorar su diseño mediante la disponibilidad de su código fuente.

El acrónimo FLOSS fue acuñado en 2001 por Rishab Aiyer Ghosh. Más tarde, ese mismo año, la Comisión Europea (CE) utilizó la frase cuando financió un estudio sobre el tema [2].

- Free software: Definición de Software Libre [3] de Richard Stallman, aprobada por la Free Software Foundation (FSF) [4], define el software libre como un asunto de libertad, no de precio. La publicación más antigua conocida de la definición de la idea del software libre fue en la edición de febrero 1986 de la publicación Boletín ahora discontinuado de GNU de la FSF. La fuente canónica para el documento está en la sección de filosofía de la página web del Proyecto GNU. En abril de 2008 se encuentra publicada en 39 idiomas.

- Open-source: La definición de open-source es utilizado por la Open Source Initiative [5] para determinar si una licencia de software califica para la insignia de la organización para el software de código abierto. La definición se basa en las directrices de software libre de Debian, escrito y adaptado principalmente por Bruce Perens. Perens se basó en las cuatro libertades del software libre de la Free Software Foundation, que eran demasiado genéricas y abstractas para poder ser utilizadas en la distribución de Debian.

1.2. LAMP[6]

LAMP es una combinación de software libre, de código abierto. El acrónimo LAMP se refiere a las primeras letras de Linux (sistema operativo), Apache HTTP Server(servidor de web), MySQL (software de base de datos) y PHP, Perl o Python(lenguajes de programar), los componentes principales para construir un servidor web de propósito general viable.

EL término LAMP fue acuñado en 1998 for Michael Kunze en la revista alemana c't [7].

Aunque los autores originales de estas tecnologías no los diseñaron para trabajar específicamente con las demás, la filosofía y las herramienta de desarrollo son compartidas y se han desarrollado en estrecha colaboración. La combinación del software se ha vuelto popular debido a que es libre de costo, de código abierto, y por lo tanto fácilmente adaptable. Asimismo, se incluyen con las distribuciones de Linux, por lo que son fáciles de

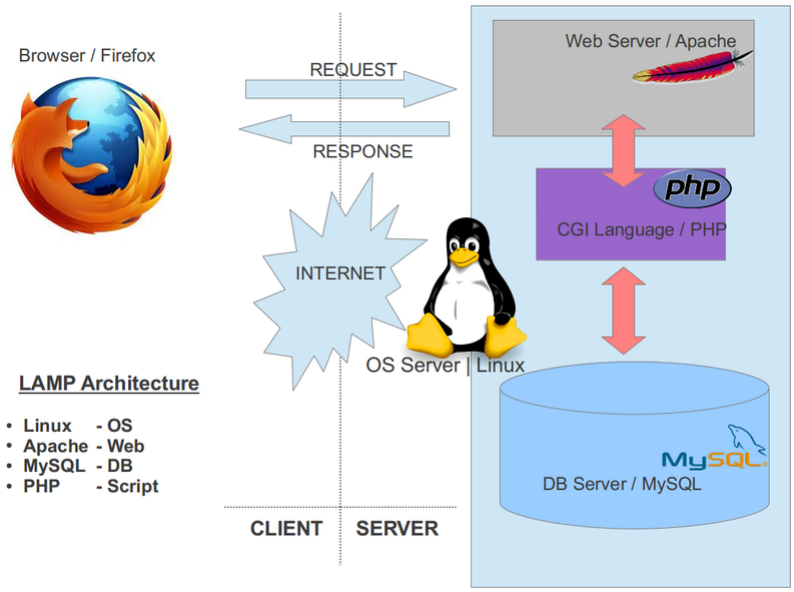


Figura 1: LAMP Arquitectura[8]

adquirir e instalar. Cuando se usan juntos, apoyan a los servidores de aplicaciones web tal y como se muestra en la figura 1.

- Linux [9]: Linus es el núcleo del sistema operativo, y generalmente se utiliza para nombrar el sistema operativo completo, aunque en ese caso es más completo hablar de GNU/Linux.
- Apache [10]: Apache HTTP Server es un servidor web de código abierto, el más popular en uso. Es un proyecto de Apache Software Foundation.
- MySQL [11]: MySQL es un sistema de gestión de base de datos (DBMS) multi-usuario y multiproceso. Existen alternativas a MySQL, por ejemplo mediante el uso de PostgreSQL (en cuyo caso, la combinación de tecnologías recibe el acrónimo de LAPP). MySQL ha sido propiedad de Oracle Corporation[12] desde el 27 de enero de 2010 hasta la compra de Sun Microsystems. Sun había adquirido MySQL originalmente el 26 de febrero de 2008.
- PHP: PHP [13] es un lenguaje de programación reflexivo diseñado originalmente para la producción de páginas web dinámicas. PHP se utiliza principalmente en aplicaciones de software de servidor, aunque también se puede utilizar desde la línea de comandos.
- Perl: Perl se pueden utilizar de manera similar a PHP, aunque su propósito inicial es más amplio. Tuvo gran apogeo con los CGIs, pero con la aparición de PHP -más específico para programar en el servidor- o Python -más moderno y legible- ha perdido popularidad en los últimos tiempos.

- Python: Python [14] es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia 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. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License,1 que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

1.3. Web Application Framework[15]

Un framework para aplicaciones web [16] es un framework diseñado para apoyar el desarrollo de sitios web dinámicos, aplicaciones web y servicios web. Este tipo de frameworks tiene como objetivo servir de ayuda en las actividades más comúnmente usadas en desarrollos web. Por ejemplo, muchos framework proporcionan bibliotecas para acceder a bases de datos, estructuras para plantillas y gestión de sesiones, y con frecuencia facilitan la reutilización de código.

En sus comienzos, la World Wide Web no era intrínsecamente dinámica, ya que las páginas web consistían en documentos de texto escritos en HTML. Para cualquier modificación de las páginas web, era necesario que el autor de las mismas realizara las modificaciones. Para proporcionar una página web dinámica que refleja las entradas del usuario, se introdujo el estándar Common Gateway Interface (CGI) que permitía conectar aplicaciones externas con los servidores web. Después, como hemos visto, se han desarrollado nuevos lenguajes específicamente para su uso en la web, tales como ColdFusion, PHP y páginas Active Server.

Finalmente, aparecieron frameworks "full stack", que a menudo se reunieron varias bibliotecas útiles para el desarrollo web en una sola pila de software coherente para los desarrolladores web para su uso. Ejemplos de esto incluyen ASP.NET, WebObjects, OpenACS, Ruby on Rails, Catalyst, Mojolicious, Zend Framework, Yii, CakePHP, Symfony, web2py, Django.

MVC *Modelo-Vista-Controlador*, muchos frameworks siguen el patrón de arquitectura MVC para separar el modelo de datos con reglas de negocio de la interfaz de usuario. Esto se considera generalmente una buena práctica ya que permite tener división de tareas, promueve la reutilización de código, y permite a múltiples interfaces que han de aplicarse. En las aplicaciones Web, esto permite que los diferentes puntos de vista que se presentarán, como las páginas web (pensadas para seres humanos), y las interfaces de servicios web para aplicaciones remotas.

Comparación Se realiza una comparación de los diferentes web frameworks, en el cuadro 1.

Lenguaje	Proyecto	Licencia	MVC	DB migración	Plantilla
ASP.NET	ASP.NET MVC	Apache v2	si	si	si
Java	WebOjects	Propietario	si	si	si
Ruby	Ruby on Rails	MIT, Ruby	si	si	si
Tcl	OpenACS	GPL	si	si	si
Perl	Catalyst	Artistic, GPL	si		si
Perl	Mojolicious	Artistic	si		si
PHP	Zend Framekwork	New BSD	si	si	si
PHP	Yii	New BSD	si	si	si
PHP	CakePHP	MIT	si	si	si
PHP	Symfony	MIT	si	Plugin	si
Pyhon	web2py	LPGL3	si	si	si
Pyhon	Django	BSD	si	si	si

Cuadro 1: Comparación de los Web Frameworks

1.4. Django

1.4.1. Descripción

Django[17] es un framework web de alto nivel escrito en Python que fomenta el rápido desarrollo y el diseño limpio y pragmático.

> The Web framework for perfectionists with deadlines.
> Django makes it easier to build better Web apps more quickly and with less code.

> El lema de Django es "El framework web para perfeccionistas con fechas límite". Esto se debe a que Django hace que sea más fácil de construir mejores aplicaciones web más rápido y con menos código.

Django[18] 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.

1.4.2. Característica

Django es un framework de desarrollo web de código abierto, que cumple en cierta medida el paradigma del *Modelo Vista Controlador*.

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 [19]). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

Las características de Django:

- Diseño elegante URL: Diseño de URLs sin limitaciones específicas del framework. Es tan flexible como se desee.
- Sistema de plantillas: Utiliza potente, extensible de Django y diseñador de usar lenguaje de plantillas de diseño independiente, el contenido y el código de Python.
- Mapeador objeto-relacional: Defina sus modelos de datos en su totalidad en Python. Usted recibe una rica API dinámica base de datos de acceso de forma gratuita - pero todavía se puede escribir SQL si es necesario.
- Interfaz de administración automática: Ahórrese el trabajo tedioso de crear interfaces para la gente a añadir y actualizar contenido. Django hace automáticamente, y es listo para la producción.
- Sistema de caché: Hook en los marcos de caché memcached o de otro tipo para un rendimiento súper - el almacenamiento en caché es tan granular como usted necesita.
- Internacionalización Django tiene soporte completo para aplicaciones multi-lenguaje y permite especificar cadenas de traducción y ofrecer ganchos para la funcionalidad específica del idioma.

1.4.3. Arquitectura

Aunque Django está fuertemente inspirado en la filosofía de desarrollo Modelo Vista Controlador, sus desarrolladores declaran públicamente que no se sienten especialmente atados a observar estrictamente ningún paradigma particular, y en cambio prefieren hacer "lo que les parece correcto". Como resultado, por ejemplo, lo que se llamaría "controlador" en un "verdadero" framework MVC se llama en Django "vista", y lo que se llamaría "vista" se llama "plantilla".

En el caso de querer hacer una correspondencia, entonces diríamos que éste es un framework *"MTV"*:

M Modelo,

T Template (plantilla)

V Vista

1.4.4. Los sitios que usan Django

Algunos sitios conocidos que utilizan Django incluyen:

Disqus Hermosa, en tiempo real, con la participación comentarios hilos para su sitio web.

Instagram Una aplicación para compartir fotos rápido, bello y divertido para iOS y Android.

Mozilla Los fabricantes de Firefox, y los defensores de la apertura, innovación y oportunidad en la web.

OpenStack Software de código abierto para la construcción de nubes privadas y públicas.

Pinterest Pinterest es un tablón de anuncios virtual para compartir las cosas hermosas que encuentres en la web.

PolitiFact.com El sitio de comprobación de los hechos, ganador del Premio Pulitzer.

Rdio Tu biblioteca musical en la nube.

Encontrará una lista más completa en djangosites[4].

[4] djangosites: http://www.djangosites.org/

2. Objetivos

Este capítulo, describe los objetivos del presente trabajo, realiza un web de Debian Counting impulsado por una base de datos de MySQL con Web Framework Django, LAMP, FLOSS, también los requisitos que debe cumplir su desarrollo.

2.1. Descripción del problema

SLOCCount es un conjunto de herramientas para el conteo de líneas físicas de código fuente (SLOC) en un gran número de idiomas de un conjunto potencialmente grande de los programas. Este conjunto de herramientas se utiliza en los papeles de David A. Wheeler Más que un Gigabuck: Estimación de GNU / Linux 's Tamaño y Estimación de GNU / Linux' s Tamaño de medir el SLOC de las distribuciones de GNU / Linux enteras. Otros han medido Debian GNU / Linux a través de este conjunto de herramientas. SLOCCount funciona en GNU / Linux, FreeBSD, Windows, y espero que en otros sistemas también. Para ejecutar en Windows, usted tiene que instalar Cygwin primero en crear un entorno Unix para SLOCCount. (texto tomado de la página web de SLOCCount y ligeramente modificado.)

Sloccount-Web de Libre Software Engineering de Universidad de Rey Juan Carlos es una interfaz web para los datos generados por SLOCCount. Se proporciona información adicional y análisis estadísticos que sloccount través de una interfaz web familiar. sloccount-web ha sido implementado como parte del proyecto de Ingeniería de Software Libre de la Universidad Rey Juan Carlos.

