
curso
django
para perfeccionistas
con deadlines
maestros
del web
CATEGORIA
PROGRAMACIÓN
NIVEL
BASICO

SOBRE EL CURSO

CURSO DJANGO

El framework para detallistas con deadlines.

Versión 1/ abril 2012

Nivel: Básico

El curso Django se encuentra en línea en:

http://www.maestrosdelweb.com/guias/#guias-django

UN PROYECTO DE MAESTROS DEL WEB:

Autor: Sergio Infante Montero.

Edición: Eugenia Tobar.

Diseño y diagramación: Iván E. Mendoza.

CONTACTO:

http://www.maestrosdelweb.com/sitio/correo/

REDES SOCIALES:

Facebook: http://www.facebook.com/maestrosdelweb

Twitter: http://www.twitter.com/maestros

SOBRE EL AUTOR

SERGIO INFANTE MONTERO

Programador, difusor, traductor, contribuidor, activista y entusiasta de Software Libre y Código Abierto. Es consultor tecnológico de varias organizaciones dedicadas al comercio, educación e industria. En los últimos años ha dedicado mayor tiempo al desarrollo de aplicaciones web.

CONTACTO

Twitter: @neosergio

curso django

ÍNDICE

curso django

CAPÍTULO 1

EL WEB FRAMEWORK PARA PERFECCIONISTAS

El crecimiento de Python es cada vez mayor y esto se ha hecho más notorio en los últimos años, con la aparición de herramientas que hacen el trabajo más simple y eficiente con este lenguaje de programación. Una de esas herramientas es Django[1], el framework hecho en python para perfeccionistas.

VENTAJAS DE DJANGO

Aparte de las ventajas que tiene por ser framework, Django promueve el desarrollo rápido, se construyen aplicaciones en cuestión de días y con el conocimiento suficiente esos días se pueden reducir a horas.

Django impulsa el desarrollo de código limpio al promover buenas prácticas de desarrollo web, sigue el principio DRY (conocido también como Una vez y sólo una). Usa una modificación de la arquitectura Modelo-Vista-Controlador (MVC)[2], llamada MTV (Model – Template – View)[3], que sería Modelo-Plantilla-Vista, está forma de trabajar permite que sea pragmático.

ORIGEN DE DJANGO

Django nace como un proyecto para publicación de noticias de Lawrence Journal-World[4], lo interesante de Django es que desde un principio fue construido como una herramienta para resolver problemas reales en un entorno empresarial, fue diseñado para optimizar el tiempo de desarrollo y los requerimientos exigentes de los desarrolladores web. El nombre de Django es en honor al famoso músico francés Django Reinhardt[5].

1 http://www.djangoproject.com/
2 http://es.wikipedia.org/wiki/Modelo_Vista_Controlador
3 http://jeffcroft.com/blog/2007/jan/11/django-and-mtv/
4 http://www2.ljworld.com/
5 http://www.youtube.com/watch?v=nS2yIPAUxzA

¿QUIENES USAN DJANGO?

La lista de sitios web es enorme, pero es bueno fijarnos en los sitios más populares que usan Django como solución a sus necesidades. Estos son:

- dpaste.com
- The Washington post (en parte)[1]
- Bitbucket.org
- Disqus
- Instagram
- Pinterest
- michaelmoore.com
- theguardian
- lawrence.com
- curse[2]
- The New York Times (represent project)[3]
- Fluendo

Se pueden encontrar más casos de implementación e incluso varios de ellos con el código fuente en djangosites[4].

DJANGO EN AMÉRICA LATINA

En américa latina las experiencias con Django también están presentes, en la mayoría de países, ya se ha implementado y desarrollado con este estupendo framework. Por ejemplo en el Campus Party de Colombia del 2010 se dio una presentación del framework[5].

¿DÓNDE ENCONTRAR MAYOR INFORMACIÓN SOBRE DJANGO?

La mejor fuente de información con respecto a Django es la documentación[6] oficial del proyecto, también existen otras fuentes muy buenas de información, aquí una lista de estos recursos:

- Django en Español[7]: Sitio web de la comunidad española de Django.

1 http://www.washingtonpost.com/
2 http://curse.com/
3 http://projects.nytimes.com/represent/
4 http://www.djangosites.org/
5 http://www.youtube.com/watch?v=pQTIJ1-ODuU
6 https://docs.djangoproject.com/en/1.4/
7 http://django.es/

- Grupo de Google para usuarios de habla hispana de Django[1].
- Django Snippets[2]: Sitio donde se comparten Snippets.

MODO DE TRABAJO

Con el contenido del curso desarrollaremos una aplicación sencilla tocando los puntos más importantes. La aplicación será un recetario de comidas y bebidas, el cual se irá construyendo a lo largo de los capítulos, el código se encuentra publicado en github.

1 http://groups.google.com/group/django-es
2 http://djangosnippets.org/

CAPÍTULO 2

INSTALACIÓN Y PRIMERA APLICACIÓN

REQUISITOS PARA SU INSTALACIÓN

Django es un framework hecho en Python, por lo tanto se necesita que hayas previamente instalado Python (2.6 o 2.7) y tengas a la mano tu editor de texto favorito. Si no sabes que editores de texto usar, o cómo instalar Python lee la primera parte de la Guia Python, con esto es suficiente para empezar a trabajar.

> **NOTA:**
> Django no funciona con Python 3.0 actualmente, debido a incompatibilidades con el intérprete de Python.

Al estar hecho en Python, sería bueno tener en cuenta la sintaxis y fundamentos de este lenguaje, así que si aún no lo sabes y quieres reforzar tus conocimientos, revisa **la guía de Python**[1] antes de continuar con el curso o da un vistazo al vídeo de **mejorando.la**

Aprende Python en 20 minutos con Arturo Jamaica[2].

OBTENER DJANGO

Para obtener Django se puede:

- **Descargar la versión estable oficial más reciente**, desde el sitio oficial de Django[3] y descomprimir el archivo, debe aparecer una carpeta con el nombre de Django seguido del número de la versión bajada. (Para esta guía usaremos la versión 1.4).
- **Obtener la versión de desarrollo desde el repositorio de Django**, para ello se debe usar Subversión, de esta manera:

```
svn co http://code.djangoproject.com/svn/django/trunk/
```

Una vez que se ha obtenido Django, es momento de instalarlo.

1 http://www.maestrosdelweb.com/guias/#guia-python
2 http://www.youtube.com/watch?v=wp4DgNbGAUI
3 https://www.djangoproject.com/download/

PYTHON Y LAS VARIABLES DE ENTORNO EN WINDOWS

Los usuarios de Windows deben asegurarse de tener Python dentro de las variables de entorno, esto servirá para instalar Django y usarlo fácilmente. Si no sabes como hacerlo, revisa el vídeo Configurar variables en entorno Windows[1].

INSTALAR DJANGO

Una vez que se descomprime el archivo descargado, debemos acceder a la carpeta desde una terminal o ventana de comandos (en caso de los usuarios de Windows). Suponiendo que la versión que elegimos es la 1.4, se tendría que digitar:

```
cd Django-1.4
```

Ya en la carpeta de instalación de Django, se debe digitar la siguiente instrucción: (Debes de tener permisos de administrador):

```
python setup.py install
```

Si usas Ubuntu GNU/Linux, sería algo así:

```
sudo python setup.py install
```

Listo eso es todo, ya tienes Django instalado. Si se desea mayor información de como instalarlo o quizás algunas otras opciones, siempre está la documentación del mismo proyecto para guiarnos. Aquí pueden encontrar la Guía completa de instalación de Django[2].

Si usas OS X Lion, quizás lo más recomendado sería instalar Django usando pip:

- Primero, debemos tener Setup Tools para instalarlo, lo descargamos, lo descomprimimos y dentro del directorio (vía el terminal):
  `sudo python setup.py install`
- Una vez con Setup Tools instalado usamos pip:
  `sudo pip install Django`

1 http://www.youtube.com/watch?v=ciYoJPw9ORg
2 https://docs.djangoproject.com/en/1.4/topics/install/

PRIMER PROYECTO

Antes de empezar es bueno aclarar que la versión que vamos a utilizar en esta guía es la más reciente (1.4) y varias cosas han cambiado, la información que podrían encontrar en Internet probablemente se encuentre desactualizada.

Para crear nuestro primer proyecto, abrimos una terminal (o ventana de comandos si asi lo conoces en windows), nos ubicamos en la carpeta en donde queremos crear nuestro proyecto y digitamos:

```
django-admin.py startproject recetario
```

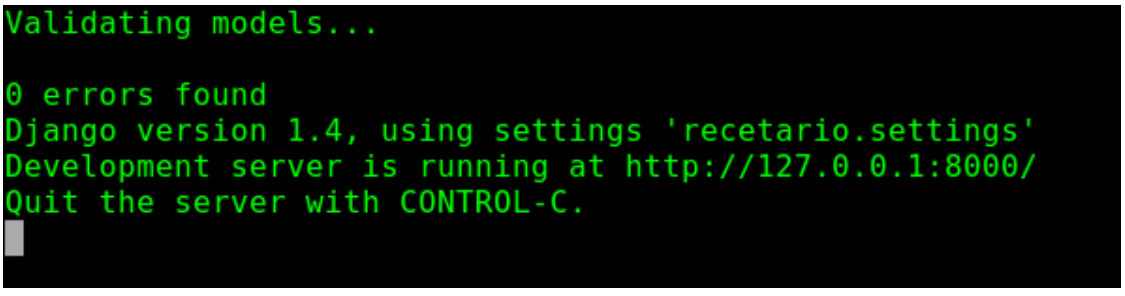
Esta instrucción creará dos directorios con el nombre del proyecto (en este caso: recetario) y 5 archivos distribuidos de la siguiente manera:

- manage.py
- recetario
  - __init__.py
  - settings.py
  - urls.py
  - wsgi.py

Para ver que el proyecto está funcionando en la terminal debemos escribir:

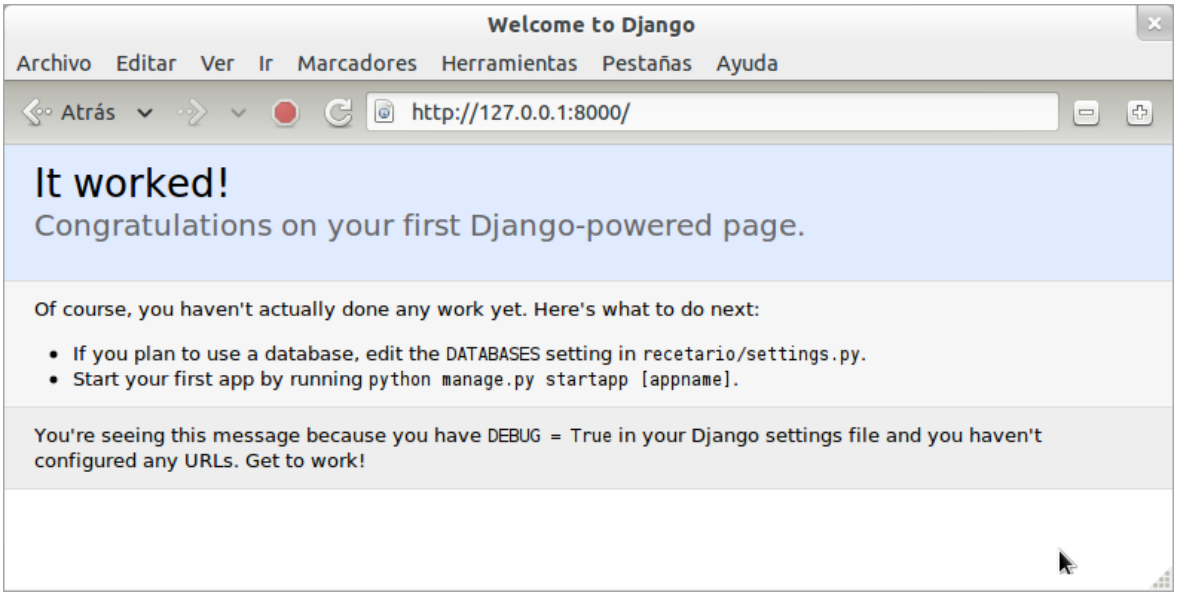
```
python manage.py runserver
```

Al ejecutar esa instrucción debemos visualizar un resultado como el siguiente:



PYTHON MANAGE.PY RUNSERVER

Abrimos en el navegador web la dirección `http://127.0.0.1:8000/` y debemos ver lo siguiente:



Ya tenemos nuestro proyecto creado. En el caso de que nos salga un error porque el puerto asignado esta en uso como la siguiente imagen:

```
Validating models...

0 errors found
Django version 1.4, using settings 'recetario.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
Error: That port is already in use.
```

DJANGO EN PUERTOS EN USO

sólo debemos indicar qué puerto usaremos para lanzar el servicio, por ejemplo si se desea usar el puerto 8888 entonces se tendría que digitar:

```
python manage.py runserver 8888
```

Usaríamos la dirección `http://127.0.0.1:8888/` y tendríamos como resultado lo siguiente:

```
Validating models...

0 errors found
Django version 1.4, using settings 'recetario.settings'
Development server is running at http://127.0.0.1:8888/
Quit the server with CONTROL-C.
```

Cada proyecto necesita de aplicaciones donde se puedan gestionar los modelos y las vistas. Un proyecto puede tener muchas aplicaciones:

PRIMERA APLICACIÓN

Para crear nuestra primer aplicación, desde la terminal y en la carpeta del proyecto, debemos digitar:

```
python manage.py startapp principal
```

Esto creará un directorio y cuatro archivos más, lo que nos dejaría con una estructura de archivos como esta:

- manage.py
- recetario
  - __init__.py
  - settings.py
  - urls.py
  - wsgi.py
- principal
  - __init__.py
  - models.py
  - test.py
  - views.py

EL PODEROSO SETTINGS.PY

Una parte muy importante del proyecto es el archivo **settings.py**, este archivo permite configurar la conexión a la base de datos, la zona horaria, el idioma, los directorios principales del proyecto, las aplicaciones del proyecto, entre otras cosas. Aprender a configurar este archivo permite optimizar el funcionamiento del proyecto, veremos las instrucciones principales a configurar:

CODIFICACIÓN DE CARACTERES

Nuestro idioma esta lleno de caracteres especiales como las tildes y las eñes que son las más comunes, la primera sugerencia para manejar esto eficientemente en Django es agregar la siguiente línea al archivo settings.py:

```
#encoding:utf-8
```

RUTA DEL PROYECTO

Es importante la configuración de la ruta del proyecto, esto permitirá lanzar la aplicación desde cualquier directorio y mover el proyecto a cualquier computador con Django instalado. Para ello debemos escribir las siguientes líneas en el archivo settings.py:

```
# Identificando la ruta del proyecto
import os
RUTA_PROYECTO = os.path.dirname(os.path.realpath(__file__))
```

> **NOTA:**
> Si no se configura la ruta del proyecto, cada vez que se cambia de directorio o de PC, se tendrá que cambiar las rutas de las plantillas, archivos estáticos y directorio de subida de contenido de los usuarios.