En el Sloccount-Web:

- Statistics es un lugar donde se muestran estadísticas macroeconómicas. Esto incluye una lista ordenada de los idiomas que se utilizan, así como una estimación del límite superior de esfuerzo, la programación y el costo.
- Packages contiene una lista de todos los paquetes (origen) y sus estadísticas. Los paquetes pueden ser seleccionados para ver las estadísticas de idiomas y el archivo y estimaciones de esfuerzo en ellos.
- Graphs enlaza varias figuras que pueden ayudar a comprender la naturaleza del software que se está estudiando.
- Un cuadro de búsqueda que encuentres un paquete determinado por su nombre.

Debian Counting de Ingeniería de Software Libre ha realizado con PHP como figura 2, y en nuestro trabajo usará el Web Framework y el lenguaje Python como figura 3.

2.2. Requisitos

El trabajo deberá cumplir ciertos requisitos básicos:
Definir las urls para las páginas ,usando expresión regular
Diseñar un plantilla para las páginas, usando django plantilla
Importar utilizando los base de datos de antiguo ,utiliza Django MTV modelo

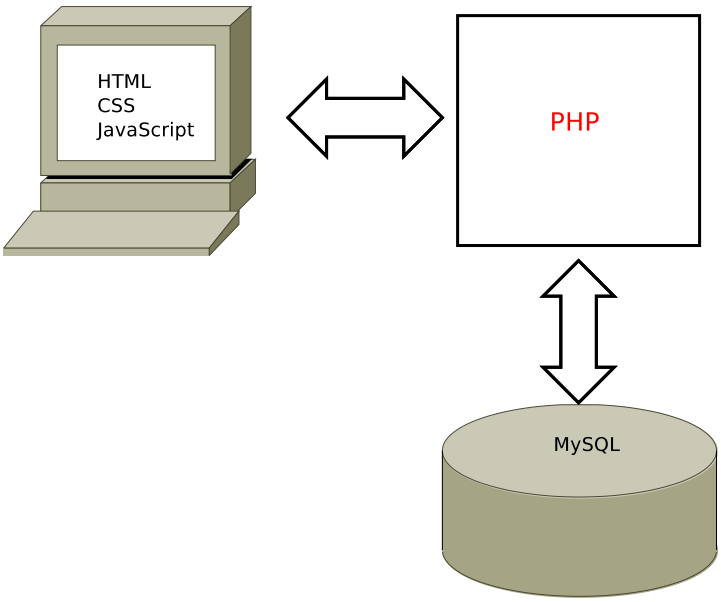


Figura 2: Debian Counting PHP

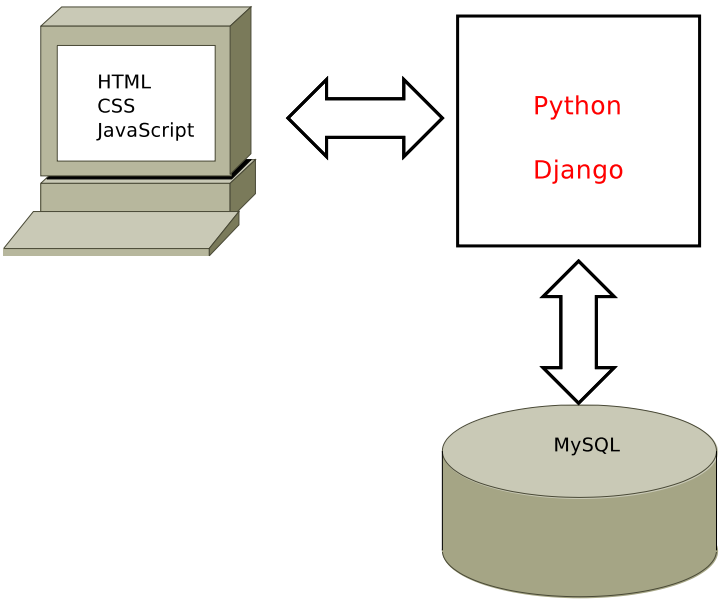


Figura 3: Debian Counting Django

3. Plataforma de desarrollo

Este capítulo aporta los detalles de instalar sobre las plataformas de trabajo utilizadas para el desarrollo del presente proyecto: Python, Django, MySQL, MySQLdb, Apache.

Linux:la Sistema operativo que usamos es Ubuntu12.04, un popular sistema desde Debian.

Django requiere Python 2.5 o superior. No se necesitan otras bibliotecas de Python para poder obtener una funcionalidad básica. En un entorno de desarrollo –especialmente si queremos experimentar con Django—no necesitamos un web servidor instalado, ya que Django trae su propio servidor liviano para éste propósito, con la restricción de solo permitir un usuario a la vez.

3.1. Python

En la mayoría distribución de sistema Linux, el Python intérprete ya existe. Se pude escribir en un terminal la comando *Python* para autenticación, de la siguiente manera:

```
localhost:~$ python
```

La ejecución de este comando inicia un intérprete interactivo de Python, mientras que habrá el siguiente resultado:

```
localhost:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> [5]
```

Si no instala intérprete Pyhotn, es posible que aparezca el siguiente mensaje de error:

```
localhost:~$ python
python: command not found
```

En este momento, usted tiene que instalar Python propio, se describen a continuación dos métodos[20]:

1. Utilice Package Manager
2. Compila desde la fuente de codigo

[5] Pulse Ctrl + D para salir del intérprete interactivo.

3.1.1. Utilice Package Manager

Instalación de Python en Debian GNU / Linux con el comando *apt:*

```
localhost:~$ su -
Password: [enter your root password]
localhost:~# apt-get install python
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed: python2.7.3
Suggested packages: python-tk python2.7.3-doc
The following NEW packages will be installed:
python python2.7.3
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
Need to get 0B/2880kB of archives.
After unpacking 9351kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Selecting previously deselected package python2.7.3.
(Reading database ... 22848 files and directories currently installed.) Unpacking pyt
Selecting previously deselected package python.
Unpacking python (from .../python_2.7.1-1_all.deb) ...
Setting up python (2.7.1-1) ... Setting up python2.7.3 (2.7.1-1) ... Compiling python
Compiling optimized python modules in /usr/lib/python2.7.3 ...
localhost:~# exit
logout
localhost:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> [press Ctrl+D to exit]
localhost:~$
```

3.1.2. Compila desde la código fuente

Si prefiere construir desde el código fuente, puede descargar el código fuente de Python desde http://www.python.org/ftp/python/. Seleccione el número de versión más alto, descargue el archivo tgz., Y luego hacer el ritual habitual de configure, make, make install:

```
localhost:~$ su -
Password: [enter your root password]
localhost:~#  wget http://www.python.org/ftp/python/2.7.3/Python-2.7.3.tgz --2013-06-
100%[======================================>] 14,135,620   588K/s   in 39s
2013-06-16 19:56:27 (352 KB/s) - ‘Python-2.7.3.tgz’ saved [14135620/14135620]
localhost:~# tar xfz Python-2.7.3.tgz
```

```
localhost:~# cd Python-2.7.3
localhost:~/Python-2.7.3# ./configure
checking MACHDEP... linux2
checking EXTRAPLATDIR...
checking for --without-gcc... no ...
localhost:~/Python-2.7.3# make
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -O3 -Wall -Wstrict-prototypes -I. -I
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -O3 -Wall -Wstrict-prototypes -I. -I
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -O3 -Wall -Wstrict-prototypes -I. -I
localhost:~$ which python
/usr/local/bin/python
localhost:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
 [GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> [press Ctrl+D to get back to the command prompt]
localhost:~$
```

3.2. Django

Instalación de Django [21]

En un momento dado, dos versiones distintas de Django están disponibles para usted: la última versión oficial y la versión de desarrollo bleeding-edge. La versión decide instalar depende de sus prioridades ¿Quieres una versión estable y probada de Django, o.. ¿quieres una versión que contiene las funciones más recientes, tal vez por lo que puede contribuir al propio Django, a expensas de la estabilidad?

Nos gustaría recomendar la pervivencia de una versión oficial, pero es importante saber que existe la versión de desarrollo, ya que le resultará menciona en la documentación y de los miembros de la comunidad.

3.2.1. Instalación de una versión oficial manualmente

Para instalar una Django versión oficial, siga las pasos continuación:

1. Descarga la última versión de página de descargas: https://www.djangoproject.com/download.
2. Descomprimir el archivo descargado (por ejemplo tar xzvf Django-XY.tar.gz, donde XY es el número de versión de la última versión), con el comando: *tar xzvf Django-XY.tar.gz*
3. Cambie al directorio creado en el paso 2 : CD *Django-XY*
4. Escriba en el indicador de comandos el comando: *sudo python setup.py install*

En caso de que usted es curioso: los archivos de Django se instalarán en el directorio site-packages de su instalación de Python - un directorio donde Python busca bibliotecas de terceros. Por lo general, se encuentra en un lugar como / usr/lib/python2.7/site-packages.

3.2.2. Comprobación de la instalación Django

Por algunos comentarios positivos después de la instalación, tome un momento para comprobar si la instalación funcionó. En una terminal, el cambio a otro directorio (por ejemplo, no el directorio que contiene el directorio django) e iniciar el intérprete interactivo de Python escribiendo python. Si la instalación se ha realizado correctamente, usted debería ser capaz de importar el módulo django:

```
meilin@Master:~$ python
Python 2.7.3 (default, Aug  1 2012, 05:16:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION
(1, 5, 1, 'final', 0)
>>>
```

3.3. MySQL y MySQLdb

En este punto, usted podría comenzar a escribir una aplicación web con Django, porque sólo requisito es una instalación de Python. Sin embargo, lo más probable es que usted estará desarrollando un sitio Web con bases de datos, en cuyo caso tendrá que configurar un servidor de base de datos.

Django es compatible con cuatro motores de base de datos:

- PostgreSQL (http://www.postgresql.org/)
- SQLite 3 (http://www.sqlite.org/)
- MySQL (http://www.mysql.com/)
- Oracle (http://www.oracle.com/)

En su mayor parte, todos los motores de aquí funcionan igual de bien con el marco básico de Django. (Una excepción notable es el soporte GIS opcional de Django, que es mucho más potente con PostgreSQL que con otras bases de datos.) Si usted no está atado a ningún sistema existente y tienen la libertad de elegir un motor de base de datos, se recomienda PostgreSQL, que logra una delicado equilibrio entre costo, características, velocidad y estabilidad.

La creación de la base de datos es un proceso de dos pasos:

En primer lugar, usted tendrá que instalar y configurar el mismo servidor de base de datos.

En segundo lugar, tendrá que instalar la biblioteca de Python para su base de datos en particular. Esto es un poco de terceros de código que permite Python para interactuar con la base de datos. Planteamos las necesidades específicas de bases de datos por las siguientes secciones.

En nosotros trabajo usaremos Django con MySQL :

3.3.1. MySQL

Django requiere MySQL 4.0 o superior. Las versiones 3.x no soportan subconsultas anidadas y otras sentencias SQL bastante estándar.

Instala MySQL con los comandos siguente:

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

intermedio puede introducir la contraseña del usuario root.

Comparación de la instalación de MySQL:

```
$sudo netstat -tap | grep mysql
```

si hay información como siguiente, la instalación está correcto.

```
tcp        0      0 localhost:mysql         *:*                     LISTEN
1053/mysqld
```

Comandos de entrar y cambiar contraseña de root:

```
$mysql -uroot -p
$sudo mysqladmin -u root password newpassword
```

3.3.2. MySQLdb

También tendrá que instalar el paquete MySQLdb de http://www.djangoproject.com/r/python-mysql/. En Linux, compruebe si el sistema de gestión de paquetes de su distribución ofrece un paquete llamado "python-mysql", "python-mysqldb", "mysql-python" o algo similar, en nuestra sistema Ubuntu 12.04,con el comando: *apt-get install python-mysqldb*

```
xxx:~$ sudo apt-get install python-mysqldb
[sudo] password for xxx:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:   python-mysqldb-dbg
```

```
The following NEW packages will be installed:   python-mysqldb
...
Selecting previously unselected package python-mysqldb.
(Reading database ... 300955 files and directories currently installed.)
Unpacking python-mysqldb (from.../python-mysqldb_1.2.3-1ubuntu0.1_i386.deb) ...
Setting up python-mysqldb (1.2.3-1ubuntu0.1) ...
xxx:~$
```

3.4. Apache

Instalar Apache y mod_wsgi

Si lo que desea es experimentar con Django, vaya a la siguiente sección, Django incluye un servidor web ligero se puede utilizar para las pruebas, por lo que no tendrá que configurar Apache hasta que esté listo para implementar Django en producción.

Si desea utilizar Django en un centro de producción, el uso de Apache con mod_wsgi. mod_wsgi puede funcionar en uno de dos modos: un modo integrado y un modo demonio. En el modo integrado, mod_wsgi es similar a mod_perl - que integra Python en Apache y carga código Python en la memoria cuando se inicia el servidor. Código permanece en la memoria durante toda la vida de un proceso de Apache, lo que conduce a mejoras de rendimiento significativas sobre otros arreglos de servidor. En el modo de demonio, mod_wsgi genera un proceso demonio independiente que controla las solicitudes. El proceso demonio se puede ejecutar como un usuario diferente que el servidor Web, que puede dar lugar a una mayor seguridad y el proceso de demonio puede reiniciarse sin necesidad de reiniciar todo el servidor Web Apache, posiblemente haciendo la actualización de su base de código más fluida. Consulte la documentación mod_wsgi para determinar cuál es el modo adecuado para su instalación. Asegúrese de que tiene Apache instalado, con el módulo mod_wsgi activado. Django funciona con cualquier versión de Apache que apoya mod_wsgi.

Consulte Cómo utilizar Django con mod_wsgi para obtener información sobre cómo configurar mod_wsgi vez que lo tenga instalado.

Si usted no puede utilizar mod_wsgi por alguna razón, no temas: Django soporta muchas otras opciones de implementación. Uno de ellos es uWSGI, funciona muy bien con nginx. Otra es FastCGI, perfecto para usar Django con servidores que no sean Apache. Además, Django sigue la WSGI especificación (PEP 3333), que permite que se ejecute en una variedad de plataformas de servidor. Consulte la página wiki server-arreglos para obtener instrucciones de instalación específicas para cada plataforma.

4. Descripción informática

Este capítulo aporta todos los detalles referentes a la implementación del web Debian Counting sobre el web framework Django. En las siguientes secciones se expondrá la estructura del web realizado.

La primera sección, realiza una descripción detalladas del método de iniciar un proyecto con Django. La segunda, introduce el diseño de modelo V(vista de MTV)y va a realizar un básico web. Las siguientes detallan cada esquema en particular, como son T(template de MTV), M(modelo de MTV).

Antes de todo, para puede cumplir los trabajos, se necesita obtener los conocimientos y tecnologías básicas sobre los temas siguientes:

HTML Hyper Text Markup Language, para crear las páginas de web.

CSS Cascading Style Sheets, para hacer la página se vea mejor.

JavaScript ECMAScript, DOM, BOM, para realizar algunos funciones de la web.

Python se usa durante todo el proceso.

MySQL las operaciones básicas como: importación de base de datos, consulta la base de datos,etc.

4.1. Iniciar un proyecto

Una vez que haya instalado Python, Django y (opcionalmente) el servidor de base de datos / biblioteca, se pueden dar el primer paso en el desarrollo de una aplicación de Django mediante la creación de un proyecto.

Un proyecto es un colección de configuraciones de una instancia de Django, incluyendo configuración, opciones específicas de Django base de datos y los ajustes específicos de la aplicación.

4.1.1. Crear el proyecto *debian_counting*

Si esta es tu primera vez usando Django, tendrá que hacerse cargo de una configuración inicial. Crear un nuevo directorio para empezar a trabajar en el, tal vez algo así como / home / username / djcode /.

Cambie al directorio que ha creado, y ejecute el comando *django-admin.py startproject debian_counting* . Esto creará un directorio *debian_counting* en el directorio actual.

El comando startproject crea un directorio que contiene cinco archivos:

```
debian_counting/

    manage.py
    debian_counting/
```

```
__init__.py
settings.py
urls.py
wsgi.py
```

Estos archivos son los siguientes:

debian_counting/: El exterior directorio DEBIAN_COUNTING/ es sólo un contenedor para su proyecto. Su nombre no importa a Django, se puede cambiar el nombre a algo que te gusta.

manage.py: Una utilidad de línea de comandos que le permite interactuar con este proyecto de Django de varias maneras. Escriba *python manage.py help* a tener una idea de lo que puede hacer. Usted nunca debería tener que editar este archivo, sino que ha creado en este directorio por pura conveniencia.

debian_counting/debian_counting/: El interior directorio DEBIAN_COUNTING/DEBIAN_COUNTING/ es el paquete de Python real para su proyecto. Su nombre es el nombre del paquete Python que tendrás que usar para importar nada en su interior (por ejemplo, la importación mysite.settings).

__init__.py: Un archivo necesario para Python para tratar el directorio misitio como un paquete (es decir, un grupo de módulos de Python). Es un archivo vacío, y por lo general no añadirá nada a ella.

settings.py: Ajustes / configuración para este proyecto Django. Echa un vistazo a él para tener una idea de los tipos de configuraciones disponibles, junto con sus valores por defecto.

urls.py: La URL para este proyecto Django. Piense en esto como el "contenido" de su sitio Django.

wsgi.py: Un punto de entrada para los servidores web compatibles con WSGI para servir a su proyecto. Ver Cómo implementar con WSGI (https://docs.djangoproject.com/en/1.4/howto/deplo para más detalles.

A pesar de su pequeño tamaño, estos archivos ya constituyen una aplicación de Django trabajo.

4.1.2. Ejecutar el servidor de desarrollo

Django incluye un ligero web servidor que puedes usar mientras estás desarrollando tu sitio. Incluimos este servidor para que puedas desarrollar tu sitio rápidamente, sin tener que lidiar con configuraciones de servidores web de producción (i.e., Apache) hasta que estés listo para la producción. Este servidor de desarrollo vigila tu código a la espera de cambios y se reinicia automáticamente, ayudándote a hacer algunos cambios rápidos en tu proyecto sin necesidad de reiniciar nada.

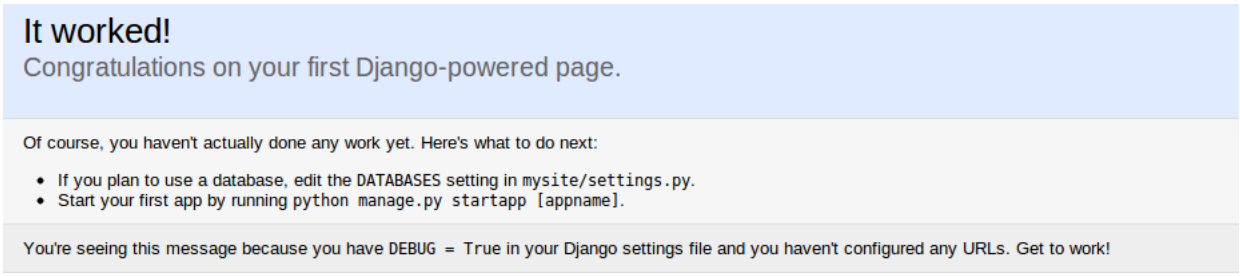


Figura 4: django bienvenido web

Entra en el directorio DEBIAN_COUNTING, y ejecuta el comando *python manage.py runserver*. Verás algo parecido a esto:

```
Validating models...
0 errors found June 17, 2013 - 19:35:44
Django version 1.5.1, using settings 'debian_counting.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Esto inicia el servidor de forma local, en el puerto 8000, accesible sólo para conexiones desde su propio ordenador. Ahora que se está ejecutando, visite http://127.0.0.1:8000/ con el navegador Web. Es posible que aparezca una versión Django diferente dependiendo de la versión de Django que ha instalado. Usted verá un "Bienvenido a Django" web como figura4.

Aunque el servidor de desarrollo es extremadamente útil para, bueno, desarrollar, resiste la tentación de usar este servidor en cualquier entorno parecido a producción. El servidor de desarrollo puede manejar fiablemente una sola petición a la vez, y no ha pasado por una auditoría de seguridad de ningún tipo. Cuando sea el momento de lanzar tu sitio, necesita hacer más.

> " Por defecto, el comando runserver inicia el servidor de desarrollo en el puerto 8000, escuchando sólo conexiones locales. Si quieres cambiar el puerto del servidor, pasa este como un argumento de línea de comandos: *python manage.py runserver 8888*
>
> También puedes cambiar las direcciones de IP que escucha el servidor. Esto es utilizado especialmente si quieres compartir el desarrollo de un sitio con otros desarrolladores. Lo siguiente: *python manage.py runserver 0.0.0.0:8888* hará que Django escuche sobre cualquier interfaz de red, permitiendo que los demás equipos puedan conectarse al servidor de desarrollo. "

Para input menos comandos cuando desarrollando el trabajo, para lanzar el servidor, escribo un bash script:

```
#!/bin/bash
python manage.py runserver 0.0.0.0:8888
```

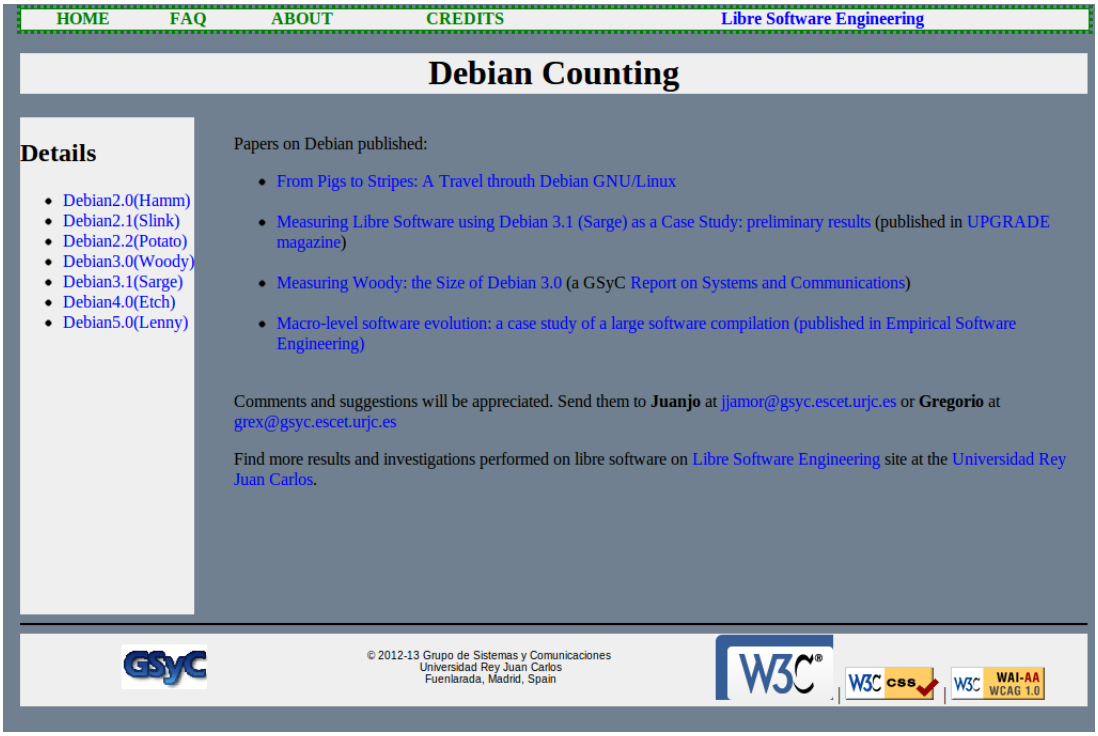


Figura 5: página principal de Debian_Counting

guardado en el fichero runserver.sh, solo ejecutar con el comando: *./runserver.sh* para lanzar el servidor.

4.2. Vista

En este sección, realizaré el función básico de mostrar el página web con Django, usando su modelo Vista de MTV. También va a explicar como procesa una petición Django.

4.2.1. Crear el pagina principal de Debian_Counting

Usando HTML y CSS, he diseñado y realizado el pagina principal de Debian_Counting, como figura5 :

El esqueleto del diseño es como siguiente, el cuerpo de página está dividido en cuatro secciones principales :

```
<html>
    <head>:
        <meta>
        <title>
```

```
        <style type=''text/css''>
    </head>
    <body>
        <div id=all>