ADMINISTRADORES

Cuando Django tiene la opción de DEBUG=False (No lo cambies por ahora, déjalo en True), las notificaciones de error de código deben ser enviadas vía correo electrónico a los administradores, junto con los detalles completos del error. Para poner los datos de los administradores debemos buscar la siguiente porción:

```
ADMINS = (
 # ('Your Name', 'your_email@example.com'),
)
```

Y modificarla para que quede con los nombres de los administradores en forma de tupla, en mi caso lo dejaré así:

```
ADMINS = (
 ('Sergio Infante Montero', 'raulsergio9@gmail.com'),
)
```

CONFIGURACIÓN DE LA BASE DE DATOS

También podemos configurar la conexión a la base de datos según nuestras necesidades, Django soporta de manera predeterminada la conexión con postgresql, mysql, sqlite3 y oracle. En nuestro proyecto usaremos sqlite3. Debemos buscar la siguiente sección del archivo:

```
DATABASES = {
 'default': {
 'ENGINE': 'django.db.backends.', # Add 'postgresql_psycopg2',
'mysql', 'sqlite3' or 'oracle'.
 'NAME': '', # Or path to database file if using sqlite3.
 'USER': '', # Not used with sqlite3.
 'PASSWORD': '', # Not used with sqlite3.
 'HOST': '', # Set to empty string for localhost. Not used with
sqlite3.
 'PORT': '', # Set to empty string for default. Not used with sqlite3.
 }
}
```

Y dejarla de la siguiente manera:

```
DATABASES = {
 'default': {
 'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2',
'mysql', 'sqlite3' or 'oracle'.
 'NAME': 'recetario.db', # Or path to database file if using sqlite3.
 'USER': '', # Not used with sqlite3.
 'PASSWORD': '', # Not used with sqlite3.
 'HOST': '', # Set to empty string for localhost. Not used with
sqlite3.
 'PORT': '', # Set to empty string for default. Not used with sqlite3.
 }
}
```

ZONA HORARIA

Django permite configurar la zona horaria del proyecto, la lista de zonas horarias disponibles1 se pueden encontrar en la wikipedia. Para configurar debemos buscar lo siguiente:

```
TIME_ZONE = 'America/Chicago'
```

1 http://en.wikipedia.org/wiki/List_of_tz_zones_by_name

Yo lo configuraré en la zona horaria de Lima/Perú, así que lo modificaré de esta forma:

```
TIME_ZONE = 'America/Lima'
```

CONFIGURACIÓN DEL IDIOMA

Django también permite configurar el idioma que usará de manera predeterminada para su funcionamiento, para configurar esto debemos buscar lo siguiente:

```
LANGUAGE_CODE = 'en-us'
```

Se puede consultar la lista de idiomas disponibles para adecuarlo a nuestras necesidades, yo lo configuraré como español de Perú:

```
LANGUAGE_CODE = 'es-PE'
```

APLICACIONES INSTALADAS

Un proyecto en Django necesita de aplicaciones, algunas ya vienen configuradas de manera predeterminada. En nuestro proyecto usaremos la aplicación de administración y su documentación, estas ya vienen construidas y también nuestra primera aplicación creada líneas arriba, llamada principal. Para habilitar estas aplicaciones debemos buscar la siguiente sección que se encuentra casi al final del archivo **settings.py**

```
INSTALLED_APPS = (
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.sites',
 'django.contrib.messages',
 'django.contrib.staticfiles',
 # Uncomment the next line to enable the admin:
 # 'django.contrib.admin',
 # Uncomment the next line to enable admin documentation:
 # 'django.contrib.admindocs',
)
```

Y modificarlas de la siguiente manera:

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

OTRAS OPCIONES DE SETTINGS.PY

Existen otras configuraciones que iremos modificando durante el desarrollo del curso, como el directorio de plantillas, el contenido estático, los archivos cargados por los usuarios, entre otras. Por ahora con estas instrucciones tenemos lo básico para continuar con nuestro proyecto.

CREACIÓN DE LA BASE DE DATOS

Hasta el momento no se ha creado la base de datos o las tablas predeterminadas del proyecto, solo se han configurado los parámetros de conexión. Para crear la base de datos, debemos digitar desde la terminal o ventana de comandos, la siguiente instrucción (recordar que debemos estar en la carpeta de proyecto para que todo se realice correctamente):

```
python manage.py syncdb
```

Esta instrucción deberá mostrar el siguiente resultado:

```
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log

You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no):
```

PYTHON MANAGE.PY SYNCDB

Hay una pregunta que debemos responder, se refiere a la creación de un superusuario (un administrador del proyecto), para lo cual respondemos: yes (en caso de responder negativamente, no podremos usar inmediatamente el administrador predeterminado de Django). Luego de ello completamos la información que nos solicita.

```
You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'sergio'):
E-mail address: raulsergio9@gmail.com
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

Al finalizar ya estará creada la base de datos, como en este proyecto se usará sqlite3, deberá aparecer un archivo nuevo llamado recetario.db, este contiene las tablas y los datos iniciales del proyecto.

LAS DIRECCIONES URL DEL PROYECTO

Para visualizar los cambios que hicimos y la interfaz administrativa de Django, aún falta modificar un archivo, este es: urls.py (recordar que este archivo se encuentra dentro del directorio recetario). Este archivo contiene lo siguiente:

```
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'^$', 'recetario.views.home', name='home'),
 # url(r'^recetario/', include('recetario.foo.urls')),
 # Uncomment the admin/doc line below to enable admin documentation:
 # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
 # Uncomment the next line to enable the admin:
 # url(r'^admin/', include(admin.site.urls)),
)
```

Debemos dejarlo de esta manera:

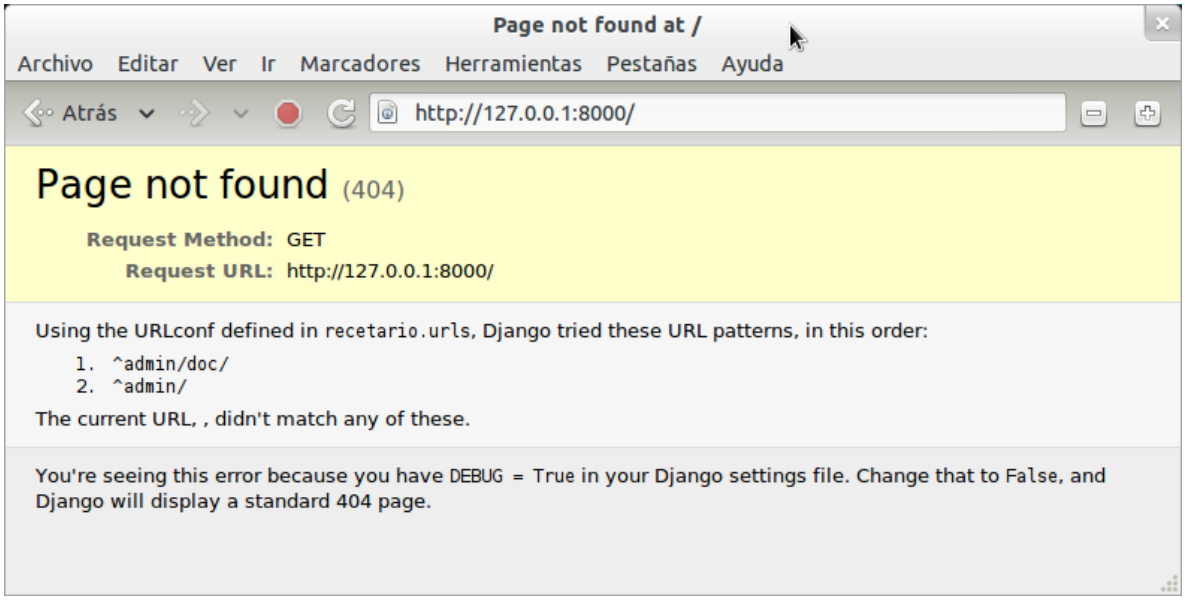
```
from django.conf.urls import patterns, include, url
from django.contrib import admin
admin.autodiscover()
urlpatterns = patterns('',
 url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
 url(r'^admin/', include(admin.site.urls)),
)
```

EJECUCIÓN DEL PROYECTO

Una vez que tenemos todo listo y configurado, debemos nuevamente iniciar el servidor de desarrollo que tiene el proyecto. Ya hicimos esto al principio, solo debemos digitar desde la terminal nuevamente (dentro del directorio del proyecto):

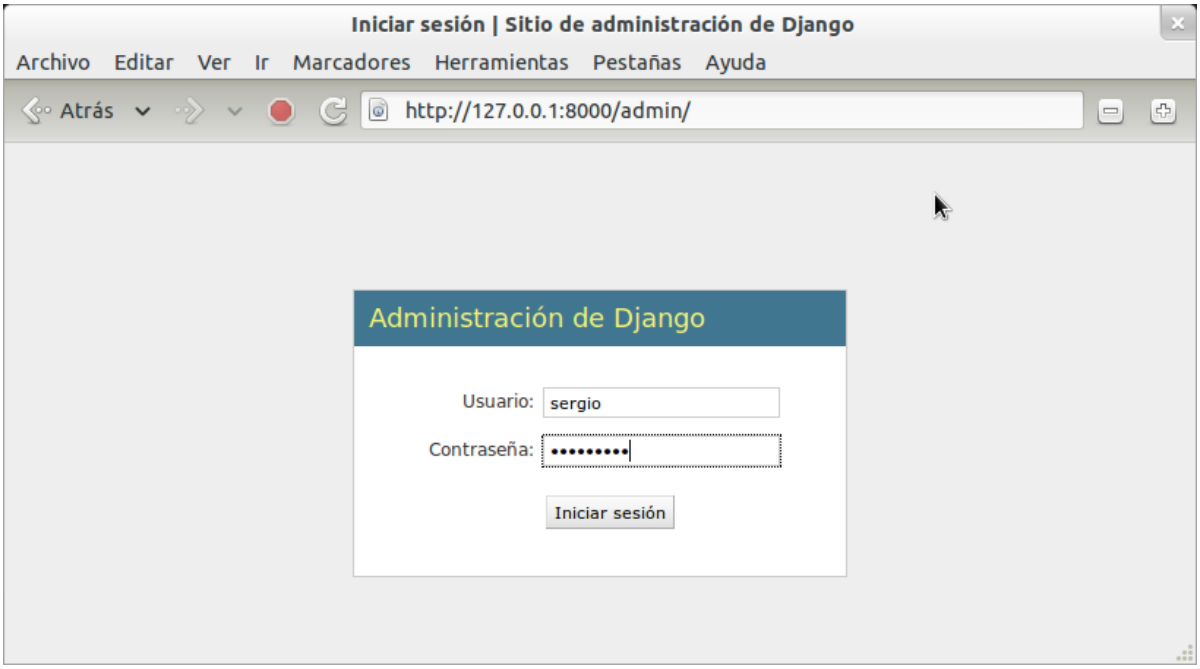
```
python manage.py runserver
```

Lucirá de esta manera:



DJANGO 404 PREDETERMINADO

Nos muestra un error 404, luego veremos el por qué. Para ingresar a la interfaz administrativa que ya viene construida con Django, ingresaremos a la dirección: `http://127.0.0.1:8000/admin`, en donde pondremos el nombre del superusuario y su respectiva contraseña, creados anteriormente (líneas arriba) con la opción syncdb.



DJANGO ADMIN LOGIN

Si todo fue correcto debemos visualizar la interfaz administrativa:

NOTA:

El proyecto se encuentra en github[1] por si te perdiste de algo y deseas ver algún archivo del mismo.

1 http://neosergio.github.com/recetario_mdw

CAPÍTULO 3

ENTENDIENDO COMO TRABAJA DJANGO

Nuestro primer proyecto ya se encuentra configurado para continuar con la construcción de los modelos. Sin embargo, antes de ello, debemos entender como funciona Django.

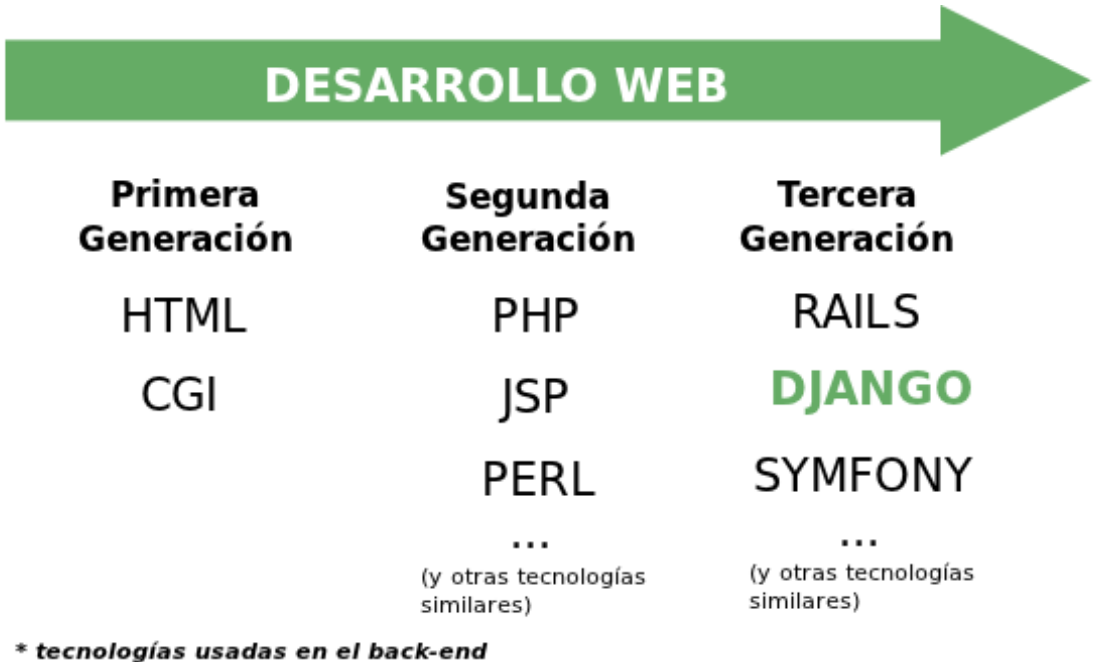
NOTA

Para tener una estupenda experiencia con Django, el orden de los archivos y directorios deben respetarse, caso contrario el caos llegará rápidamente a nuestro proyecto y fracasará al momento de hacerlo crecer con más personas o requerimientos.

Para contextualizar la explicación de cómo funciona Django revisa el vídeo La revolución de Ruby y Python[1]. Ahora que ya entiendes el contexto, empezamos!

MTV Y DJANGO

Se podría clasificar a Django como parte de la tercera generación del desarrollo web:



1 http://www.youtube.com/watch?v=c6JLEsXhEEc

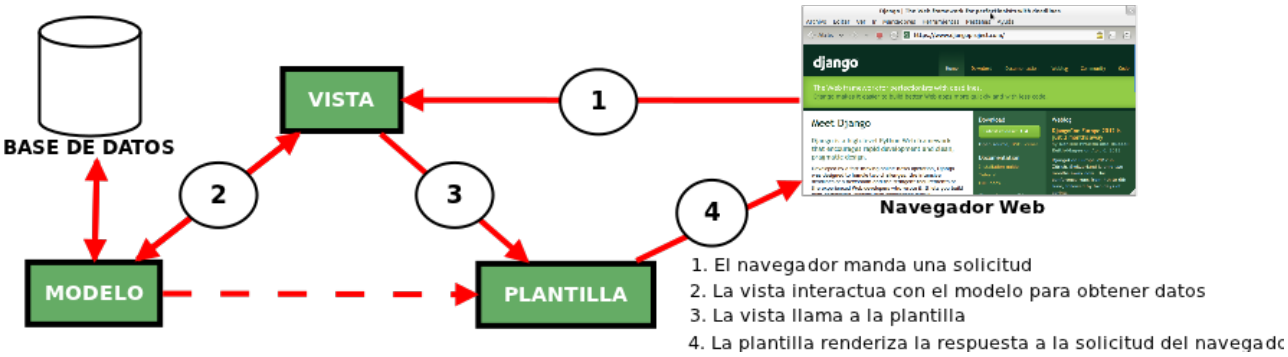
Sin embargo más allá de las clasificaciones que podrían existir, está el entender como funciona realmente, al entenderlo se puede llegar a dominarlo.

Al principio del curso[1] de Django, les dije que era un framework MTV (una modificación de MVC, nada que ver con música), esto se debe a que los desarrolladores no tuvieron la intención de seguir algún patrón de desarrollo, sino hacer el framework lo más funcional posible.

Para empezar a entender MTV debemos fijarnos en la analogía con MVC.

- El modelo en Django sigue siendo modelo.
- La vista en Django se llama Plantilla (Template).
- El controlador en Django se llama Vista.

Una imagen nos hará entender mejor esta relación:



FUNCIONAMIENTO DEL MTV DE DJANGO

Veamos que hace cada uno de ellos con un poco más de detalle y algunos conceptos adicionales.

EL MODELO

El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

LA VISTA

La vista se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizados, entre otras cosas más que iremos viendo conforme avanzamos con el curso. El ORM de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios exter-

1 http://www.maestrosdelweb.com/editorial/curso-django-introduccion/

nos y la validación de datos a través de formularios. Lo más importante a entender con respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla.

LA PLANTILLA

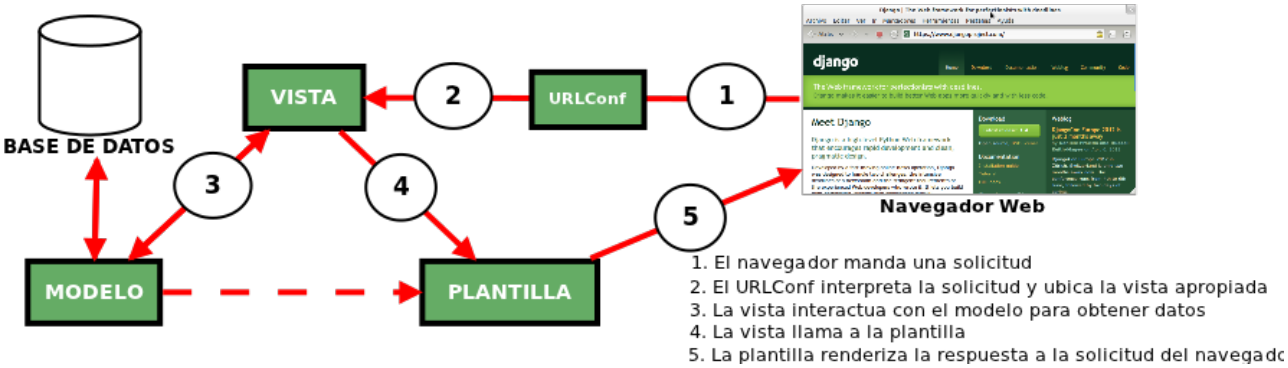
La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc). Por ahora nos enfocaremos a lo básico: el HTML.

La plantilla recibe los datos de la vista y luego los organiza para la presentación al navegador web. Las etiquetas que Django usa para las plantillas permiten que sea flexible para los diseñadores del frontend, incluso tiene estructuras de datos como if, por si es necesaria una presentación lógica de los datos, estas estructuras son limitadas para evitar un desorden poniendo cualquier tipo de código Python. Esto permite que la lógica del sistema siga permaneciendo en la vista.

LA CONFIGURACIÓN DE LAS RUTAS

Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, ésta configuración es conocida como URLConf. El trabajo del URLConf es leer la URL que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo. El URLConf esta construido con expresiones regulares en Python[1] y sigue su filosofía: Explícito es mejor que implícito[2].

Este URLConf permite que las rutas que maneje Django sean agradables y entendibles para el usuario. Si consideramos al URLConf en el esquema anterior tendríamos este resultado más completo.



FUNCIONAMIENTO DEL MTV DE DJANGO Y SU URLCONF

1 http://www.maestrosdelweb.com/editorial/guia-python/
2 http://www.python.org/dev/peps/pep-0020/

LOS ARCHIVOS PREDETERMINADOS

Otra parte importante es entender el propósito de los archivos que se crean de manera predeterminada, estos son:

ARCHIVOS DEL PROYECTO

- **__init__.py**: Es un archivo vacío que le dice a Python que debe considerar este directorio como un paquete de Python. Si se desea saber más, se puede consultar la documentación sobre los paquetes[1].
- **manage.py**: Contiene una porción de código que permite interactuar con el proyecto de Django de muchas formas. Si se desea mayor detalle se encuentra la documentación oficial con respecto a manage.py[2].
- **settings.py**: Contiene todas las configuraciones para el proyecto, la documentación al respecto puede darnos mas detalles de la configuración de un proyecto en Django[3].
- **urls.py**: Contiene las rutas que están disponibles en el proyecto, manejado por URLConf, los detalles completos como siempre en la documentación oficial nos da más detalles sobre las urls y Django[4].

ARCHIVOS DE LA APLICACIÓN

- **__init__.py**: La misma descripción anterior (líneas arriba).
- **models.py**: Se declaran las clases del modelo.
- **views.py**: Se declaran las funciones de la vista.
- **test.py**: Se declaran las pruebas necesarias para la aplicación, para mayor detalle sobre las pruebas y Django[5] se puede consultar la documentación del proyecto.

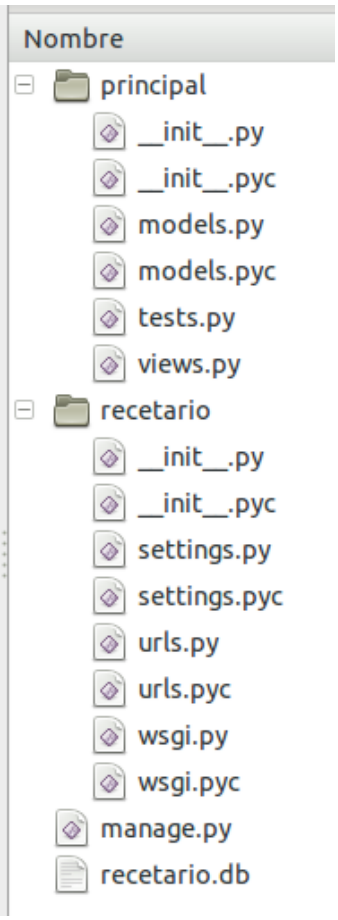
Una vez que tenemos claro el funcionamiento básico de Django, desarrollemos un ejemplo dentro del proyecto que iniciamos en el primer capítulo[6], para experimentar por nuestra cuenta.

1 http://docs.python.org/tutorial/modules.html#packages
2 https://docs.djangoproject.com/en/dev/ref/django-admin/#ref-django-admin
3 https://docs.djangoproject.com/en/dev/topics/settings/#topics-settings
4 https://docs.djangoproject.com/en/dev/topics/http/urls/#topics-http-urls
5 https://docs.djangoproject.com/en/dev/topics/testing/
6 http://www.maestrosdelweb.com/editorial/curso-django-instalacion-y-primera-aplicacion/

MODELO, VISTA Y PLANTILLA EN 15 MINUTOS

Antes de continuar, deben entender que esto solamente es un pequeño ejemplo, el modelo, vista y plantillas serán desarrollados a detalle en los próximos capítulos. Si eres muy impaciente siempre tienes a disposición la documentación oficial del proyecto[1].

Nuestro proyecto debe tener una estructura similar a esta:



ÁRBOL DE FICHEROS DE UN PROYECTO EN DJANGO

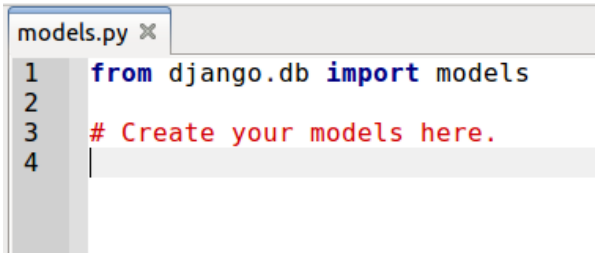
Tendremos que manejar para este proyecto 6 archivos:

- **models.py** (ya existe dentro de la carpeta principal).
- **views.py** (ya existe dentro de la carpeta principal).
- **admin.py** (aún no existe, pero estará dentro de la carpeta principal).
- **urls.py** (ya existe dentro de la carpeta recetario).
- **settings.py** (ya existe dentro de la carpeta recetario).
- **lista_bebidas.html** (aún no existe, estará dentro de una carpeta en la carpeta recetario).

1 https://docs.djangoproject.com/

MODELS.PY

Al principio `models.py` luce asi:



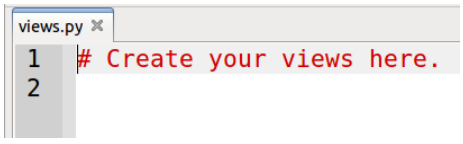
MODELS.PY PREDETERMINADO

Usamos nuestro editor de texto favorito para editar models.py y redactamos lo siguiente:

```
from django.db import models
class Bebida(models.Model):
 nombre = models.CharField(max_length=50)
 ingredientes = models.TextField()
 preparacion = models.TextField()
 def __unicode__(self):
 return self.nombre
```

VIEWS.PY

`views.py` luce así al principio:



Igualmente con nuestro editor favorito abrimos views.py y lo dejamos de la siguiente forma:

```
from principal.models import Bebida
from django.shortcuts import render_to_response
def lista_bebidas(request):
 bebidas = Bebida.objects.all()
 return render_to_response('lista_bebidas.html',{'lista':bebidas})
```

ADMIN.PY

Este archivo no existe, así que lo creamos manualmente dentro de nuestro directorio de la aplicación, en este caso dentro del directorio principal. Lo debemos dejar de esta manera:

```
from principal.models import Bebida
from django.contrib import admin
admin.site.register(Bebida)
```

URLS.PY

Este archivo ya existe, se encuentra dentro del directorio recetario, así que lo único que debemos hacer es modificarlo para dejarlo de la siguiente manera:

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
admin.autodiscover()
urlpatterns = patterns('',
 url(r'^$','principal.views.lista_bebidas'),
 url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
 url(r'^admin/', include(admin.site.urls)),
)
```

SETTINGS.PY

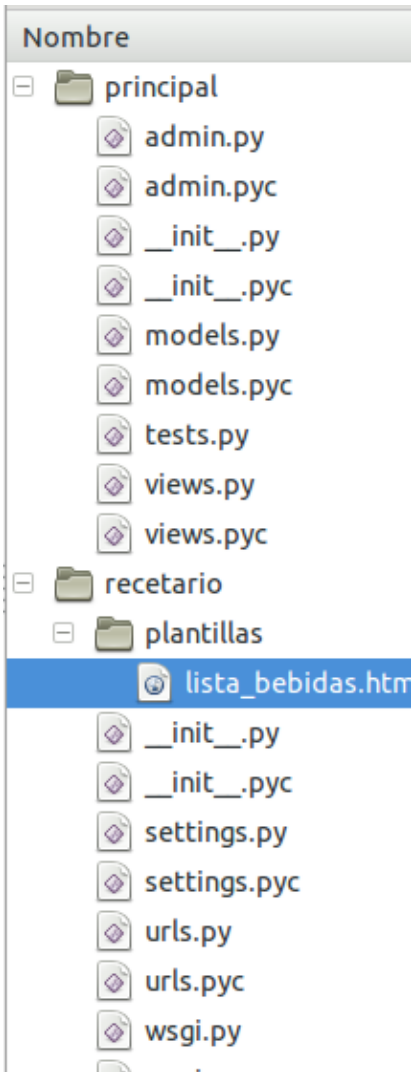
Este archivo lo modificamos en el capítulo anterior, se encuentra dentro de recetario. Debemos buscar la directiva TEMPLATE_DIRS (Debe encontrarse en la línea 113 aproximadamente). Debemos dejar esta sección de esta forma:

```
TEMPLATE_DIRS = (
 os.path.join(RUTA_PROYECTO,'plantillas'),
)
```

En esta sección le estamos diciendo que debe buscar las plantillas dentro de la carpeta del proyecto, en una carpeta llamada plantillas, como esta carpeta aún no existe debemos crearla manualmente dentro de recetario.

LISTA_BEBIDAS.HTML

Este archivo no existe, lo crearemos manualmente dentro de la carpeta plantillas (si, esa que acabas de crear dentro de recetario). Debes finalizar con una estructura similar a esta:



ÁRBOL DE FICHEROS DEL PROYECTO RECETARIO

Una vez creado le ponemos la etiquetación HTML que nos permita visualizar la lista de bebidas. Podría quedarse de esta manera:

```
<!DOCTYPE html>
<html lang=’es’>
 <head>
 <title>Lista de bebidas de ejemplo</title>
 <meta charset=’utf-8’>
 </head>
```

```
 <body>
 <h1>Recetario de Bebidas</h1>
 {% for elemento in lista %}
 <ul>
 <li><strong>{{elemento.nombre}}</strong></li>
 <li>{{elemento.ingredientes}}</li>
 <li>{{elemento.preparacion}}</li>
 </ul>
 {% endfor %}
 </body>
</html>
```

NOTA:
Tener precaución al momento de digitar el código en Python, las mayúsculas y minúsculas importan y mucho.

PONER EN FUNCIONAMIENTO TODO

Tenemos todo listo para ver funcionando nuestro ejemplo, para ello debemos sincronizar nuevamente nuestra base de datos; nos ubicamos en la carpeta principal del proyecto (aquella donde se encuentra el archivo manage.py). Una vez en la carpeta principal, desde una terminal digitamos:

```
python manage.py syncdb
```

Esto nos dará el siguiente resultado: (prestar atención a la segunda línea que nos indica la creación de una nueva tabla):

```
Creating tables ...
Creating table principal_bebidas
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

CREACIÓN DE TABLA PRINCIPAL_BEBIDAS

Una vez realizado esto, lanzamos nuestro servidor de desarrollo:

```
python manage.py runserver
```

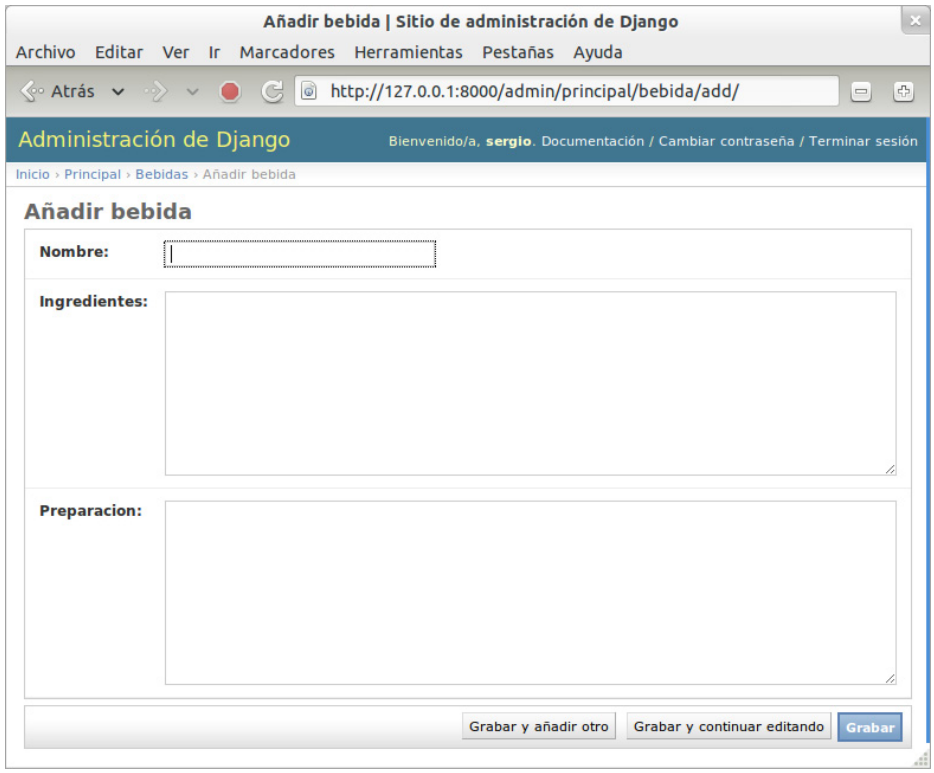
Nos ubicamos en `http://127.0.0.1:8000/admin`, ingresamos con nuestros datos configurados en la primera vez que sincronizamos la base de datos (si no recuerdas los datos que pusiste, puedes borrar el archivo recetario.db y volver a digitar: python manage.py syncdb para resincronizar).

Una vez dentro de la interfaz administrativa veremos: (notar que aparece una opción Bebidas).

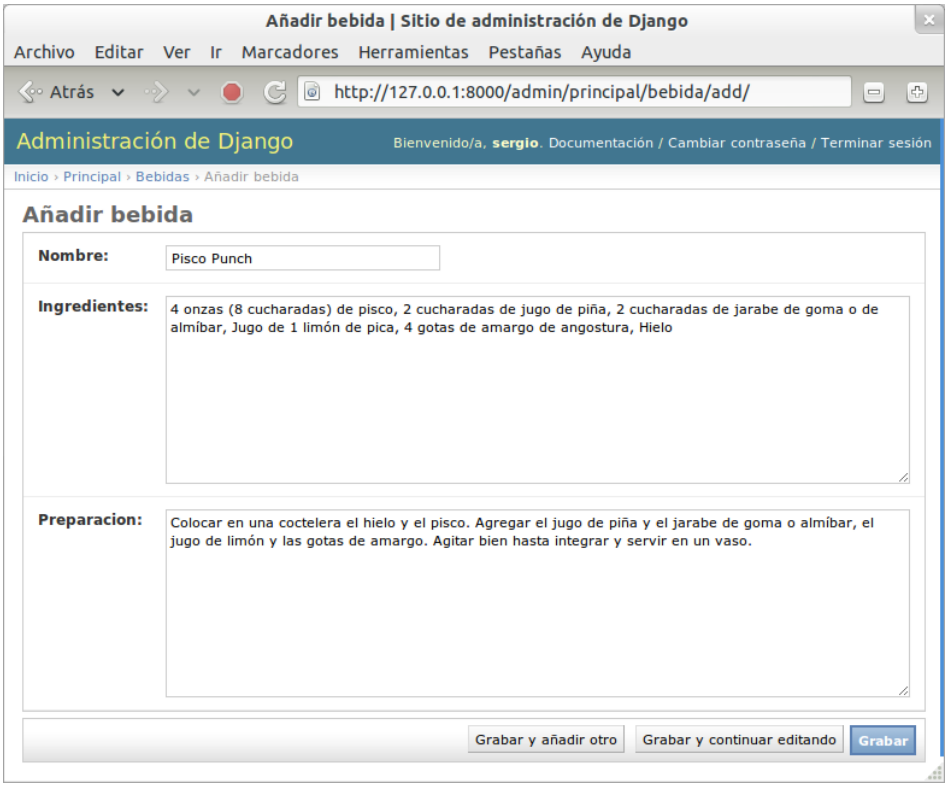
INTERFAZ ADMINISTRATIVA DE DJANGO + BEBIDAS

Podemos seguir intuitivamente la interfaz e ingresar datos:

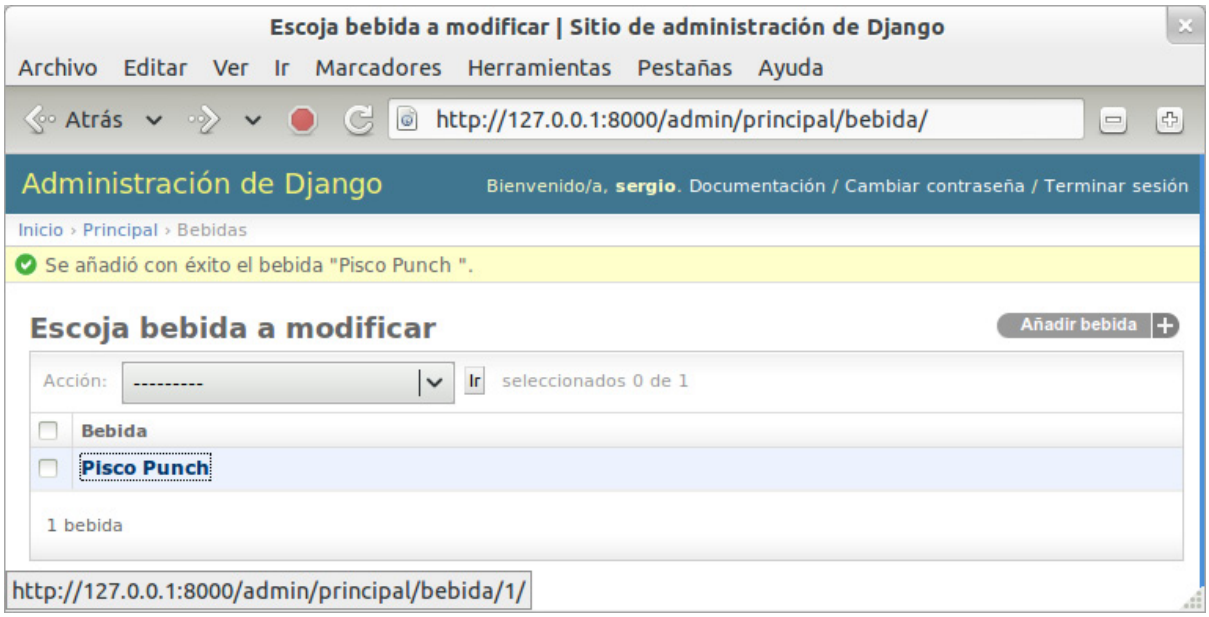
INTERFAZ INICIAL DE BEBIDAS



FORMULARIO PARA INGRESO DE DATOS

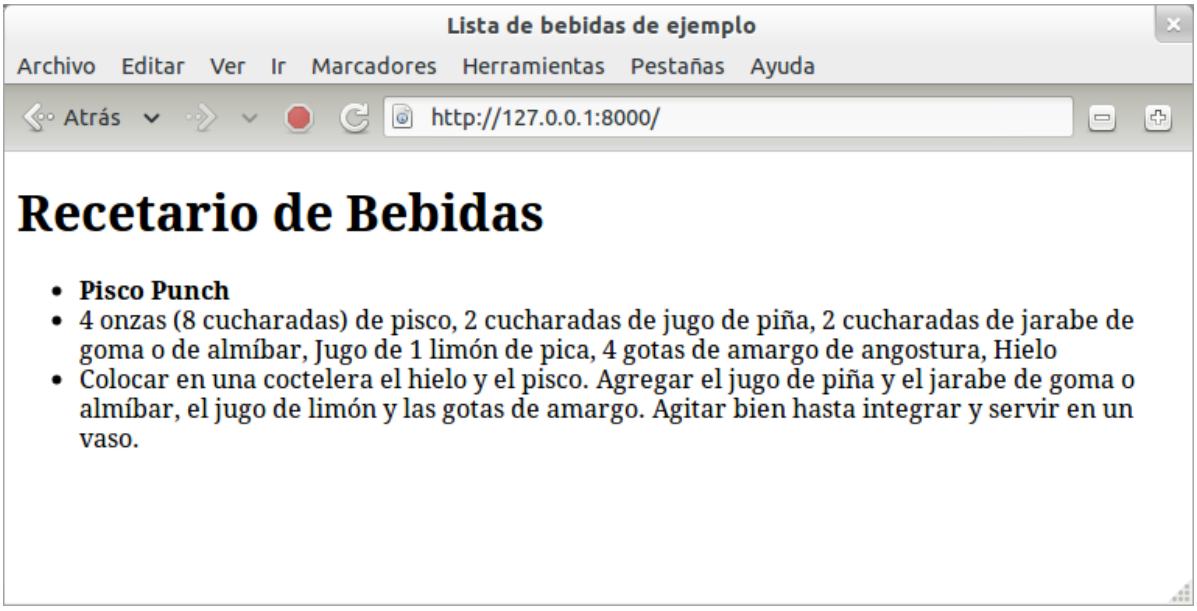


INGRESO DE CONTENIDO



LISTA DE ELEMENTOS INGRESADOS

Una vez guardados podemos ir a `http://127.0.0.1:8000/` y veremos lo que acabamos de ingresar. De esta manera ya tienes una interfaz rápida de ingreso y visualización de datos.



EJEMPLO DE PLANTILLA CON DJANGO

En este capítulo he tocado muchos puntos de forma superficial, esto no debe preocuparte, porque en los siguientes capítulos desarrollaré a detalle esos puntos, deseo recordarte que el objetivo de este pequeño ejemplo, es disfrutar la experiencia de la velocidad de Django, para el desarrollo de aplicaciones.

Si desean ver los archivos oficiales del proyecto, no olviden que pueden acceder al repositorio del mismo en Github[1]. Los datos de acceso para la interfaz administrativa del ejemplo del repositorio son los siguientes:

usuario: sergio
clave: maestros

1 http://neosergio.github.com/recetario_mdw

CAPÍTULO 4

EL MODELO DE DATOS

Una vez que ya tenemos instalado Django, creado nuestro primer proyecto y haber revisado como funciona, necesitamos definir el modelo de datos para nuestra aplicación.

EL MODELO

Un modelo es la representación de los datos de nuestra aplicación. Contiene los campos básicos y el comportamiento de los datos que serán almacenados. Por lo general, cada modelo se convierte en una tabla de la base de datos.

LO FUNDAMENTAL

- Cada modelo es una subclase de `django.db.models.Model`.
- Cada atributo de un modelo representa a un campo de una tabla.
- Django automáticamente nos da acceso a la base de datos.

REFERENCIA DE LOS CAMPOS

La referencia de la versión actual está detallada por completo en la documentación del framework[1], con este recurso podemos elaborar todas las referencias necesarias para el proyecto, sin embargo, leerlo por completo puede resultar al principio un poco tedioso. Para ello siempre es muy útil un cheat sheet o chuleta para visualizarlos juntos (ver en siguiente página).

EJEMPLO DE MODELO EN NUESTRO PROYECTO

Vamos con el ejemplo del capítulo para poder entender mejor como va esto del modelo.

Nuestro archivo: `models.py` del capítulo anterior quedó así:

```
from django.db import models
class Bebida(models.Model):
 nombre = models.CharField(max_length=50)
 ingredientes = models.TextField()
 preparacion = models.TextField()
 def __unicode__(self):
 return self.nombre
```

1 https://docs.djangoproject.com/en/1.4/ref/models/fields/

Referencia del Modelo

Tipos de campos	Opciones de campo
AutoField **BigIntegerField** **BooleanField** **NullBooleanField** **CharField** max_length **CommaSeparateIntegerField** **DateField** auto_now = False auto_now_add = False **DateTimeField** auto_now = False auto_now_add = False **DecimalField** max_digits = 10 decimal_places = 2 **IntegerField** **SlugField** max_length = 50 **SmallIntegerField** **TextField** auto_now = False auto_now_add = False **URLField** verify_exists = True max_length = 200 **FloatField** **IPAddressField** **GenericIPAddressField** **EmailField** max_length = 75 **FileField** upload_to = 'cargas/' storage = FileSystemStorage **FilePathField** path = '/home/archivos' match = r'\.pngS' recursive = False **ImageField** upload_to = 'cargas/' height_field = 'nombre_campo' width_field = 'nombre_campo' **PositiveIntegerField** **PositiveSmallIntegerField** **ForeignKey** related_name = 'modelo' limit_choices_to = query_kwargs to_field = 'llave_campo' **ManyToManyField** related_name = 'modelo' limit_choices_to = query_kwargs symmetrical = True through = 'ModeloIntermedio' **OneToOneField** parent_link = 'campo'	**null** = False **blank** = False **choices** = tupla_de_opciones **db_column** = 'nombre de columna' **db_index** = False **db_tablespace** = 'nombre_tablespace' **default** = 'valor' **editable** = True **error_messages** = diccionario_de_mensajes **help_text** = 'text' **primary_key** = True **unique** = True **unique_for_date** = 'campo_fecha' **unique_for_month** = 'campo_fecha' **unique_for_year** = 'campo_fecha' **verbose_name**='nombre' **validators** = lista_de_validadores

Este modelo solo fue un ejemplo para saber cómo funcionaba, ahora escribiré algunas líneas que nos ayudarán a tener un modelo más completo:

```
#encoding:utf-8
from django.db import models
from django.contrib.auth.models import User
class Bebida(models.Model):
 nombre = models.CharField(max_length=50)
 ingredientes = models.TextField()
 preparacion = models.TextField()
 def __unicode__(self):
 return self.nombre
class Receta(models.Model):
 titulo = models.CharField(max_length=100, unique=True)
 ingredientes = models.TextField(help_text='Redacta los ingredientes')
 prepacion = models.TextField(verbose_name='Preparación')
 imagen = models.ImageField(upload_to='recetas', verbose_
name='Imágen')
 tiempo_registro = models.DateTimeField(auto_now=True)
 usuario = models.ForeignKey(User)

 def __unicode__(self):
 return self.titulo
```

Son clases en Python (respetar la indentación), les explicaré de qué trata todo esto:

```
#encoding:utf-8 <- Esta línea permite usar tildes y caracteres
especiales
from django.db import models <- Clase con la descripción de modelos
from django.contrib.auth.models import User <- Llama al modelo usuario
```

La clase Bebida del capítulo anterior sólo era un ejemplo, no le daremos importancia de ahora en adelante, pasemos con la clase Receta (y sus respectivos comentarios previos a cada línea).

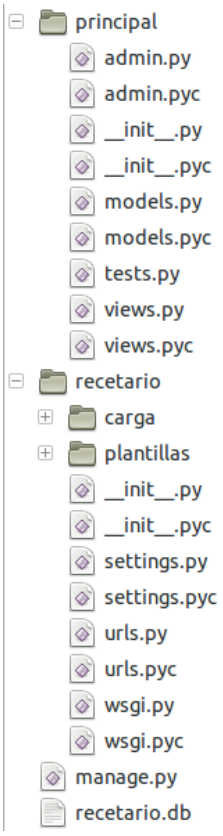
```
#Dato cadena, longitud máxima 100 y único
titulo = models.CharField(max_length=100, unique=True)
#Dato texto, con texto de ayuda
```

```
ingredientes = models.TextField(help_text='Redacta los ingredientes')
#Dato texto, con nombre: Preparación
prepacion = models.TextField(verbose_name='Preparación')
#Dato imagen, se almacenarán en la carpeta recetas, titulo: Imágen
imagen = models.ImageField(upload_to='recetas', verbose_name='Imágen')
#Dato Fecha y Hora, almacena la fecha actual
tiempo_registro = models.DateTimeField(auto_now=True)
#Enlace al modelo Usuario que Django ya tiene construido
usuario = models.ForeignKey(User)
```

La clase Receta (modelo) tiene un atributo imagen, el cuál está direccionando las cargas que haga el usuario a la carpeta 'recetas' (carpeta que estará dentro de otra llamada: 'carga'), para que esto funcione debemos modificar nuestro archivo settings.py, exactamente debemos buscar la línea: MEDIA_ROOT (debe ser la número 56 aproximadamente), y dejarla así:

```
MEDIA_ROOT = os.path.join(RUTA_PROYECTO,'carga')
```

Luego de haber puesto esta línea, debemos crear una carpeta que se llame 'carga' dentro de la carpeta del proyecto. Debemos tener un árbol de ficheros de este tipo:



MAPA DE FICHEROS DEL PROYECTO RECETARIO

Dentro de la carpeta 'carga' aparecerá una carpeta 'recetas' al momento de guardar un registro. Pero antes de ello debemos buscar el archivo admin.py y dejarlo asi:

```
from principal.models import Bebida, Receta
from django.contrib import admin
admin.site.register(Bebida)
admin.site.register(Receta)
```

Esto nos permitirá agregar el modelo Receta dentro de la interfaz administrativa. Por último debemos habilitar las URL para visualizar las imágenes, para ello debemos dejar el archivo urls.py asi:

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
from django.conf import settings
admin.autodiscover()
urlpatterns = patterns('',
 url(r'^$','principal.views.lista_bebidas'),
 url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
 url(r'^admin/', include(admin.site.urls)),
 url(r'^media/(?P<path>.*)$','django.views.static.serve',
 {'document_root':settings.MEDIA_ROOT,}
 ),
)
```

Esto nos permitirá acceder a las imágenes que subamos desde `http://127.0.0.1:8000/media/recetas/nombre-imagen.jpg` (o cualquier otra imagen).

PONIENDO A CORRER TODO

Para hacer funcionar todo, debemos sincronizar la base de datos nuevamente (esto se debe hacer cada vez que se modifique el modelo).

> **NOTA**
> Toda ejecución de comando se realiza desde una terminal o ventana de comando y dentro de la carpeta del proyecto (donde se encuentra el archivo manage.py)

```
python manage.py syncdb
```

Esto nos debe mostrar un resultado así (prestar atención a la segunda línea):

```
Creating tables ...
Creating table principal_receta
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

PYTHON MANAGE.PY SYNCDB

En caso de que queramos modificar un modelo ya existente podemos también reiniciar todos los modelos de la aplicación principal así:

```
python manage.py reset principal
```

Sin embargo si ya se tienen datos almacenados estos se perderán al momento de reiniciarlos, para evitar eso podemos usar aplicaciones como South, que nos permitirán trabajar con los datos de manera más profesional, puedes visualizar el ejemplo South (Django Tool)[1].

PROBANDO EL EJEMPLO

Ya tenemos todo listo, el modelo nuevo sincronizado, las configuraciones listas y la interfaz administrativa. Corremos el proyecto:

```
python manage.py runserver
```

Entramos a `http://127.0.0.1:8000/admin` y debemos observar el nuevo modelo: Recetas, listo para ser usado. Para ver las imágenes que se van cargando podemos ir a `http://127.0.0.1:8000/media/recetas/nombre-imagen.jpg` y si son proactivos, pueden modificar la plantilla de la semana pasada para ver el nuevo modelo de Recetas y sus imágenes, o pueden crear nuevos modelos para ver como pueden interactuar entre sí.

No se olviden que pueden revisar el repositorio del proyecto en github[2] si desean comprobar cómo deben quedar los archivos (en orden y en contenido).

1 http://www.youtube.com/watch?v=s41rV2kfRPI
2 http://neosergio.github.com/recetario_mdw

CAPÍTULO 5

EL SHELL DE DJANGO

Los modelos nos permiten manipular los datos: registrarlos, editarlos, actualizarlos, consultarlos, eliminarlos y realizar procesos con ellos. Toda esta manipulación se reflejará en las vistas y posteriormente en las plantillas para mostrar los resultados en el navegador, esta manipulación se le conoce generalmente como: Consultas.

EL SHELL

Es el intérprete interactivo de Python, que nos permitirá probar los modelos, hacer consultas, analizar resultados, antes de elaborar las vistas. Es muy útil si queremos ahorrar tiempo al momento de responder a los requerimientos que los usuarios de la aplicación puedan necesitar.

Para acceder al shell, abrimos una terminal o ventana de comandos y nos ubicamos en la carpeta de proyecto (en donde se encuentre el archivo manage.py) y digitamos:

```
python manage.py shell
```

Debemos visualizar el siguiente resultado:

```
Python 2.7.3 (default, Apr 20 2012, 22:44:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

Lo importante es reconocer los elementos: el prompt se representa por >>> y el resultado de nuestras instrucciones se visualizarán en nuevas líneas (sin ningún símbolo previo a ellas).

LAS CONSULTAS

Las consultas en base a los modelos de Django son la base de todo el desarrollo en este framework, estas consultas nos permiten saber, por ejemplo, la lista de usuarios, los correos electrónicos de los que hacen comentarios, el primer comentario de un artículo, los artículos del mes de mayo y la lista de posibilidades es larga.

Las consultas están descritas muy claramente en la documentación oficial[1], para poder hacer esto más simple aquí también usaremos esta pequeña chuleta o cheat sheet como también es conocido.

Metodos para hacer consultas

Metodos que retornan nuevos conjuntos de consultas	Metodos que no retornar conjuntos de consultas
filter(búsquedas) exclude(búsquedas) annotate(anotaciones) order_by(campos) reverse( ) distinct( ) values(campos) values_list(campos) dates(campos) none( ) all( ) select_related(campos) prefetch_related(búsquedas) extra( ) defer(campos) only(campos) using(alias) select_for_update(nowait=False)	get(búsquedas) create(atributos) get_or_create(atributos) bulk_create(objetos) count( ) in_bulk(id_list) iterator( ) latest(campo) aggregate(agregaciones) exists( ) update(atributos) delete( )
Búsquedas	**Funciones de Agregaciones / Anotaciones**
exact iexact contains icontains in gt gte lt lte startswith istartswith endswith iendswith range year month day week_day isnull search regex iregex	Avg Count Max Min StdDev Sum Variance

Elaborado por: **Sergio Infante (@neosergio)**

Imagen: http://www.maestrosdelweb.com/images/2012/05/django-queries.pdf

La referencia de todos estos métodos[2] se encuentran también en la documentación oficial.

1 https://docs.djangoproject.com/en/1.4/topics/db/queries/
2 https://docs.djangoproject.com/en/1.4/ref/models/querysets/

EJEMPLOS

Ahora pasamos a lo divertido, seguimos trabajando con nuestro proyecto de ejemplo[1], para poder realizar algunos ejemplos tenemos que modificar y corregir el models.py y dejarlo así: (ya no existe el modelo Bebida, ya no es necesario, también se debe eliminar cualquier rastro del modelo Bebida de admin.py)

```
#encoding:utf-8
from django.db import models
from django.contrib.auth.models import User

class Receta(models.Model):
    titulo = models.CharField(max_length=100, verbose_name='Título', unique=True)
    ingredientes = models.TextField(help_text='Redacta los ingredientes')
    preparacion = models.TextField(verbose_name='Preparación', help_text='El proceso de preparación')
    imagen = models.ImageField(upload_to='recetas', verbose_name='Imágen')
    tiempo_registro = models.DateTimeField(auto_now=True)
    usuario = models.ForeignKey(User)

    def __unicode__(self):
        return self.titulo

class Comentario(models.Model):
    receta = models.ForeignKey(Receta)
    texto = models.TextField(help_text='Tu comentario', verbose_name='Comentario')

    def __unicode__(self):
        return self.texto
```

Una vez que tenemos listo el models.py, reiniciamos el modelo (si no recuerdas como reiniciar revisa el capítulo: El modelo de datos):

```
python manage.py reset principal
```

Una vez con los modelos listos, es hora de probar el intérprete interactivo (shell). No olvidar que la siguiente instrucción se debe lanzar desde el directorio donde se encuentra el archivo manage.py:

```
python manage.py shell
```

Una vez dentro del shell, empezamos a importar los modelos, para ello digitamos:

```
from principal.models import Receta, Comentario
```

Esta instrucción importa todo el modelo al shell. Aquí es donde podemos consultar el contenido de los modelos y otras consultas, la sintaxis es del tipo:

```
Nombre_de_modelo.objects.metodo()
```

1 http://neosergio.github.com/recetario_mdw/

Donde podemos reemplazar Nombre_de_modelo y método por las diversas opciones que están en el cheat sheet (chuleta):

```
>>> from principal.models import Receta, Comentario
>>> Receta.objects.all()
[]
>>> Comentario.objects.all()
[]
>>>
```

En la imagen vemos como resultado una lista vacía [ ], entonces vamos a agregar elementos, para ello también necesitamos al modelo usuario que ya lo tiene Django, una vez que lo importamos sacamos al usuario cuya llave primaria (pk) sea 1.

```
>>> from django.contrib.auth.models import User
>>> primer = User.objects.get(pk=1)
>>>
```

Ahora si introducimos datos a los modelos: (la r es solo una variable cualquiera):

```
>>> r = Receta.objects.create(
... titulo='Huevo Frito',
... ingredientes='Huevo, Sal, Aceite',
... preparacion='Calentar la sarten y bla bla...',
... imagen='',
... usuario=primer)
>>>
```

Y comprobamos que se ha registrado:

```
>>> Receta.objects.all()
[<Receta: Huevo Frito>]
>>>
```

Agregamos un comentario a la receta anterior: (la c es solo una variable cualquiera)

```
>>> c = Comentario.objects.create(
... receta=r,
... texto='comida de campeones :P')
>>> Comentario.objects.all()
[<Comentario: comida de campeones :P>]
>>>
```

Agregamos una receta más: (Se puede escribir todo en una sola línea, yo lo hago para que se vea bien en la imagen):

```
>>> s = Receta.objects.create(
... titulo='Agua hervida',
... ingredientes='Agua',
... preparacion='Poner la tetera en bla bla...',
... imagen='',
... usuario=primer)
>>> Receta.objects.all()
[<Receta: Huevo Frito>, <Receta: Agua hervida>]
>>>
```

Sigamos jugando, ahora mostraremos las recetas cuyos ingredientes no empiecen con la letra A: (notar que luego de ingredientes hay dos guiones bajos)

```
>>> Receta.objects.exclude(ingredientes__startswith='A')
[<Receta: Huevo Frito>]
>>>
```