        <div id=header>       --- los enlaces de paǵinas:HOME,FAQ, ABOUT,etc
            <div id=page_title>   --- titulo de la página
            <div id=nav_content>  --- navegación bar  y contenido de la página
                <div id=navebar>
                <div id=content>
            <div id=footer>       --- información del web
        </div>
    </body>
</html>
```

El elemento *div* es un elemento de bloque se utiliza para agrupar los elementos HTML. A pesar de que es posible crear diseños bonitos con tablas HTML, tablas fueron diseñadas para presentar datos de tabla - no como una herramienta de *layout*!

CSS se utiliza para colocar los elementos, o para crear fondos o aspecto colorido para las páginas.

El fichero de página se guarda en el nombre *index.html*.

4.2.2. Usar Vista

Dentro del directorio debian_counting que django-admin.py startproject hizo en el sección anterior 4.1.1, cree un archivo vacío llamado *views.py*. Este módulo Python incluirá nuestras funciones para usar. Tenga en cuenta que no hay nada de especial en el nombre *views.py*. Django no importa lo que el archivo se llama, como se verá un poco - pero es una buena idea llamar views.py como una convención, para el beneficio de otros desarrolladores leer el código.

Aquí está toda la función, además de las declaraciones de importación, que se debe escribir en el fichero de *views.py*:

```
from django.http import HttpResponse
from django.shortcuts import render_to_response
def homepage_view(request):

    return render_to_response('index.html',locals())
```

Repasemos el código anterior línea a línea:

- Primero, importamos la clase HttpResponse y render_to_response, las cuales pertenecen al módulo django.http y django.shortcuts. Para ver más detalles de los objetos HttpRequest y HttpRespnonse puedes consular el Django Book 2.0, Apéndice A.

- A continuación, se define una función llamada *homepage_view*. Cada función de vista toma por lo menos un parámetro, llamado *request* por convención. Este es un objeto que contiene información acerca de la solicitud Web actual que ha provocado este punto de vista, y es una instancia de la clase django.http.HttpRequest. En este ejemplo, no hacemos nada con la solicitud, pero debe ser el primer parámetro de la vista, no obstante. Tenga en cuenta que el nombre de la función de vista no importa, no tiene que ser llamado de una manera determinada para que Django la reconozca. Lo llamamos hola aquí, porque ese nombre indica claramente la esencia de la vista, pero podría muy bien llamarse "home_page", o algo igual de repugnante. En la siguiente sección, "Mapeando URLconf", arrojará luz sobre cómo encuentra Django esta función.

- La función es un sencillo de una sola línea: simplemente devuelve un objeto HttpResponse que ha sido creada antes "index.html".

La principal lección es esta: una visión es sólo una función de Python que toma un HttpRequest como primer parámetro y devuelve una instancia de HttpResponse. Para que una función de Python para ser vistas Django, debe hacer estas dos cosas.

4.2.3. Mapeando URLs a Vistas

Si en este punto se ejecuta *./runserver.sh* verá el "Bienvenido a Django", sin ningún rastro de nuestro punto de vista "index.html" en cualquier lugar. Esto se debe a nuestro proyecto debian_counting todavía no sabe acerca de la vista homepage_view, tenemos que decirle a Django explícitamente que estamos activando este punto de vista en una URL determinada. Para conectar una función de vista a una URL concreta con Django, utilice un URLconf.

Una URLconf es como una tabla de contenido para su sitio Web con Django. Básicamente, es un mapeo entre las direcciones URL y las funciones de vista que deben ser llamados por los URLs. Es la forma de decirle a Django: "Por esta URL, llamar a este código, y por esa URL, llame a ese código." Por ejemplo, "Cuando alguien visita el URL / foo /, llame a la función de vista foo_view (), que vive en el Python módulo views.py ".

Abrir el archivo urls.py. De forma predeterminada, se ve algo como esto:

```
from django.conf.urls import patterns, include, url
# Uncomment the next two lines to enable the admin:
# from django.contrib import admin
# admin.autodiscover()
urlpatterns = patterns('',
```

```
    # Examples:
    # url(r'^$', 'mysite.views.home', name='home'),
    # url(r'^mysite/', include('mysite.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:     # #url(r'

    # Uncomment the next line to enable the admin:
    # url(r'^admin/', include(admin.site.urls)),
    )
```

Este URLconf defecto incluye algunas características de Django utilizados comentadas, por lo que la activación de estas características es tan fácil como descomentar las líneas correspondientes. Si ignoramos el código comentado de salida, esto es la esencia de una URLconf:

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('', )
```

Demos un paso a través de este código de una línea a la vez:

- Las primeras importaciones de línea de tres funciones del módulo django.conf.urls.defaults, que es la infraestructura URLconf de Django: patrones, incluyen, y urls.
- La segunda línea llama a los patrones de función y guarda el resultado en una variable llamada urlpatterns. La función de los patrones se pasa un solo argumento - la cadena vacía.

Lo principal a tener en cuenta es la urlpatterns variables, que Django espera encontrar en el módulo URLconf. Esta variable define la correspondencia entre las direcciones URL y el código que se encarga de las direcciones URL. De forma predeterminada, como se puede ver, la URLconf está vacía - la aplicación de Django es una pizarra en blanco. (Como nota al margen, así es como Django sabía que le muestre el "Bienvenido a Django" en el último capítulo. Si su URLconf está vacía, Django asume que acaba de iniciar un nuevo proyecto y, por lo tanto, muestra ese mensaje.)

Para agregar una dirección URL y ver a la URLconf, sólo tiene que añadir una asignación entre un patrón de URL y la función de vista. Aquí es cómo conectar a nuestro *homepage_ view*:

```
from django.conf.urls.defaults import patterns, include, url
from debian_counting import views
urlpatterns = patterns('',

    url(r'^$',views.homepage_view),

)
```

Hemos hecho dos cambios aquí:

- Primero, importamos la vista homepage_view de su módulo - debian_counting / views.py, que se traduce en debian_counting.views en la sintaxis de importación de Python. (Esto supone debian_counting / views.py está en la ruta de Python.)
- A continuación, añadimos la línea url (r '^$', views.homepage_view), a urlpatterns. Esta línea se conoce como un patrón URL. La url () función indica a Django cómo manejar la url que está configurando. El primer argumento es una cadena de patrones (una expresión regular, más información en Apéndices A.1) y el segundo argumento es la función para utilización con ese patrón.

4.2.4. Configurar el settings.py

Si, en este punto, otra vez se ejecutó *./runserver.sh* más, usted todavía ve el "Bienvenido a Django" del mensaje, sin ningún rastro de nuestro punto de vista "index.html" en cualquier lugar. Esto se debe a nuestro proyecto debian_counting todavía no sabe donde está el fichero "index.html", tenemos que decirle a Django explícitamente que donde están los html ficheros.

Abrir el archivo *settings.py* y encontrar el ajuste TEMPLATE_DIRS. De forma predeterminada, es una tupla vacía, probablemente contiene algunos comentarios generados automáticamente:

```
TEMPLATE_DIRS = (

    # Put strings here, like "/home/html/django_templates" or "C:/www/django/template
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.

)
```

Este ajuste le indica mecanismo de carga de plantillas de Django donde buscar plantillas. Elija un directorio en el que desea almacenar las plantillas y añadirlo a TEMPLATE_DIRS, de este modo:

```
TEMPLATE_DIRS = (

    '/home/xxx/django/debian_counting/templates',

)
```

Hay algunas cosas a tener en cuenta:

- Puede especificar cualquier directorio que desee, siempre y cuando el directorio y las plantillas dentro de ese directorio pueden ser leídos por la cuenta de usuario bajo el que se ejecuta el servidor Web. Si usted no puede pensar en un lugar apropiado para poner sus plantillas, se recomienda la creación de un directorio de plantillas dentro de su proyecto (es decir, dentro del directorio misitio que ha creado en 4.1.1.

- Si sus TEMPLATE_DIRS incluye solamente un directorio, no se olvide de la coma al final de la cadena del directorio!

4.2.5. Cómo procesa una petición Django

El servidor se está ejecutando en el http://127.0.0.1:8888/, por lo que abrir un navegador Web y vaya a http://127.0.0.1:8888. Usted debe ver el página "index.html" - la salida de su vista de Django.

Debemos señalar varias cosas en lo que hemos visto. Este es el detalle de lo que sucede cuando ejecutas el servidor de desarrollo de Django y hacemos una petición a una página Web:

Todo comienza con el archivo de configuración *settings.py*. Cuando se ejecuta python manage.py runserver, el script busca un archivo llamado settings.py en el directorio debian_counting interior. Este archivo contiene todas las clases de configuración de este proyecto de Django en particular, todo en mayúsculas: TEMPLATE_DIRS, BASES DE DATOS, etc El ajuste más importante se llama ROOT_URLCONF. ROOT_URLCONF le dice a Django qué módulo Python debe utilizarse como la URLconf de este sitio Web.

Recuerda cuando django-admin.py startproject creado los archivos settings.py y urls.py, el settings.py autogenerado contiene una configuración ROOT_URLCONF que apunta a la autogenerado urls.py. Abra el archivo settings.py y ver por sí mismo, sino que debe tener este aspecto:

```
ROOT_URLCONF = 'debian_counting.urls'
```

Esto se corresponde con el archivo de *debian_ counting / urls.py*.

Cuando llega una petición para una URL en particular - por ejemplo, una solicitud de / hola / - Django carga la URLconf apuntada por el ajuste ROOT_URLCONF. A continuación, se comprueba cada uno de los URLpatterns en que URLconf, en orden, la comparación de la dirección URL solicitada con los patrones de uno en uno, hasta que encuentra uno que coincida. Cuando encuentra uno que coincida, se llama a la función vista asociada a ese patrón, pasándole un objeto HttpRequest como primer parámetro. (Vamos a cubrir los detalles de HttpRequest más tarde.)

Como vimos en nuestro primer ejemplo de vista, una función de vista debe devolver un HttpResponse. Una vez que se hace esto, Django hace el resto, la conversión del objeto de Python a una respuesta web adecuada con las cabeceras HTTP apropiadas y el cuerpo (es decir, el contenido de la página web).

En resumen:

1. Cuando lanzado el servidor de Django,Django determina la configuración según el fichero settings.py.

2. A petición llega desde navegador.

3. Django se ve en todos los URLpatterns en la URLconf (fichero urls.py)para el primero que coincida con la petición .

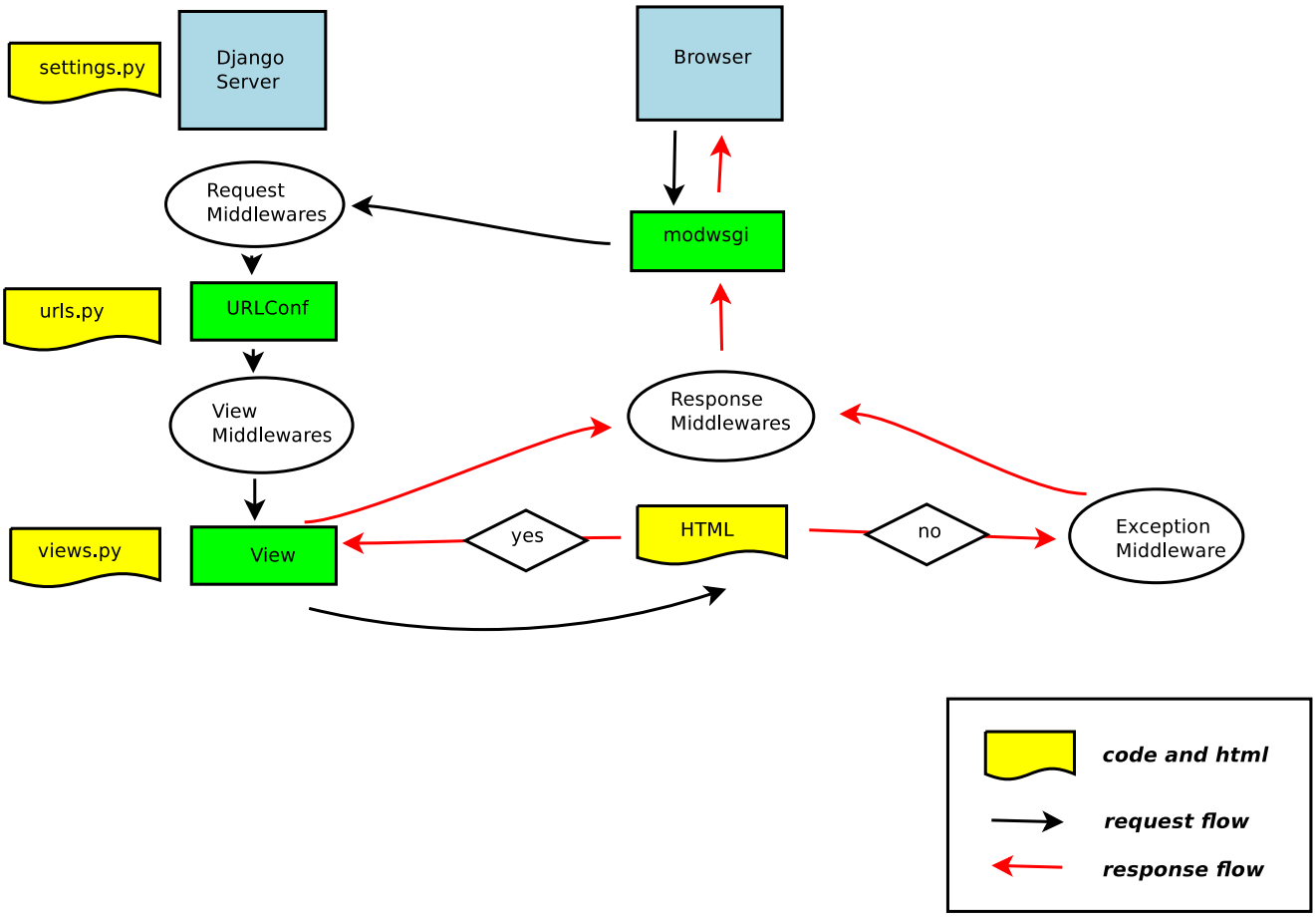


Figura 6: Proceso de vista

4. Si se encuentra una coincidencia, se llama a la función vista asociada.

5. La función de vista(en el fichero views.py) devuelve una HttpResponse.

6. Django convierte la HttpResponse a la respuesta HTTP adecuada, lo que se traduce en una página Web.

Ahora ya sabe los fundamentos de cómo hacer que las páginas con Django. Es muy sencillo, en realidad - simplemente escriba funciones de vista y asignar direcciones URL a través URLconfs. Tenemos figura 6 para explicar esto proceso:

4.3. Template (plantilla)

Una plantilla de Django es una cadena de texto que se pretende separar la presentación de un documento a partir de sus datos. Una plantilla define los marcadores de posición y varios trozos de lógica básica (etiquetas de plantilla) que regulan cómo se debe mostrar el documento. Por lo general, las plantillas se utilizan para producir HTML, pero las

plantillas de Django son igualmente capaces de generar cualquier formato basado en texto.

4.3.1. Variable, Filtro y Etiqueta

Vamos a explicar con una plantilla de ejemplo sencillo[6]. Esta plantilla de Django describe una página HTML que gracias a una persona para hacer un pedido a una empresa. Piense en ello como una carta modelo:

```
<html>
<head><title>Ordering notice</title></head>
<body>
<h1>Ordering notice</h1>
<p>Dear {{ person_name }},</p>
<p>Thanks for placing an order from {{ company }}.
It's scheduled to ship on {{ ship_date|date:"F j, Y" }}.</p>
<p>Here are the items you've ordered:</p>
<ul>
{% for item in item_list %}
    <li>{{ item }}</li>
{% endfor %}
</ul>
{% if ordered_warranty %}
    <p>Your warranty information will be included in the packaging.</p>
{% else %}
    <p>You didn't order a warranty, so you're on your own when
    the products inevitably stop working.</p>
{% endif %}
<p>Sincerely,<br />{{ company }}</p>
</body>
</html>
```

Esta plantilla es básica en HTML con algunas variables y etiquetas de plantilla arrojados pulg Repasemos lo siguiente:

- Cualquier texto rodeado por un par de llaves (por ejemplo, {{person_name}}) es una variable. Esto significa "introducir el valor de la variable con el nombre dado." (¿Cómo especificar los valores de las variables? Ya llegaremos a eso en un momento.)

- el segundo párrafo de esta plantilla contiene un ejemplo de un filtro, que es la forma más conveniente para alterar el formato de una variable. En este ejemplo, {{ship_date | fecha: "F j, Y"}}, estamos pasando la variable ship_date al filtro de fecha, indicando la fecha filtrar el argumento de la "F j, Y". La fecha de formatos

[6] el ejemplo es de Django Book

filtro fechas en un formato determinado, según lo especificado por el argumento. Los filtros se unen utilizando un carácter de barra vertical (|), como una referencia a las tuberías de Unix.

- Finalmente, cualquier texto que está rodeado por llaves y signos de porcentaje (por ejemplo, {% if ordered_warranty %}) es una etiqueta de plantilla. La definición de una etiqueta es muy amplio: una etiqueta sólo le dice el sistema de plantillas de "hacer algo." Esta plantilla contiene un ejemplo de etiqueta for ({% for item in item_list %}) y una etiqueta if ({% if ordered_warranty %}).

Cada plantilla de Django tiene acceso a varias variables incorporadas y filtros, en nuestro trabajo, los más usados son: {{ variables }}, {% for item in item_list %}.

La etiqueta {% for %} le permite iterar sobre cada elemento de una secuencia. Al igual que en Python para la declaración, la sintaxis es para X en Y, donde Y es la secuencia de bucle de una y X es el nombre de la variable a utilizar para un ciclo particular del bucle. Cada vez en el ciclo, el sistema de plantillas hará que todo entre {% for %} y {% endfor %}.

Dentro de cada {% for %} bucle, tendrá acceso a una variable llamada forloop. Esta variable tiene unos atributos que le dan información sobre el estado del bucle:

- forloop.counter siempre se establece en un entero que representa el número de veces que el ciclo se ha introducido. Este es uno indexada, por lo que la primera vez a través del bucle, forloop.counter se establece en 1. aquí un ejemplo:

```
{% for item in todo_list %}
    <p>{{ forloop.counter }}: {{ item }}</p>
{% endfor %}
```

- forloop.counter0 es como forloop.counter, excepto que es indexada a cero. Su valor se establece en 0, la primera vez en el ciclo.

4.3.2. Herencia de Plantilla

Al utilizar el sistema de plantillas de Django para crear páginas HTML enteras, lo cual lleva a un problema de desarrollo Web común. Través de un sitio Web, ¿cómo reducir la duplicación y la redundancia de áreas de página comunes, tales como la navegación en todo el sitio?

Una forma clásica de resolver este problema es utilizar las inclusiones de servidor, las directivas se pueden incrustar en sus páginas HTML de "incluir" una página Web dentro de otra. De hecho, Django admite este enfoque, con el {% include %} etiqueta de plantilla que acabamos de describir.

Pero la mejor forma de resolver este problema con Django es usar una estrategia más elegante llamada herencia de plantillas.

Consideramos nosotros debian_counting web , tenemos cuatro páginas en figura 7.

index.html Página inicial de el web

Figura 7: comparación de webs

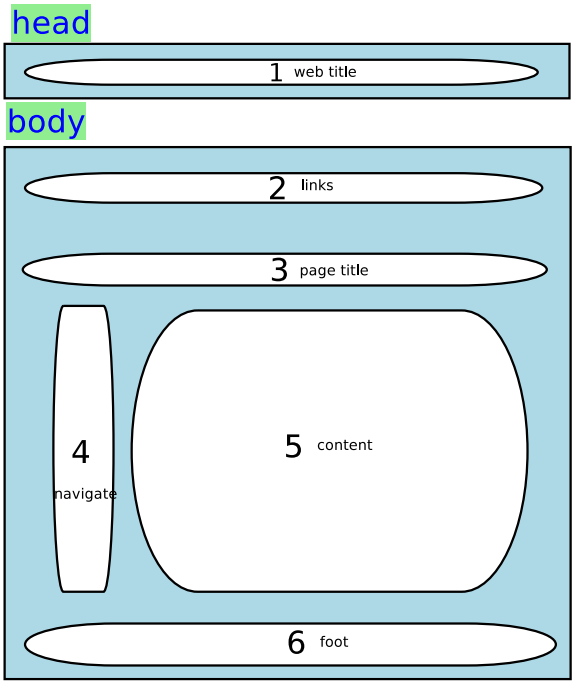


Figura 8: arquitectura de la web

faq.html Información de permite profundizar en los objetivos de SLOCCount y la investigación que se hace en el campo de Software Libre.

about.html Por lo general usted encontrará aquí las respuestas que surgen cuando nos fijamos en COCOMO y los números y cifras que da. Echa un vistazo antes de ponerse en contacto con los encargados del sitio.

credits.html Gente detrás de este sitio y el proyecto SLOCCount.

Las páginas tienen común arquitectura como en figura 8:

Entre ellos , hay áreas comunes como número: 2, 4, 6, las diferentes son 1,3,5. Normalmente, necesitas repetir todos trabajos para cada página.

Sistema de herencia de plantillas de Django resuelve estos problemas, se definen los fragmentos que son diferentes. El primer paso es definir una plantilla de base - el esqueleto de la página que las plantillas hijas serán posteriormente herencia. Aquí hay una plantilla base para nuestro web actual:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title> {% block web_title%} 1 Debian Counting {% endblock%} </title>
    <link rel="stylesheet" type="text/css" href="">
    <style type="text/css">     </style>
</head>
<body bgcolor="SlateGray">
<div id="all">
    <div id="header">
    {% block headerlink%}             2 links          {% endblock%}
    </div>  <!-- end header -->

    <div id="page_title">
    <h1 align="center">{%block page_title%}  3 Debian Counting{% endblock%}</h1>
    <div id="nav_content">
        <div id="navebar" style="float:left">
        {% block navebar%}           4 navigate        {% endblock%}
        </div> <!-- end navebar -->

        <div id="content" style="float:right">
        {% block content%}        5 content          {% endblock%}
        </div> <!-- end content -->

        <div style="clear:both"></div>
        </div> <!-- end nav_content -->

    <hr color="negro" />  <!--  add a  black line for the web -->

    <div id="footer">
    {% block footer%}            6 foot          {% endblock%}
    </div>  <!-- end footer -->
</div>  <!-- end all -->
</body>
</html>
```

Esta plantilla, que llamaremos base_all.html, define un documento esqueleto HTML simple que usaremos para todas las páginas del sitio. Es el trabajo de las plantillas hijas para reemplazar o añadir o dejar solo el contenido de los bloques.

Estamos usando una etiqueta de plantilla : el {% block %} {% endblock %}. Todos los {% block %} etiquetas decirle al motor de plantillas que una plantilla hija puede sustituir aquellas partes de la plantilla.

Ahora que tenemos esta plantilla base, podemos modificar nuestras htmls (index.html, faq.html, about.html, credits.html):

index.html

```
{% extends "base_all.html"%}
{% block web_title%} Debian Counting {% endblock%}
{% block page_title%} Debian Counting {% endblock%}
{% block content%} ... {% endblock%}
```

faq.html

```
{% extends "base_all.html"%}
{% block web_title%} FAQ {% endblock%}
{% block page_title%} Frequently Asked Questions {% endblock%}
{% block content%} ... {% endblock%}
```

about.html

```
{% extends "base_all.html"%}
{% block web_title%} About SLOCCount {% endblock%}
{% block page_title%} About SLOCCount {% endblock%}
{% block content%} ... {% endblock%}
```

credits.html

```
{% extends "base_all.html"%}
{% block web_title%} Credits {% endblock%}
{% block page_title%} Credits {% endblock%}
{% block content%} ... {% endblock%}
```

Cada plantilla contiene sólo el código que es único en la plantilla. No se necesita redundancia. Si usted necesita hacer un cambio de diseño en todo el sitio, así que el cambio de *base_ all.html*, y todas las otras plantillas reflejarán el cambio inmediatamente.

Aquí explica cómo funciona. Cuando se carga el index.html plantilla, el motor de plantillas ve la { % extends %} etiqueta, señalando que esta plantilla es una plantilla hija. El motor de carga inmediatamente la plantilla padre - en este caso, base_all.html.

En ese momento, el motor de la plantilla da cuenta de los tres { % block %} etiquetas en base_all.html y sustituye a los bloques con el contenido de la plantilla de niño. Por lo tanto, el título que hemos definido en { % block %} del título se utilizará, así como la { % block content %}.

Tenga en cuenta que desde la plantilla hija no define el bloque{ % block headerlink %}, { % block navebar %}, { % block footer %} el sistema de plantillas utiliza el valor de la plantilla padre en su lugar. Contenido dentro de un { % block % }etiqueta en una plantilla padre siempre se utiliza como reserva.

Herencia no afecta a la contexto de la plantilla. En otras palabras, toda la plantilla en el árbol de herencia tendrá acceso a cada una de sus variables de plantilla a partir del contexto.

Aquí hay algunas pautas para trabajar con herencia de plantillas:

- Si utiliza {% extends %} en una plantilla, que debe ser la primera etiqueta de plantilla en esa plantilla. De lo contrario, la herencia de plantillas no funcionará.
- En general, mientras más {% block %} etiquetas en tus plantillas base, mejor. Recuerde, las plantillas hijas no tienen que definir todos los bloques de los padres, para que pueda rellenar valores por defecto razonables en una serie de bloques, y luego definen sólo los que necesita en las plantillas hijas. Es mejor tener más que menos anzuelos anzuelos.
- Usted no puede definir múltiples {% block %} etiquetas con el mismo nombre en la misma plantilla. Esta limitación existe debido a una etiqueta de bloque trabaja en "ambas direcciones". Es decir, una etiqueta de bloque no sólo proporciona un agujero para llenar, sino que también define el contenido que llena el agujero en el padre. Si hay dos nombres similares {% block %} etiquetas en una plantilla, los padres de esa plantilla no se sabe cuál de los contenidos de los bloques a usar.

4.3.3. Uso en trabajo

Para hacer funcionar las plantillas en el web, es necesario seguir para completar los pasos siguientes:

- Dentro de el carpeta debian_counting/debian_counting/,creado en el sección 4.1.1, crear una nueva carpeta se llama *templates,* y pone los archivos de páginas: base_all.html, index.html, faq.html,etc dentro esto carteta.
- Modificar URLs, en el fichero de urls.py y views.py, agregue el siguiente códigos:

```
urls.py
.....................................
url(r'^faq.html/$',views.faq_view),
url(r'^about.html/$',views.about_view),
url(r'^credits.html/$',views.credits_view),
views.py
.....................................
def faq_view(request):
    return render_to_response('faq.html',locals())
def about_view(request):
    return render_to_response('about.html',locals())
def credits_view(request):
    return render_to_response('credits.html',locals())
```

- Modificar el fichero settings.py , Abrir el archivo *settings.py* y encontrar el ajuste TEMPLATE_DIRS. De forma predeterminada, es una tupla vacía, probablemente contiene algunos comentarios generados automáticamente:

```
TEMPLATE_DIRS = (
```

```
    # Put strings here, like "/home/html/django_templates" or "C:/www/django/template
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
)
```

Este ajuste le indica mecanismo de carga de plantillas de Django donde buscar plantillas. Elija un directorio en el que desea almacenar las plantillas y añadirlo a TEMPLATE_DIRS, de este modo:

```
TEMPLATE_DIRS = (
    '/home/xxx/django/debian_counting/templates',
)
```

Hay algunas cosas a tener en cuenta:

- Puede especificar cualquier directorio que desee, siempre y cuando el directorio y las plantillas dentro de ese directorio pueden ser leídos por la cuenta de usuario bajo el que se ejecuta el servidor Web. Si usted no puede pensar en un lugar apropiado para poner sus plantillas, se recomienda la creación de un directorio de plantillas dentro de su proyecto (es decir, dentro del directorio misitio que ha creado en 4.1.1.
- Si sus TEMPLATE_DIRS incluye solamente un directorio, no se olvide de la coma al final de la cadena del directorio!

Hasta aquí, hemos utilizado el Template y Vista de MTV de Django, la funciona del web ahora está como figura 9:

4.4. Modelo

En las aplicaciones Web modernas, la lógica arbitraria a menudo implica la interacción con una base de datos. Detrás de las escenas, un sitio Web impulsado por bases de datos se conecta a un servidor de base de datos, recupera algunos datos de esta, y los muestra con un formato agradable en una página Web. El sitio también puede proporcionar la funcionalidad que permita los visitantes del sitio poblar la base de datos por su propia cuenta.

Django es apropiado para crear sitios web que manejen una base de datos, ya que incluye una manera fácil pero poderosa de realizar consultas a bases de datos utilizando Python. Este sección explica esta funcionalidad: la capa de la base de datos de Django. Tenemos una lista de copias de seguridad de base de datos de cada versión de Debian usando en el Debian_Counting.

- hamm.dump
- slink.dump

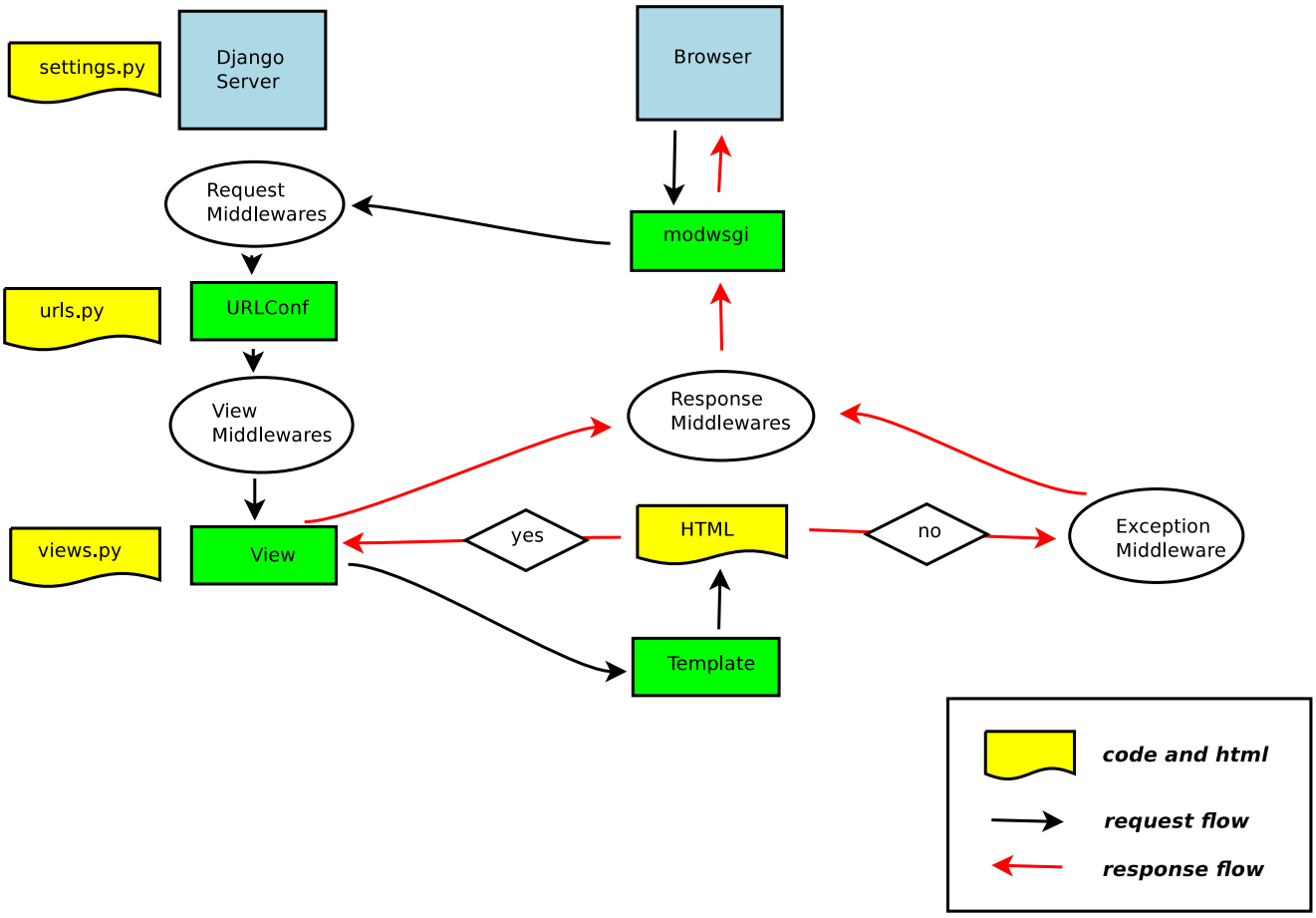


Figura 9: Proceso de Template(plantilla) y Vista

- potato.dump
- woody.dump
- sarge.dump
- etch.dump
- lenny.dump

4.4.1. Parte de MySQL, importa los datos a MySQL

Es necesario conocer teoría básica de bases de datos y SQL para usar la capa de base de datos de Django. En siguientes, va a explicar como crear usuarios y bases de datos, y importar datos existe a MySQL, etc.

crear nueva usuario entra el MySQL con la cuenta de 'root', y ejecuta los siguientes comandos:

```
xxx:~$ mysql -uroot -p
Enter password:
mysql> insert into mysql.user(Host,User,Password) values('localhost','xmlwolf',passwo
mysql> flush privileges;
```

crear nueva base de datos

```
mysql> create database HAMM;
Query OK, 1 row affected (0.01 sec)
```

autorizar base de datos al usuario

```
mysql> grant all privileges on HAMM.* to xmlwolf;
Query OK, 0 rows affected (0.00 sec)
```

importar datos

```
xxx:~$ mysql -u root -p HAMM<tfm/debcounting_database/hamm.dump
Enter password:
```

Ahora ,pudes entrar con el cuenta 'xmlwolf' , a ver que hay en el database de 'HAMM';

```
meilin@Master:~$ mysql -uxmlwolf -p
Enter password:
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
```

```
| information_schema |
| HAMM               |
| test               |
+--------------------+
3 rows in set (0.00 sec)
mysql> use HAMM;

mysql> show tables;
+----------------+ |
Tables_in_HAMM |
+----------------+
| files          |
| packages       |
+----------------+
2 rows in set (0.00 sec)
```

Puedes consultar más información de MySQL en apéndice A.2.

4.4.2. Configuración de la Base de Datos

Hemos configurado y activado un servidor de base de datos de MySQL en sección 3.3.1, también ha creado una base de datos en su interior se llama "HAMM" en sección 4.4.1.

Ahora, tenemos que tener cuidado de algunas tareas de configuración inicial, tenemos que decirle a Django qué servidor de base de datos a usar y cómo conectarse a ella.

La configuración de base de datos se encuentra en el archivo de configuración de Django, llamado settings.py de forma predeterminada. Edita este archivo y busque la configuración de base de datos:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.',
        'NAME': '',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    },
}
```

Aquí hay un resumen de cada ajuste.

ENGINE le indica a Django qué base de datos a utilizar. Si está utilizando una base de datos con Django, ENGINE debe establecerse en una de las cadenas que se muestran en la Tabla 2:

Configuración	Base de datos	Adaptador requerido
django.db.backends.postgresql_psycopg2	PostgreSQL	psycopg version 2.x
django.db.backends.mysql	MySQL	MySQLdb
django.db.backends.sqlite3	SQLite	No necesita adaptador
django.db.backends.oracle	Oracle	cx_Oracle

Cuadro 2: Configuración de ENGINE de base de datos

NAME le dice a Django el nombre de su base de datos. Por ejemplo:

```
'NAME': 'HAMM',
```

User usuario, le dice a Django qué nombre de usuario se utiliza para conectarse a la base de datos. Por ejemplo: estamos utilizando MySQL,

```
'USER': 'xmlwolf',
```

PASSWORD CONTRASEÑA, le dice a Django qué contraseña se utiliza para conectarse a la base de datos. Por ejemplo: estamos utilizando MySQL,

```
'PASSWORD': '1985xumeilin',
```

HOST le dice a Django que albergan a usar cuando se conecta a la base de datos. Si su base de datos está en el mismo equipo que la instalación de Django (es decir, localhost), déjelo en blanco.

PORT PUERTO, le dice a Django qué puerto se utiliza para conectarse a la base de datos. Si estás utilizando SQLite, deja este espacio en blanco. De lo contrario, si se deja en blanco, el adaptador de base de datos subyacente usará el puerto por defecto para el servidor de base de datos dada. En la mayoría de los casos, el puerto por defecto está muy bien, así que usted puede dejarlo en blanco.

Una vez que haya introducido los ajustes y guardado *settings.py*, es una buena idea para probar la configuración. Para ello, ejecute *python manage.py shell* [7], dentro del directorio del proyecto debian_counting.

```
>>> from django.db import connection
>>> cursor = connection.cursor()
```

Si no ocurre nada, entonces su base de datos está configurado correctamente.