Ahora las recetas, que mencionen en su preparación la palabra 'tetera':

```
>>> Receta.objects.filter(preparacion__contains='tetera')
[<Receta: Agua hervida>]
>>>
```

Y si queremos ordenar alfabéticamente por título de receta:

```
>>> Receta.objects.all().order_by('titulo')
[<Receta: Agua hervida>, <Receta: Huevo Frito>]
>>>
```

Ahora lo invertimos:

```
>>> Receta.objects.all().order_by('titulo').reverse()
[<Receta: Huevo Frito>, <Receta: Agua hervida>]
>>>
```

Mostramos los comentarios de cada receta:

```
>>> Comentario.objects.filter(receta=r)
[<Comentario: comida de campeones :P>]
>>> Comentario.objects.filter(receta=s)
[]
>>>
```

Ahora me gustaría actualizar el título 'Huevo Frito' a la de 'Huevito Frito':

```
>>> Receta.objects.filter(titulo='Huevo Frito').update(titulo='Huevito Frito')
1
>>> Receta.objects.all()
[<Receta: Huevito Frito>, <Receta: Agua hervida>]
>>>
```

Sigo agregando comentarios a la primera receta y deseo saber cuantos comentarios hay hasta el momento en ambas recetas:

```
>> d = Comentario.objects.create(
.. receta = r,
.. texto = 'Para pasar el hambre')
>> Comentario.objects.filter(receta=r).count(

>> Comentario.objects.filter(receta=s).count(

>>
```

Agrego un comentario a la segunda receta (un comentario troll) y luego lo elimino:

```
>>> e = Comentario.objects.create(
... receta = s,
... texto = 'que inutil receta')
>>> Comentario.objects.filter(receta=s)
[<Comentario: que inutil receta>]
>>> e.delete()
>>> Comentario.objects.filter(receta=s)
[]
>>>
```

Ahora deseo saber el nombre de usuario y correo electrónico del usuario que agregó la receta con el título exacto de 'Agua hervida':

```
>>> Receta.objects.get(titulo='Agua hervida').usuario.username
u'sergio'
>>> Receta.objects.get(titulo='Agua hervida').usuario.email
u'raulsergio9@gmail.com'
>>>
```

Como verán hacer consultas en Django es simple, sin embargo siempre es bueno tener la documentación a la mano[1], ya que recordar tantos métodos puede ser tedioso al principio.

1 https://docs.djangoproject.com/en/dev/ref/models/querysets/

NOTA:
Las consultas que se realizan en el Shell no se guardan en el proyecto, es por ello que si revisan el repositorio del proyecto de ejemplo[1], sólo encontrarán cambios en models.py y admin.py (retirando la clase Bebida, que ya no es necesaria).

Las consultas son el paso previo a trabajar con las vistas y las plantillas, nos permitirán entregar a los usuarios del proyecto, la información que requieran. Practica tus propias consultas, juega con los modelos, diviértete!

1 http://neosergio.github.com/recetario_mdw/

CAPÍTULO 6

LAS VISTAS

Un función de vista o una vista, como es conocida generalmente, es una función en Python que hace una solicitud Web y devuelve una respuesta Web, esta respuesta puede ser el contenido de una página, un error 404, una imagen, un documento XML, entre muchas cosas más.

La vista contiene toda la lógica necesaria para devolver una respuesta, todas estas respuestas se encuentran en un único archivo y este archivo se llama: views.py, que se encuentra dentro de cada aplicación de Django.

LAS CONSULTAS Y LAS VISTAS

En el capítulo anterior, jugamos con algunas consultas, estas consultas son parte fundamental de las vistas, permiten elegir qué tipo de contenido se visualizará. Revisa y practica, si no lo hiciste, para poder obtener mejores resultados.

LAS PLANTILLAS Y LAS VISTAS

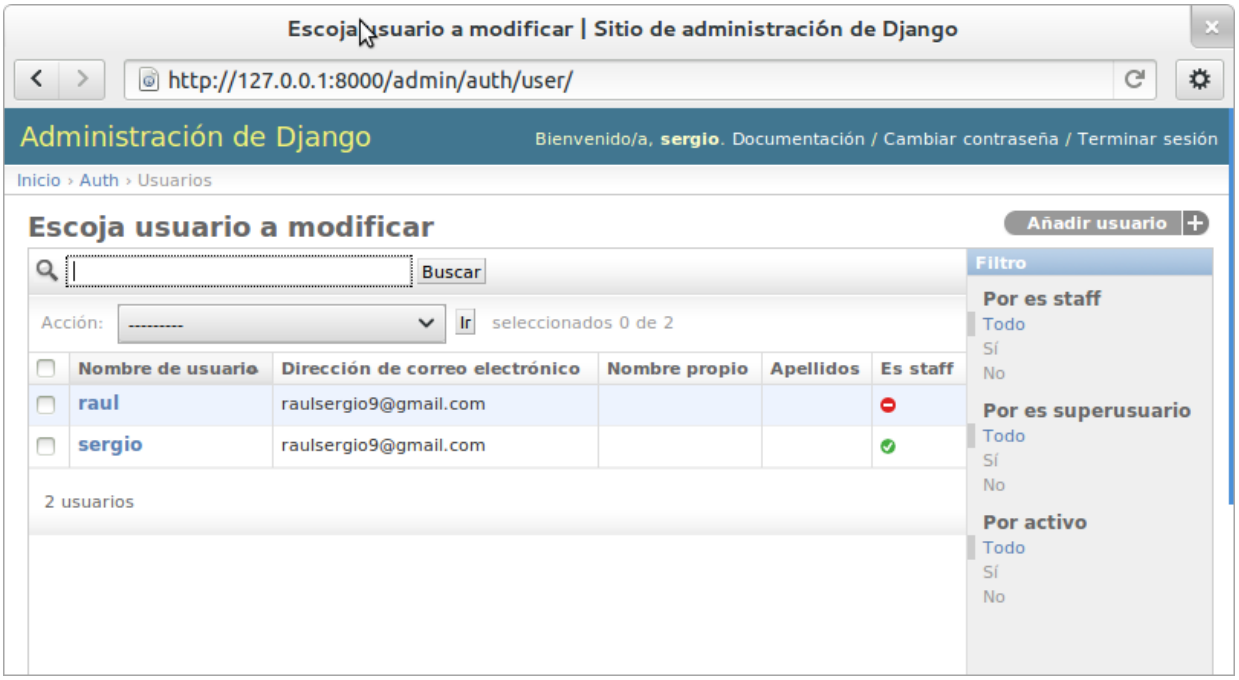
Las plantillas son muy importantes, permiten acomodar el resultado que devuelve la vista. Django tiene un estupendo motor de plantillas que permite separar eficientemente la presentación, de la lógica de programación, esta semana trabajaremos con algunas plantillas simples para no causar confusión.

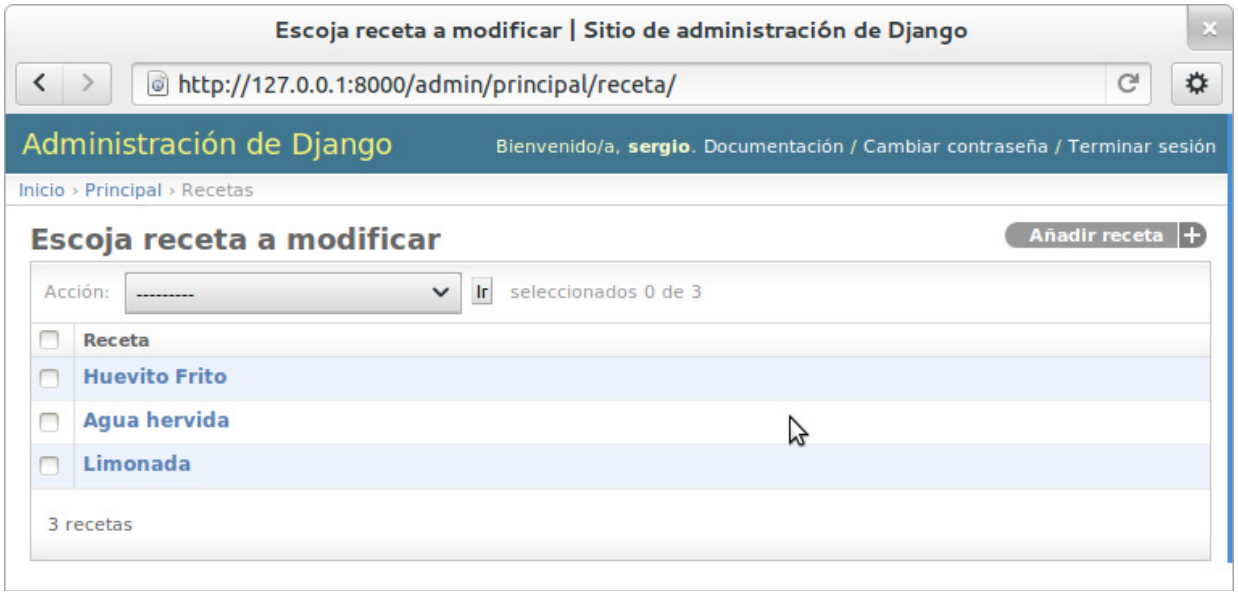
PROYECTO DE EJEMPLO

Debemos tener en cuenta que nuestros modelos almacenados en models.py dentro de la aplicación principal, quedaron así:

```
#encoding:utf-8
from django.db import models
from django.contrib.auth.models import User

class Receta(models.Model):
    titulo = models.CharField(max_length=100, verbose_name='Título', unique=True)
    ingredientes = models.TextField(help_text='Redacta los ingredientes')
    preparacion = models.TextField(verbose_name='Preparación', help_text='El proceso de preparación')
    imagen = models.ImageField(upload_to='recetas', verbose_name='Imágen')
    tiempo_registro = models.DateTimeField(auto_now=True)
    usuario = models.ForeignKey(User)

    def __unicode__(self):
        return self.titulo

class Comentario(models.Model):
    receta = models.ForeignKey(Receta)
    texto = models.TextField(help_text='Tu comentario', verbose_name='Comentario')

    def __unicode__(self):
        return self.texto
```

Rellenamos algunos datos para obtener resultados:





Con estos datos empezaremos a hacer algunas vistas para nuestro proyecto.

Iniciaremos con las importaciones necesarias, que son estas:

```
from principal.models import Receta, Comentario
from django.contrib.auth.models import User
from django.shortcuts import render_to_response, get_object_or_404
```

La primera de ellas es porque necesitamos los modelos, la segunda línea para usar los datos del modelo usuario, la tercera línea es la respuesta más simple (`HttpResponse`), la cuarta importa: `render_to_response` (para las plantillas) y `get_object_or_404` (para lanzar un error 404, de no encontrar ningún valor almacenado) y la quinta línea importa el `RequestContext` (en este caso para poder usar la ruta de las imágenes almacenadas previamente en la carpeta carga y referenciadas por la url media).

Ahora pasemos a ver donde usamos específicamente cada uno de estos componentes.

VISTA: SOBRE

Esta vista es un ejemplo muy simple del uso de HttpResponse, está devolviendo el contenido HTML:

```
def sobre(request):
    html = "<html><body>Proyecto de ejemplo en MDW</body></html>"
    return HttpResponse(html)
```

> **NOTA:**
> Vamos a modificar urls.py, reescribir casi por completo, si eres impaciente y quieres ver como queda al final urls.py o no entiendes muy bien como hacerlo, puedes echarle un vistazo[1].

Debemos tener en cuenta siempre, acompañar una vista de su configuración en urls.py, así que en este archivo debemos tener esta línea: (ubicada dentro de patterns)

```
url(r'^sobre/$','principal.views.sobre'),
```

Lanzamos nuestro servidor de desarrollo:(como en todos los capítulos)

```
python manage.py runserver
```

Y vamos a la dirección:

`http://127.0.0.1:8000/sobre/`

1 http://www.maestrosdelweb.com/editorial/curso-django-las-vistas/#urlspy

VISTA: INICIO

```
def inicio(request):
    recetas = Receta.objects.all()
    return render_to_response('inicio.html',{'recetas':recetas})
```

La vista inicio, hace una consulta de todas las recetas y las almacena en una variable llamada recetas (también), es aquí donde es apropiado usar `render_to_response` para indicarle que vamos a usar la plantilla inicio.html y que le pasaremos en un diccionario (concepto de python), el resultado de la consulta.

La plantilla `inicio.html` debemos crearla en la carpeta 'plantillas' que se encuentra en 'recetario' (creada previamente en capítulos anteriores), su contenido debe ser similar al siguiente:

```
<header>
    <nav>
        <ul>
            <li><a href='/'>Inicio</a></li>
            <li><a href='/usuarios'>Usuarios registrados</a></li>
            <li><a href='/recetas'>Recetas</a></li>
            <li><a href='/sobre'>Sobre</a></li>
        </ul>
    </nav>
</header>
<section>
    <h1>Recetario ;)</h1>
    <p>Actualmente tenemos registrados:</p>
    <ul>
    {% for dato in recetas %}
    <li>{{dato.titulo}}</li>
    {% endfor %}
    </ul>
</section>
```

Veremos unos símbolos `{% %}` que no deben distraernos, así que no entraré en detalles al respecto, pueden intuir de qué trata, pero serán explicados en el siguiente capítulo. Igual que en el ejemplo anterior debemos agregar la siguiente línea en nuestra configuración de las url en `urls.py`:

```
url(r'^$','principal.views.inicio'),
```

Nuestro servidor debe estar funcionando y podemos ingresar a la siguiente dirección:

`HTTP://127.0.0.1:8000/`

VISTA: USUARIOS

Esta vista saca los datos de los usuarios y recetas registradas. Repite el mismo procedimiento de la vista anterior.

```
def usuarios(request):
    usuarios = User.objects.all()
    recetas = Receta.objects.all()
    return render_to_response('usuarios.html',{'usuarios':usuarios,'recetas':recetas})
```

La plantilla `usuarios.html` debe ser creada en la carpeta plantillas y debe tener un contenido similar al siguiente:

```
<header>
    <nav>
        <ul>
            <li><a href='/'>Inicio</a></li>
            <li><a href='/usuarios'>Usuarios registrados</a></li>
            <li><a href='/recetas'>Recetas</a></li>
            <li><a href='/sobre'>Sobre</a></li>
        </ul>
    </nav>
</header>
```

```
<section>
<h1>Colaboradores</h1>
<ul>
    {% for dato in usuarios %}
    <li>{{dato.username}} - <em>{{dato.email}}</em>
        <ul>
            {% for item in recetas %}
                {% if item.usuario == dato %}
                    <li>{{item}}</li>
                {% endif %}
            {% endfor %}
        </ul>
    </li>
    {% endfor %}
</ul>
</section>
```

En esta plantilla se estan juntando a los usuarios y sus respectivas recetas, agregamos la siguiente regla al `urls.py`:

```
url(r'^usuarios/$','principal.views.usuarios'),
```

para ver el resultado (servidor corriendo), debemos entrar a:

`http://127.0.0.1:8000/usuarios/`

VISTA: LISTA_RECETAS

```
def lista_recetas(request):
    recetas = Receta.objects.all()
    return render_to_response('recetas.html',{'datos':recetas}, context_instance=RequestContext(request))
```

Esta vista, se diferencia de las anteriores por el uso de context_instance=RequestCon text(request), usada para indicarle a la plantilla el parámetro `{{MEDIA_URL}}` que refe- rencia a la ruta de las imágenes, la plantilla recetas.html, debe quedar así:

```
<style>
    img { width:100px; }
</style>
<header>
    <nav>
        <ul>
            <li><a href='/'>Inicio</a></li>
            <li><a href='/usuarios'>Usuarios registrados</a></li>
            <li><a href='/recetas'>Recetas</a></li>
            <li><a href='/sobre'>Sobre</a></li>
        </ul>
    </nav>
</header>
<section>
<h1>Nuestras recetas</h1>
<ul>
    {% for dato in datos %}
    <li>
        <a href='/receta/{{dato.id}}'>{{dato.titulo}}</a><br>
        <img src='{{MEDIA_URL}}{{dato.imagen}}'>
    </li>
    {% endfor %}
</ul>
</section>
```

La línea en urls.py debe ser así:

```
url(r'^recetas/$','principal.views.lista_recetas'),
```

Y se deben visualizar los resultados en:

http://127.0.0.1:8000/recetas/

VISTA: DETALLE_RECETA

```
def detalle_receta(request, id_receta):
    dato = get_object_or_404(Receta, pk=id_receta)
    comentarios = Comentario.objects.filter(receta=dato)
    return render_to_response('receta.html',{'receta':dato,'comentarios':comentarios}, context_instance=RequestContext(request))
```

VISTA DETALLE_RECETA

Esta vista recibe un parámetro más: `id_receta`, que le sirve para hacer una consulta del detalle de esa receta (corresponde a su llave primaria) y usa `get_object_or_404(Receta, pk=id_receta)`, para obtener un objeto o un error 404 en caso de no encontrar resultado alguno.

Con este resultado, se realiza una consulta para obtener sus comentarios. La plantilla `receta.html`, quedará así:

```
<header>
    <nav>
        <ul>
            <li><a href='/'>Inicio</a></li>
            <li><a href='/usuarios'>Usuarios registrados</a></li>
            <li><a href='/recetas'>Recetas</a></li>
            <li><a href='/sobre'>Sobre</a></li>
        </ul>
    </nav>
</header>
<section>
    <h1>{{receta.titulo}}</h1>
    <div id='ingredientes'>
        <h2>Ingredientes</h2>
        <p>{{receta.ingredientes}}</p>
    </div>
    <div id='preparacion'>
        <h2>Preparación</h2>
        <p>{{receta.preparacion}}</p>
    </div>
    <div id='referencia'>
        <img src='{{MEDIA_URL}}{{receta.imagen}}'>
    </div>
    <div id='comentarios'>
        <h2>Comentarios</h2>
            {% for item in comentarios %}
                <p>{{item.texto}}</p>
            {% empty %}
                <p>Sin comentarios registrados</p>
            {% endfor %}
    </div>
</section>
<footer>
    <p>Registrado {{receta.tiempo_registro}} por {{receta.usuario}}</p>
</footer>
```

La línea en `urls.py` debe ser así: (notar las expresiones regulares en python)

```
url(r'^receta/(?P<id_receta>\d+)$','principal.views.detalle_receta'),
```

Y se deben visualizar los resultados al darle clic en los enlaces a recetas en (ver siguiente página):

http://127.0.0.1:8000/recetas/

ARCHIVO FINAL: VIEWS.PY

Al final de todo, `views.py` debe quedar de esta manera:

```
from principal.models import Receta, Comentario
from django.contrib.auth.models import User
from django.http import HttpResponse
from django.shortcuts import render_to_response, get_object_or_404
from django.template import RequestContext


def sobre(request):
    html = "<html><body>Proyecto de ejemplo en MDW</body></html>"
    return HttpResponse(html)

def inicio(request):
    recetas = Receta.objects.all()
    return render_to_response('inicio.html',{'recetas':recetas})

def usuarios(request):
    usuarios = User.objects.all()
    recetas = Receta.objects.all()
    return render_to_response('usuarios.html',{'usuarios':usuarios,'recetas':recetas})

def lista_recetas(request):
    recetas = Receta.objects.all()
    return render_to_response('recetas.html',{'datos':recetas}, context_instance=RequestContext(request))

def detalle_receta(request, id_receta):
    dato = get_object_or_404(Receta, pk=id_receta)
    comentarios = Comentario.objects.filter(receta=dato)
    return render_to_response('receta.html',{'receta':dato,'comentarios':comentarios}, context_instance=RequestContext(request))
```

ARCHIVO: URLS.PY

El archivo `urls.py`, permite tener unas urls limpias, para nuestro ejemplo este debe quedar de la siguiente manera: (si fuiste impaciente, regresa a la vista: sobre)

```
from django.conf.urls import patterns, include, url
from django.contrib import admin
from django.conf import settings

admin.autodiscover()

urlpatterns = patterns('',
    url(r'^$','principal.views.inicio'),
    url(r'^usuarios/$','principal.views.usuarios'),
    url(r'^recetas/$','principal.views.lista_recetas'),
    url(r'^receta/(?P<id_receta>\d+)$','principal.views.detalle_receta'),
    url(r'^sobre/$','principal.views.sobre'),
    url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^media/(?P<path>.*)$','django.views.static.serve',
        {'document_root':settings.MEDIA_ROOT,}
    ),
)
```

URLS.PY

Para finalizar, si se perdieron con alguna línea o si desean corroborar que todo va bien, pueden estar pendientes del repositorio del proyecto[1].

1 http://neosergio.github.com/recetario_mdw/

CAPÍTULO 7

LAS PLANTILLAS

Django posee un componente conocido como “el motor de plantillas”, este motor brinda un poderoso mini-lenguaje para definir detalles de la capa de la aplicación, que visualizará el usuario. Esto refuerza la separación de la lógica de programación y de presentación.

Las plantillas pueden ser desarrolladas y mantenidas por cualquier persona con un poco de conocimiento de HTML y lógica común. No necesita fundamentos de Python. Puedes revisar la documentación con respecto al lenguaje de plantillas[1].

ETIQUETAS Y FILTROS

Es necesario para aplicar las plantillas con Django, conocer más sobre su propio mini-lenguaje de plantillas. Para ello tenemos las chuletas[2] de la siguiente página.

Les hago recordar que la revisión de la documentación oficial es muy importante. Están divididas en etiquetas y filtros predefinidos[3] y el lenguaje de plantillas de Django para programadores de Python[4].

LAS PLANTILLAS DE NUESTRO PROYECTO

En el capítulo anterior, trabajamos con algunas plantillas simples, sin entrar en detalle de su funcionamiento, en este capítulo modificaremos las plantillas, utilizaremos algunos filtros y etiquetas predeterminadas, obviamente veremos a detalle de que tratan.

> **NOTA:**
> Las plantillas están almacenadas en la carpeta configurada en el settings.py.

1 https://docs.djangoproject.com/en/1.4/topics/templates/
2 http://www.maestrosdelweb.com/images/2012/05/django-templates.pdf
3 https://docs.djangoproject.com/en/1.4/ref/templates/builtins/
4 https://docs.djangoproject.com/en/1.4/ref/templates/api/

Plantillas en Django

Etiquetas	Filtros
{% autoescape on %} {% endautoescape %} {% block nombre %} {% endblock%} {% comment %} {% endcomment %} {% csrf_token %} {% cycle 'elemento1' 'elemento2' %} {% debug %} {% extends 'base.html' %} {% filter force_escape|lower %} {% endfilter %} {% firstof variable1 variable2 variable 3 %} {% for i in lista %} {% empty %} {% endfor %} {{ forloop.counter }} {{ forloop.counter0 }} {{ forloop.revcounter }} {{ forloop.revcounter0 }} {{ forloop.first }} {{ forloop.last }} {{ forloop.parentloop }} {% if condicion %} {% elif condicion %} {% else %} {% endif %} {% ifchanged %} {% else %} {% endifchanged %} {% ifequal %} {% endifequal %} {% ifnotequal %}{% endifnotequal %} {% include 'otra_plantilla.html' %} {% load %} {% now %} {% regroup %} {% spaceless %} {% endspaceless %} {% ssi %} {% templatetag openblock / closeblock / openvariable / closevariable / openbrace / closebrace / opencomment / closecomment %} {% url %} {% widthradio %} {% with %} {% endwith %}	add addslashes capfirst center cut date default default_if_none dictsort dictsortreversed divisibleby escape escapejs filesizeformat first fix_ampersands floatformat force_escape get_digit iriencode join last length length_is linebreaks linebreaksbr linenumbers ljust lower make_list phone2numeric pluralize pprint random removetags rjust safe safeseq slice slugify stringformat striptags time timesince timeuntil title truncatechars truncatewords unordered_list upper urlencode urlize urlizetrunc wordcount wordwrap yesno

Elaborado por: **Sergio Infante (@neosergio)**

Formatos de Fecha

Formato	Descripción	Ejemplo
a	'a.m' o 'p.m'	'a.m.'
A	'AM' o 'PM'	'AM'
b	Mes, textual, 3 letras en minúsculas	'may'
B	NO ESTA IMPLEMENTADO	
c	Formato ISO 8601	2012-05-20T21:01:56.447370-05:00
d	Día del mes, 2 dígitos con ceros adelante	'01' al '31'
D	Día de la semana, textual, 3 letras	'dom'
e	El nombre de la zona horaria.	'PET'
E	Mes, utilizado generalmente para representación de datos largos y en formato local	'mayo'
f	Tiempo en formato de 12 horas y minutos. Extensión propietaria	'9:01'
F	Mes, textual, largo	'mayo'
g	Hora, formato de 12 horas sin ceros	'1' a '12'
G	Hora, formato de 24 horas sin ceros	'1' a '23'
h	Hora, formato de 12 horas	'01' a '12'
H	Hora, formato de 24 horas	'01' a '23'
i	Minutos	'00' a '59'
I (i mayuscula)	NO ESTA IMPLEMENTADO	
j (j minuscula)	Día del mes, sin ceros	'1' a '31'
l (ele minuscula)	Día del mes, textual, largo	'domingo'
L	Verdadero o Falso, para saber si el año es bisiesto	'True'
m	Mes, 2 digitos con ceros	'01' a '12'
M	Mes, textual, 3 letras	'May'
n	Mes sin ceros	'1' a '12'
N	Mes abreviado. Extensión propietaria.	'mayo'
o	ISO 8601	'2012'
O	Diferencia a la hora de Greenwich	'-0500'
P	Tiempo, en formato 12 horas. Extensión propietaria	'9:05 p.m.'
r	Formato de fecha tipo RFC 2822	'dom, 20 May 2012 21:05:07 -0500'
s	Segundos, 2 dígitos con ceros	'00' a '59'
S	Sufijo ordinario ingles para el día del mes	'th'
t	Número de días en el mes dado	'28' a '31'
T	Zona horaria	'PET'
u	Microsegundos	0 a 999999
U	Segundos desde la epoca UNIX (1 de enero de 1970 00:00:00 UTC)	1337565907
w	Día de la semana, dígitos sin ceros	0 (domingo) a 6 (sabado)
W	ISO 8601, número de la semana del año, con las semanas iniciando lunes	1, 53
y	Año, con 2 dígitos	12
Y	Año, con 4 dígitos	2012
z	Días del año	0 al 365
Z	La zona horario en segundos. El desplazamiento para zonas horarias al oeste de UTC es siempre negativo, y para aquellos al este de UTC es siempre positivo	-18000
DATE_FORMAT	Formato predeterminado	20 de mayo de 2012
DATETIME_FORMAT	Formato predeterminado	20 de mayo de 2012 a las 21:05
SHORT_DATE_FORMAT	Formato predeterminado	20/05/2012
SHORT_DATETIME_FORMAT	Formato predeterminado	20/05/2012 21:05

BASE.HTML

Esta es una nueva plantilla, no habíamos utilizado anteriormente alguna de este tipo, esta plantilla base, hace exactamente lo que su nombre sugiere, sirve de base para las demás.

```
<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='utf-8'>
  <title>Recetario para MDW - {% block titulo %}{% endblock %}</title>
  <style>
  {% block style_css %}{% endblock %}
  </style>
</head>
<body>
  <header>
    {% block encabezado %}{% endblock %}
    <nav>
      <ul>
        <li><a href='/'>Inicio</a></li>
        <li><a href='/usuarios'>Usuarios registrados</a></li>
        <li><a href='/recetas'>Recetas</a></li>
        <li><a href='/sobre'>Sobre</a></li>
      </ul>
    </nav>
    <hr>
  </header>
  <section class='contenido'>
    {% block contenido %}{% endblock %}
  </section>
  <footer>
    <p>Proyecto de ejemplo para el curso Django de Maestros del Web &copy;2012</p>
  </footer>
</body>
</html>
```

En esta plantilla es importante notar, que se encuentra la estructura principal de toda la aplicación. Además se usa la etiqueta `{% block _____ %} {% endblock %}` (reemplaza _____ con el nombre del bloque), esta etiqueta se reemplazará con contenido en cada una de las plantillas que restan. Por ahora agregaremos CSS dentro de la etiquetación HTML, no lo hacemos todavía desde un archivo CSS porque esto lo veremos con mayor detalle en el capítulo de archivos estáticos.