Como tenemos más que una base de dato, para interactuar con múltiples bases de datos, tendrá que tomar algunas medidas adicionales[22], al final, configuramos el *settings.py* como abajo, deja el "default" vacío :

[7] manage.py shell is a way to run the Python interpreter with the correct Django settings activated. This is necessary in our case, because Django needs to know which settings file to use in order to get your database connection information.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.',
        'NAME': '',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
    },
    'hamm': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'HAMM',
        'USER': 'xmlwolf',
        'PASSWORD': '1985xumeilin',
        'HOST': '',
        'PORT': '',
    },
}
```

4.4.3. Crear la aplicación

Ahora que verificamos que la conexión está funcionando, es hora de crear una Aplicación de Django – una colección de archivos de código fuente, incluyendo modelos y vistas, que conviven en un solo paquete de Python y representen una aplicación completa de Django.

Vale la pena explicar la terminología aquí, porque esto es algo que suele hacer tropezar a los principiantes. Ya hemos creado un proyecto, en el sección 4.1.1, entonces, ¿cuál es la diferencia entre un proyecto y una aplicación ? La diferencia es la que existe entre la conguración y el código:

- Un proyecto es una instancia de un cierto conjunto de aplicaciones de Django, más las conguraciones de esas aplicaciones. Técnicamente, el único requerimiento de un proyecto es que este suministre un archivo de congu- ración, el cual dene la información hacia la conexión a la base de datos, la lista de las aplicaciones instaladas, la variable TEMPLATE_DIRS, y así sucesivamente.

- Una aplicación es un conjunto portable de una funcionalidad de Django, típicamente incluye modelos y vistas, que conviven en un solo paquete de Python. Por ejemplo, Django incluye un número de aplicaciones, tales como un sistema de comentarios y una interfaz de administración automática. Una cosa clave para notar sobre estas aplicaciones es que son portables y reusables en múltiples proyectos.

Si estás usando la capa de base de datos de Django (modelos), debes crear una aplicación de Django. Los modelos deben vivir dentro de aplicaciones. En nuestro trabajo, necesitamos crear total 7 apps para los diferentes versiones de Debian. Explicamos con el modelo de *hamm*.

Dentro del directorio del proyecto debian_counting, escriba este comando para crear una aplicación de los libros:

```
python manage.py startapp hamm_model
```

Este comando no produce ningún resultado, pero sí crear un directorio de *hamm_model* dentro del directorio debian_counting. Echemos un vistazo a los contenidos de ese directorio:

```
hamm_model/
    __init__.py
    models.py
    tests.py
    views.py
```

Estos archivos contendrán los modelos y puntos de vista para esta aplicación.

Echa un vistazo a models.py y views.py en su editor de texto favorito. Ambos archivos están vacíos, excepto por los comentarios y la importación en models.py. Esta es la página en blanco para su aplicación Django.

```
from django.db import models
# Create your models here.
```

Vamos a modificar el models.py en el siguiente sección 4.4.4.

Para ello, el primer paso es activar estos modelos en nuestro proyecto Django. Hacemos esto mediante la adición de la aplicación de los libros de la lista de "las aplicaciones instaladas" en el archivo de configuración.

Edite el archivo settings.py , y buscar la configuración INSTALLED_APPS. INSTALLED_APPS le dice a Django qué aplicaciones están activadas para un proyecto determinado. De forma predeterminada, se ve algo como esto:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
)
```

Añadir “hamm_model” a la lista INSTALLED_APPS, por lo que el ajuste termina pareciéndose a esto:

```
INSTALLED_APPS = (
    'django.contrib.auth',
```

```
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.sites',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'debian_counting.hamm_model',
    )
```

'debian_counting.hamm_model' se refiere a la aplicación de los libros que estamos trabajando. Cada aplicación en INSTALLED_APPS está representada por su ruta completa Python - es decir, el camino de paquetes, separados por puntos, lo que lleva al paquete de aplicación.

Ahora que la aplicación de Django se ha activado en el archivo de configuración, podemos crear las tablas de base de datos en nuestra base de datos.

4.4.4. Integración con bases de datos heredadas y Aplicaciones

Django es el más adecuado para la llamada creación de polos industriales - es decir, a partir de proyectos desde cero, como si estuviera construyendo un edificio en un nuevo campo de hierba verde. Pero a pesar de que Django favorece proyectos partir de cero, es posible integrar el marco en bases de datos y aplicaciones heredadas.

La capa de base de datos de Django genera esquemas SQL desde código Python – pero con una base de datos existente, tú ya tienes los esquemas SQL. En tal caso, necesitas crear modelos para tus tablas de la base de datos existente. Para este propósito, Django incluye una herramienta que puede generar el código del modelo leyendo el diseño de las tablas de la base de datos. Esta herramienta se llama inspectdb, y puedes llamarla ejecutando el comando manage.py inspectdb.

- Ejecute el comando python proyecto *manage.py inspectdb database=hamm*. Esto examinar las tablas de la base de datos DATABASE_NAME e imprimir el modelo de clase generado para cada tabla. Echa un vistazo a la salida para tener una idea de lo que inspectdb puede hacer.
- Guarde el resultado en el archivo models.py dentro de la aplicación mediante el uso de redirección de la salida estándar de la shell:

```
python manage.py inspectdb --database=slink > debian_counting/hamm_model/models.py
```

- Edite el proyecto models.py para limpiar los modelos generados y hacer las personalizaciones necesarias.

A final, tenemos el models.py como abjao:

```
from __future__ import unicode_literals
from django.db import models
```

```
class Files(models.Model):
    id_field = models.BigIntegerField(db_column='id')
    file_field = models.CharField(db_column='file',max_length=255L)
    language = models.CharField(max_length=16L)
    sloc = models.IntegerField()

    def __unicode__(self):
        return self.language

    class Meta:
        db_table = 'files'
class Packages(models.Model):
    id_field = models.BigIntegerField(db_column='id')
    name = models.CharField(max_length=64L)
    version = models.CharField(max_length=32L)
    slocs = models.BigIntegerField()
    files = models.IntegerField()

    def __unicode__(self):
        return self.name

    class Meta:
        db_table = 'packages'
```

Ya tenemos un modelo de hamm, Django proporciona automáticamente una API Python de alto nivel para trabajar con esos modelos. Pruébelo ejecutando *python manage.py shell* y escriba lo siguiente por ejemplo:

```
>>> from debian_counting.hamm_model.models import Packages
>>> n = Packages.objects.using('hamm').count()
>>> n
1096
```

Ya el modelo de hamm funciona bien, podemos sacar información desde MySQL usando APIs de Django.

4.5. MTV

Para crear un aplicación entero de hamm, necesitamos integrar la capa de Template (plantilla) y Vista explicando en sección 4.2 y 4.3 con el capa de Modelo realizado en sección 4.4.

Modelo Como describe en sección 4.4.4.

Template (plantilla): En nuestro debian_counting, para cada versión de Debian, tenemos cuatro enlaces principal y nua serarch box para demostrar los informaciones, para cada enlace una plantilla .

Index hamm.html

Statistics statistics.html

Packages packages.html, también hay otro dos plantillas: packages_count.html y packages_file.html ,para información detallada de cada paquete.

Graphs graphs.html

Searchbox search_results.html

Puede consultar más detalles en los fuentes de códigos. Y se necesita modificar el fichero *settings.py* añadir el path de template de hamm a el tumple de TEMPLATE_DIRS:

```
TEMPLATE_DIRS = (
    '/home/meilin/djcode/debian_counting/debian_counting/templates',
    '/home/meilin/djcode/debian_counting/debian_counting/templates/hamm',
)
```

Vista En el capa de Vista , necesitamos crear un fichero "hamm_model_views.py" en el carpeta *hamm_model* creado en sección 4.4.3 y modificar el urls.py para configurar URLConf.

hamm_model_views.py define los funciones de vista que usando el administrador objects a buscar datos de modelo, tenemos esqueleto del esto archivo abajo, consulta más detalles en el fuente de código.

```
# Create your views here.
from __future__ import division # 7/3 = 2.3333
from django.http import HttpResponse
from django.shortcuts import render_to_response
from debian_counting.hamm_model.models import Packages
from debian_counting.hamm_model.models import Files
...
def hamm_view(reques,part): ...
def hamm_search(request): ...
def hamm_database_view(request,part,part2): ...
```

urls.py

```
from debian_counting.hamm_model import hamm_model_views
...
url(r'^hamm/(.*)/(.*)/$',hamm_model_views.hamm_database_view),
url(r'^hamm/search/$',hamm_model_views.hamm_search),
url(r'^hamm/(.*)/$',hamm_model_views.hamm_view),
```

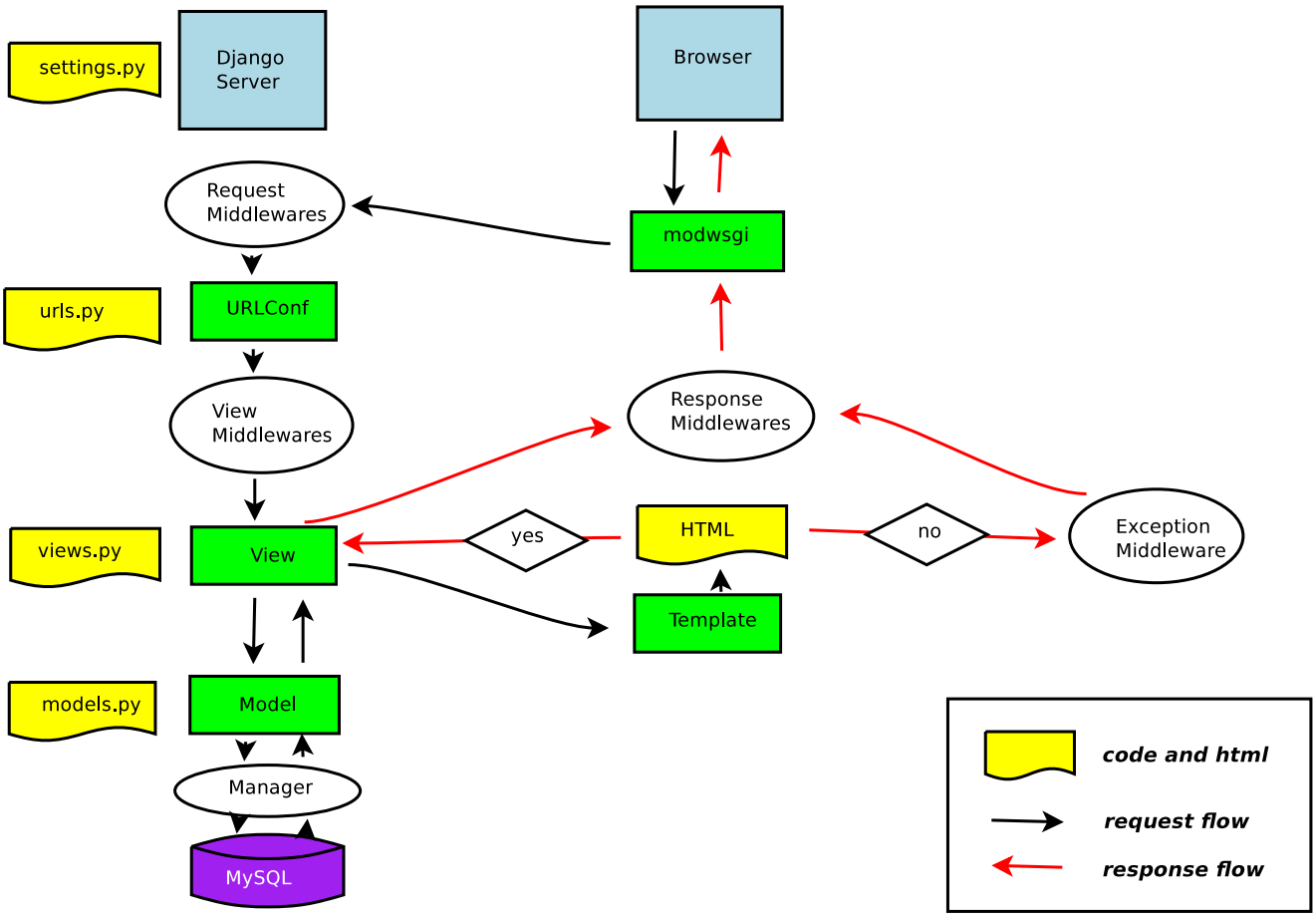
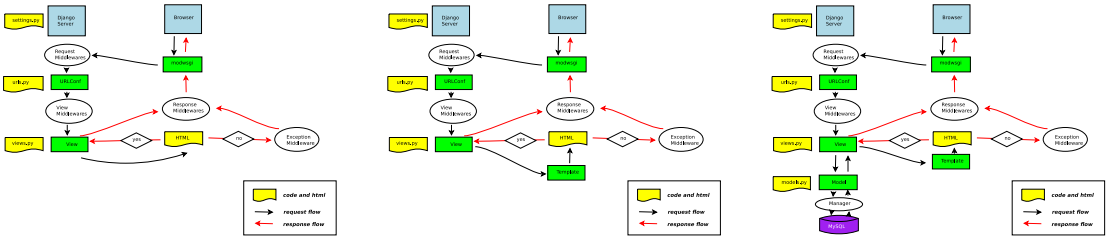


Figura 10: Proceso MTV

Cuando todo está bien configurado, tenemos un web entero con el modelo de MTV. Tengo figura 10 para explicar todo el proceso de MTV:



(a) Vista (b) Template,Vista (c) Modelo,Template,Vista

Figura 11: Proceso del desarrollo

5. Conclusiones

Este capítulo presenta las conclusiones obtenidas tras el desarrollo de este, comentando de manera escueta los puntos relevantes que han permitido el desarrollo del trabajo, y las principales dificultades encontradas. También se comentarán posibles trabajos futuros en pos de la mejora y ampliación del comportamiento aquí descrito.

5.1. Conclusiones

Como conclusión principal extraemos, que se ha logrado la implementación,del un sitio web impulsado por una base de datos, con Web Framework Django y su servidor interno. Dicho sitio web, ha separado a siete aplicaciones por cada versión de Debian (2.0 Hamm, 2.1 Slink, 2.2 Potato, 3.0 Woody, 3.1 Sarge, 4.0 Etch, 5.0 Lenny).

Por cada aplicación, integrando con bases de datos heredadas de sitio web de Debian Counting de Libre Software Engineering de Universidad de Rey Juan Carlos (http://debian-counting.libresoft.es), hay dos funciones principales:

1. Crear páginas web dinámicas según los datos sacando de un base de datos;
2. Un Forma-Handling que puede buscar los nombres similares de los paquetes del base de datos, y demostrar los resultados en forma de enlaces.

Gracias a el patrón de desarrollo MTV de Django, he realizado el sitio web paso a paso con el capa de Vista, Template (plantilla), y Modelo,como en figura 11. También, podemos organizar los archivos de el proyecto metódicamente y claramente en diferentes carpetas según los capas. Al final, tenemos la arquitectura de total el sitio web como figura 12:

5.2. 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:

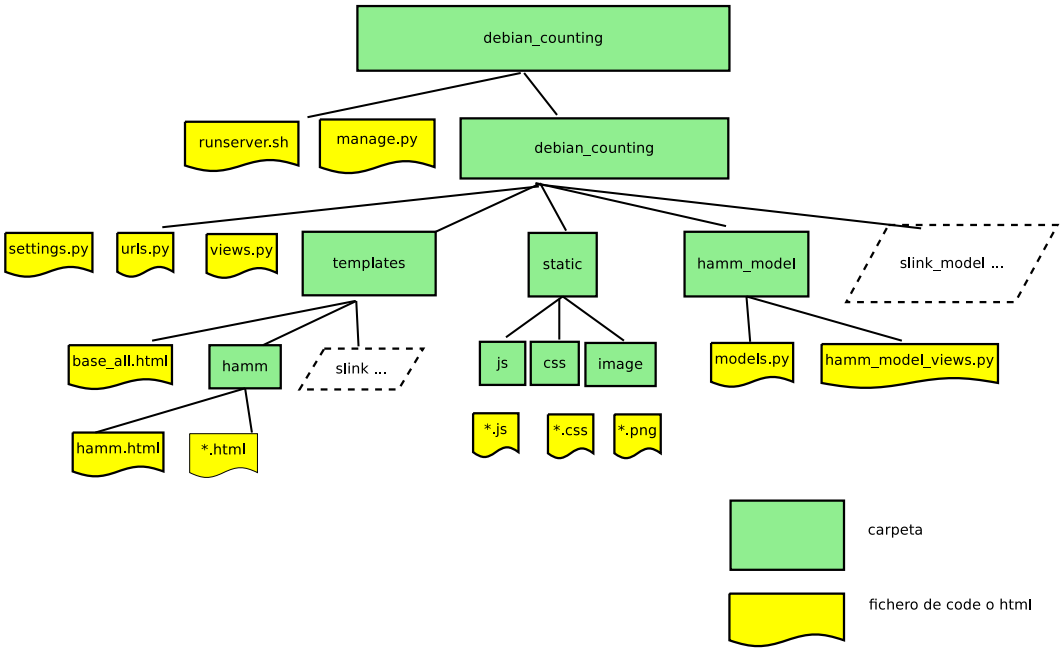


Figura 12: Arquitectura Debian_Counting

- Las tecnologías de desarrollo front-end web como: HTML, CSS, JavaScript.
- La practica en el lenguaje Python, y con él en la programación orientada a objetos. En especial, se adquirió cierta destreza en el manejo de expresiones regulares y lista de objetivos.
- El aprendizaje de los comandos básicos para poner en marcha y configurar un servidor de MySQL, además del importar los datos existe a MySQL y comandos de consulta de MySQL.
- La comparación de los diferentes web framework de distinto lenguajes(ASP.NET, Java, Ruby, Perl, PHP, Python) y patrón de diseño MVC (Modelo, Vista, Controlador).
- El principal objetivo, aprendo utilizar el web framework Django, con su patrón de diseño MTV(Modelo, Template, Vista), crear un sitio web impulsado por un base de datos de MySQL.
- Al final, se ha aprendido también a manejar LyX, el procesador de documentos de "WYSIWYM[8]" empleado en la realización de esta memoria y de otros tantos documentos en el futuro, con total seguridad.

[8]What You See Is What You Mean

5.3. Trabajos futuros

Una vez finalizado el proyecto, se plantean algunos campos de trabajo que supondrían una extensión de su funcionalidad y contribuirían a mejor el sitio web.

HTML5 HTML5 (HyperText Markup Language, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML. Podemos utilizar más características de HTML5 para hacer que el sitio web se vea mejor.

Implementación de Django estamos usando la servidor interno de Django lo que hace las cosas muy fácil - con la ejecución del *runserver*, usted no tiene que preocuparse acerca de la configuración del servidor Web. Pero runserver está destinado sólo para el desarrollo en el equipo local, no para la exposición en la web pública. Para implementar la aplicación Django, tendrás que conectarlo a un servidor Web de potencia industrial como Apache.

Seguridad En nuestro trabajo, no consideramos mucho sobre la seguridad. El Internet no es seguro. En estos días, nuevos problemas de seguridad se aparece cada día. Hemos visto virus que se propagan a una velocidad sorprendente, enjambres de ordenadores comprometidos esgrimidos como armas.Como desarrolladores de web, tenemos el deber de hacer todo lo posible para luchar contra estas fuerzas de la oscuridad. Cada desarrollador Web tiene que considerar la seguridad como un aspecto fundamental de la programación Web.

A. algunos detalles

A.1. Expresión regular

Las Expresiones Regulares (o regexes) son la forma compacta de especificar patrones en un texto. Aunque las URLconfs de Django permiten el uso de regexes arbitrarias para tener un potente sistema de definición de URLs, probalemente en la prácrica no utilices más que un par de patrones regex. Esta es una pequeña selección de patrones comunes:

Símbolo	Coincide con
.	Cualquier carácter
\d	Cualquier dígito
[A-Z]	Cualquier carácter,A-Z mayúsculas
[a-z]	Cualquier carácter,a-z minúsculas
[A-Z a-z]	Cualquier carácter,A-Z o a-z
+	Una o más ocurrencias de la expresión anterior
[^/]+	Todos los caracteres excepto la barra
*	Cero o más ocurrencias de la expresión anterior
{1,3}	Entre una y tres (inclusive) ocurrencias de la expresión anterior

A.2. MySQL[23]

Las siguientes secciones tratan sobre las categorías básicas de comandos utilizados en SQL para realizar funciones numerosas manifestaciones. Estas funciones incluyen la construcción de objetos de base de datos, manipulación de objetos, llenar tablas de bases de datos con los datos, la actualización de los datos existentes en las tablas, borrar datos, perfroming consultas de bases de datos, control de acceso a base de datos y administración de base de datos global. Las principales categorías son,y sus comandos en cuadro 3:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data Control Language (DCL)
- Data administration commands
- Transactional control commands

DDL
CREATE TABLE
ALTER TABLE
DROP TABLE
CREATE INDEX
ALTER INDEX
DROP INDEX
CREATE VIEW
DROP VIEW

DML
INSERT
UPDATE
DELETE

DQL
SELECT

DCL
ALTER PASSWORD
GRANT
REVOKE
CREATE SYNONYM

DAC
START AUDIT
STOP AUDIT

TCC
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

Cuadro 3: Types of SQL Commands

A.3. Staticfiles – Gestión de archivos estáticos

Websites generalmente necesitan para servir archivos adicionales como imágenes, JavaScript o CSS. En Django, nos referimos a estos archivos como "static files". Django proporciona django.contrib.staticfiles para ayudar a tratarlo. En nuestro web ha usando los imágenes, antes de usarlos, debe hacer algunos configuración:

- En el settings.py, asegúrese de que *django.contrib.staticfiles* está incluido en sus INSTALLED_APPS.

```
INSTALLED_APPS = (
    ...
    'django.contrib.staticfiles',
)
```

- En el settings.py, definir STATIC_URL y STATICFILES_DIRS por ejemplo:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (
     '/home/meilin/djcode/debian_counting/debian_counting/static',
)
```

- Almacene sus archivos estáticos en una carpeta llamada estática de su aplicación. por ejemplo:

```
debian_counting/debian_counting/static/image/*.jpg
debian_counting/debian_counting/static/css/*.css
debian_counting/debian_counting/static/js/*.js
```

- En las plantillas, puede usar dos formas: codificar la url como /static/css/*.css, o preferibelmente, usar la etiqueta de plantilla estática para construir la url.

```
{% load staticfiles%}
<link rel="stylesheet" type="text/css" href="{% static "css/envision.min.css"%}" />
<script type="text/javascript" src="{% static "js/envision.min.js"%}" ></script>
...
{% load staticfiles%}
{% static "image/gsyc.png" as gsyc%}
{% static "image/w3c.png" as w3c%}
{% static "image/vcss.png" as vcss%}
{% static "image/wcag1AA.gif" as wcag1AA%}
...
<img src="{{ gsyc }}" border="0" alt="GSyC">
<img src="{{ w3c }}" border="0" alt="Valid HTML">
<img src="{{ vcss }}" border="0" alt="Valid css!">
<img src="{{ wcag1AA }}" border="0" alt="AA Accesibility">
```

B. Recursos libres

Aquí tenemos algunos buenos websites y recursos libres para estudiar las temas relacionados con el trabajo.

Python

- Python documentation: http://docs.python.org/2/index.html
- Mark Pilgrim: Dive Into Python http://www.diveintopython.net/
- Swaroop C H: A byte of Python http://swaroopch.com/notes/python/

Django

- Django documentation https://docs.djangoproject.com/en/1.5/
- Django Book http://www.djangobook.com/en/2.0/index.html

MySQL

- MySQL Reference Manuals: http://dev.mysql.com/doc/

HTML/CSS/JavaScript

- w3schools: http://www.w3schools.com/

Referencias

[1] Wikipedia http://en.wikipedia.org/wiki/Free_and_open_source_software
La traducción es realizada con Google Translate

[2] Free/Libre and Open Source Software: Survey and Study http://flossproject.org/

[3] The Free Software Definition http://www.gnu.org/philosophy/free-sw.html

[4] Free Software Foundation http://www.fsf.org/

[5] Open Source Initiative http://opensource.org/

[6] Wikipedia http://en.wikipedia.org/wiki/LAMP_(software_bundle)
La traducción es realizada con Google Translate

[7] The birth of the open source enterprise stack http://lwn.net/Articles/189186/

[8] LAMP Architecture.png author : kesavan muthuvel ; source : own work; URL : kesavan.info

[9] The Linux Kernel Archives https://www.kernel.org/

[10] The Apache Software Foundation http://www.apache.org/foundation/

[11] MySQL http://www.mysql.com/

[12] ORACLE MySQL http://www.oracle.com/us/products/mysql/overview/index.html

[13] Hypertext Preprocessor http://php.net/

[14] Python http://www.python.org/

[15] Wikipedia http://en.wikipedia.org/wiki/Web_application_framework
La traducción es realizada con Google Translate

[16] Web application framework http://en.wikipedia.org/wiki/Web_application_framework

[17] Django project https://www.djangoproject.com/

[18] Django (web framework) http://en.wikipedia.org/wiki/Django_(web_framework)

[19] Dont Reapeat Yourself http://c2.com/cgi/wiki?DontRepeatYourself

[20] Magnus Lie Hetland. Beginning Python From Novice to Professional Second Etition.Apress, 2008

[21] The Django Book 2.0 http://www.djangobook.com/en/2.0/index.html

[22] Django Software Foundation: Django Documentation Release 1.5.1 secion 3.2.7

[23] Ryan Stephens, Ron Plew, Arie.D.Jones: Sams Teach Yourself SQL in 24 Hours (4th Edition) Sams Publishing 2008