INICIO.HTML

Esta es la plantilla que estamos usando para la página de inicio, y debe quedar con las siguientes modificaciones:

```
{% extends 'base.html' %}

{% comment %} Aqui van comentarios {% endcomment %}

{% block titulo %} Inicio {% endblock %}

{% block style_css %}
  .impar { background-color:#90EE90; }
  .par { background-color:#5ACC5A; }
{% endblock %}

{% block encabezado %}
    <h1>Recetario ;) </h1>
{% endblock %}

{% block contenido %}
  <p>
      Actualmente tenemos registradas:
     {% with total=recetas.count %}
        {{total}} receta{{total|pluralize}}
     {% endwith %}
  </p>
    <ul>
      {% for dato in recetas %}
      <li class="{% cycle 'impar' 'par' %}">{{dato.titulo}}</li>
      {% empty %}
      <li>No hay recetas registradas aún.</li>
      {% endfor %}
    </ul>
{% endblock %}
```

INICIO.HTML

- **{% extends 'base.html' %},** esta línea permite importar la plantilla base.html y reemplazar cada vez que encuentre los bloques, con los de esta plantilla.
- **{% comment %}{% endcomment %}** es un ejemplo de como pueden incluirse comentarios en el lenguaje de plantillas de Django, todo lo que se encuentre dentro de estas etiquetas, será ignorado.
- **{% with %}** permite poner la cantidad de elementos de alguna respuesta y puede ser usado para mostrar palabras en plurales, dependiendo de la circunstancia.
- **{% for dato in recetas %}** permite acceder a cada receta en particular.
- **{% cycle 'impar' 'par' %}** permite iterar con los valores 'impar' o 'par', que pueden verse afectados con algunas reglas en CSS simples que se encuentran en **{% block style_css %}**
- **{% empty %}** es usado en caso de que no existan elementos en las recetas.

USUARIOS.HTML

La plantilla que muestra los usuarios registrados:

```
{% extends 'base.html' %}
{% block titulo %} Usuarios registrados {% endblock %}

{% block encabezado %}
  <h1>Colaboradores</h1>
{% endblock %}

{% block contenido %}
<ul>
    {% for dato in usuarios %}
    <li>{{dato.username|capfirst}} - <em>{{dato.email}}</em>
        <ul>
            {% for item in recetas %}
                {% if item.usuario == dato %}
                    <li>{{item}}</li>
                {% endif %}
            {% empty %}
                <li>Este usuario aún no ha registrado recetas.</li>
            {% endfor %}
        </ul>
    </li>
    {% empty %}
    <li>No hay usuarios registrados, aún.</li>
    {% endfor %}
</ul>
{% endblock %}
```

USUARIOS.HTML

Además de las etiquetas vistas anteriormente, aquí podemos encontrar:

- **{{dato.username|capfirst}}** esto se usa para capitalizar el nombre del usuario (primera letra en mayúscula).
- **Bloque {% if %}{% endif %}** para comprobar que receta pertenece al usuario actual del bucle **{% for %}**

RECETAS.HTML

Esta plantilla nos muestra la lista de recetas registradas:

```
{% extends 'base.html' %}
{% block titulo %} Recetas registradas {% endblock %}

{% block css %}
  <style>
    img { width:100px; }
  </style>
{% endblock %}


{% block encabezado %}
  <h1>Nuestras recetas</h1>
{% endblock %}

{% block contenido %}
<ul>
    {% for dato in datos %}
    <li>
        <a href='{% url principal.views.detalle_receta dato.id %}'>{{dato.titulo}}</a>
        <img src='{{MEDIA_URL}}{{dato.imagen}}'>
    </li>
    {% empty %}
    <li>Aún no hay recetas registradas</li>
    {% endfor %}
</ul>
{% endblock %}
```

{% url principal.views.detalle_receta dato.id %} esta etiqueta, trabaja con views.py (ubica una vista y le pasa un parámetro), se traduce en/receta/dato.id, donde dato.id es reemplazado por la id de la receta del elemento actual del bucle **{% for %}**.

RECETA.HTML

Esta plantilla se usará para mostrar los detalles de cada receta registrada:

```
{% block titulo %} {{receta.titulo}} {% endblock %}

{% block encabezado %}
  <h1>{{receta.titulo|title}}</h1>
{% endblock %}

{% block contenido %}
    <div id='ingredientes'>
        <h2>Ingredientes</h2>
        <p>{{receta.ingredientes}}</p>
    </div>
    <div id='preparacion'>
        <h2>Preparación</h2>
        <p>{{receta.preparacion}}</p>
    </div>
    <div id='referencia'>
        <img src='{{MEDIA_URL}}{{receta.imagen}}'>
    </div>
    <div id='comentarios'>
        <h2>Comentarios</h2>
            {% for item in comentarios %}
                <p>{{item.texto}}</p>
            {% empty %}
                <p>Sin comentarios registrados</p>
            {% endfor %}
    </div>
    <footer>
      <p>Registrado {{ receta.tiempo_registro|date:'SHORT_DATETIME_FORMAT' }} por {{receta.usuario}}</p>
    </footer>
{% endblock %}
```

Además de todas las etiquetas usadas y ya explicadas anteriormente se puede notar:

- **{{receta.titulo|title}}** transforma los títulos de la receta en formato título (la primera letra de cada palabra en mayúscula.
- **{{ receta.tiempo_registro|date:'SHORT_DATETIME_FORMAT' }}** configura la impresión de fecha como: dd/mm/aaaa hh:mm (ejemplo: 20/05/2012 21:05), se puede ver mas formatos de fecha a usar en los cheatsheets de arriba.

PRACTICA Y PRACTICA

La mejor forma de aprender Django es practicando, así que supongo que lo estas haciendo, durante todos los capítulos, ahora toca que crees tus propias plantillas, para tus propias consultas y vistas, que seguramente ya programaste con ayuda del curso de Django.

No te olvides de revisar la documentación[1] por si buscas más ejemplos y detalles de las diversas etiquetas, es la mejor fuente para aprender más sobre este estupendo framework. Puedes leer también: Cómo se utiliza Python y Django en Mejorando.la [2]y si el inglés no es problema para tí, puedes ver una charla del PyCon 2012 sobre las plantillas en Django[3].

1 https://docs.djangoproject.com/en/1.4/
2 http://www.maestrosdelweb.com/editorial/python-django-mejorandola/
3 http://www.youtube.com/watch?v=ahM4GBZ-6qg&feature=player_embedded

CAPÍTULO 8

LOS FORMULARIOS

Los formularios permiten el ingreso de datos para su procesamiento, ya sea para crear nuevos contenidos, para modificar el contenido que ya esta registrado previamente y hasta para realizar búsquedas.

Django posee una interfaz administrativa que hemos estado utilizando, pero cuando nos encontramos en la posición de elaborar interfaces de entrada o edición de datos, para los usuarios de nuestra aplicación, en algunas circunstancias, darles acceso al administrador de Django resulta contraproducente. Es por ello que resulta mejor preparar nuestros formularios para manejar estas circunstancias, entre otras ocasiones.

Para entender mejor lo que Django permite, veamos esta lista de sus 'superpoderes':

- Mostrar un formulario HTML generado a partir de reglas básicas.
- Generar formularios HTML validados a partir de modelos ya declarados.
- Validar la información que se desea registrar a través del formulario.
- Mostrar nuevamente el formulario, haciendo notar los errores que ha producido la validación.
- Convertir la información subida en tipos de datos de Python, para procesarlos de acuerdo a las vistas.

Para el ejemplo de este capítulo, usaremos tres formularios, dos de ellos generados a partir de modelos y uno sin relación a ningún modelo, pero con la capacidad de procesar la información y enviar un correo electrónico, utilizando gmail.

> **NOTA:**
> Para crear formularios se usa por convención un archivo nuevo llamado: `forms.py` que se ubicará en la carpeta de la aplicación, esa misma donde se encuentran los archivos: `models.py` y `views.py`. Sin embargo esto no es obligatorio, pueden crearse también en el archivo `models.py`

Creamos nuestro archivo `forms.py` y en las primeras líneas ponemos:

```
#encoding:utf-8
from django.forms import ModelForm
from django import forms
from principal.models import Receta, Comentario
```

Estas líneas permitirán usar los elementos ya construidos (ModelForm para usar los modelos ya declarados, forms para declarar nuevas reglas para un formulario y los modelos de nuestra aplicación principal).

HOJA DE TRUCOS

En este capítulo también tenemos una chuleta[1] (cheat sheet) para el manejo de formularios con Django, esta contiene los campos que vienen ya construidos, los atributos y las opciones de validación.

Campos

BooleanField
CharField
max_length
min_length
ChoiceField
choices
TypedChoiceField
choices
coerce
empty_value
DateField
input_formats
DateTimeField
input_formats
DecimalField
max_value
min_value
max_digits
decimal_places
EmailField
FileField
FilePathField
path
recursive
match
allow_files
allow_folders
FloatField
ImageField
IntegerField
max_value
min_value
IPAddressField
GenericIPAddressField
MultipleChoiceField
choices
TypedMultipleChoiceField
choices
coerce
empty_value
NullBooleanField
RegexField
regex
SlugField
TimeField
input_formats
URLField
max_length
min_length

Campos ligeramente complejos

ComboField
fields
MultiValueField
fields
SplitDateTimeField
input_date_formats
input_time_formats

Argumentos comunes para los campos

required
label
initial
widget
help_text
error_messages
validators
localize

Widgets

Widget
attrs
TextInput
PasswordInput
render_value
HiddenInput
MultipleHiddenInput
choices
FileInput
ClearableFileInput
DateInput
format
DateTimeInput
format
TimeInput
format
Textarea
CheckboxInput
check_test
Select
choices
NullBooleanSelect
SelectMultiple
RadioSelect
CheckboxSelectMultiple
MultiWidget
widgets
SplitDateTimeWidget
date_format
time_format
SplitHiddenDateTimeWidget
SelectDateWidget
years

Campos que manejan relaciones

ModelChoiceField
queryset
empty_label
ModelMultipleChoiceField
queryset

Validadores
(también sirven para modelos)

RegexValidator
regex
message
code
URLValidator
validate_email
validate_slug
validate_ipv4_address
validate_ipv6_address
validate_ipv46_address
validate_comma_separated_integer_list
MaxValueValidator
max_value
MinValueValidator
min_value
MaxLengthValidator
max_length
MinLengthValidator
min_length

maestros del web

Con esta referencia en mano empezamos a crear nuestros formularios.

1 Elaborada por Sergio Infante (@neosergio)
http://www.maestrosdelweb.com/images/2012/06/django-forms.pdf

OBJETOS FORMULARIO

Un objeto formulario en Django es una secuencia de campos y reglas de validación, que permiten depurar la información requerida y procesarla eficientemente. Estos campos y reglas deben ser declarados en el orden que se desea que aparezcan. Las clases formulario son creadas como subclases de `django.forms.Form` y tienen un estilo de declaración muy similar a los modelos de Django.

Para nuestro ejemplo consideraremos implementar un formulario de contacto, que tendrá solo dos campos, un correo electrónico y un mensaje, debe lucir así:

```
class ContactoForm(forms.Form):
    correo = forms.EmailField(label='Tu correo electrónico')
    mensaje = forms.CharField(widget=forms.Textarea)
```

Este Formulario, debe ir de la mano de una vista que permita procesar los datos que reciba, ésta vista será declarada en `views.py`, primero debemos modificar nuestras primeras líneas para importar todo lo necesario, luego declarar una nueva vista.

```
from principal.models import Receta, Comentario
from principal.forms import RecetaForm, ComentarioForm, ContactoForm
from django.contrib.auth.models import User
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response, get_object_or_404
from django.template import RequestContext
from django.core.mail import EmailMessage
```

```
def contacto(request):
    if request.method=='POST':
        formulario = ContactoForm(request.POST)
        if formulario.is_valid():
            titulo = 'Mensaje desde el recetario de Maestros del Web'
            contenido = formulario.cleaned_data['mensaje'] + "\n"
            contenido += 'Comunicarse a: ' + formulario.cleaned_data['correo']
            correo = EmailMessage(titulo, contenido, to=['destinatario@email.com'])
            correo.send()
            return HttpResponseRedirect('/')
    else:
        formulario = ContactoForm()
    return render_to_response('contactoform.html',{'formulario':formulario}, context_instance=RequestContext(request))
```

En donde encontramos: `destinatario@email.com`, debe ser reemplazado por el email que recibirá el mensaje. Para que funcione el envío del mensaje al correo electrónico debemos, agregar al final de nuestro archivo `settings.py`, las siguientes configuraciones:

```
158  #Configuraciones para enviar mensajes usando gmail
159  EMAIL_USE_TLS = True
160  EMAIL_HOST = 'smtp.gmail.com'
161  EMAIL_HOST_USER = 'remitente@gmail.com'
162  EMAIL_HOST_PASSWORD = 'clavedelcorreo'
163  EMAIL_PORT = 587
```

En donde encontramos: `remitente@gmail.com`, debe ser reemplazado por la cuenta que enviará el mensaje (debe ser una cuenta válida en gmail). Necesitamos también una plantilla, en este caso se llamará: `contacto.html`

```
{% extends 'base.html' %}
{% block contenido %}
    <form id='formulario' method='post' action=''>{% csrf_token %}
    {{formulario.as_p}}
    <p><input type='submit' value='Confirmar'/></p>
    </form>
{% endblock %}
```

Para que nuestra aplicación sea navegable, modificamos `base.html`, para que quede con el siguiente menú:

```
<nav>
  <ul>
    <li><a href='/'>Inicio</a></li>
    <li><a href='/usuarios'>Usuarios registrados</a></li>
    <li><a href='/recetas'>Recetas</a></li>
    <li><a href="/comenta">Comenta</a></li>
    <li><a href='/sobre'>Sobre</a></li>
    <li><a href="/contacto">Contactanos</a></li>
  </ul>
</nav>
```

También es necesaria una URL que convoque a esta vista por lo tanto debemos agregar la siguiente línea dentro de nuestra configuración en `urls.py`:

```
url(r'^contacto/$','principal.views.contacto'),
```

FORMULARIOS A PARTIR DE MODELOS

Si se esta construyendo una aplicación que gestiona una base de datos, lo más apropiado es usar los modelos ya declarados como formularios y así evitar estar repitiendo las mismas reglas para procesar los datos.

Por esta razón, Django provee una clase de ayuda que permite crear un formulario a partir de un modelo, esta clase se llama ModelForm y se emplea así (`forms.py`):

```
class RecetaForm(ModelForm):
    class Meta:
        model = Receta

class ComentarioForm(ModelForm):
    class Meta:
        model = Comentario
```

Las vistas para manejar estos formularios serían estas:

```
def nueva_receta(request):
    if request.method=='POST':
        formulario = RecetaForm(request.POST, request.FILES)
        if formulario.is_valid():
            formulario.save()
            return HttpResponseRedirect('/recetas')
    else:
        formulario = RecetaForm()
    return render_to_response('recetaform.html',{'formulario':formulario}, context_instance=RequestContext(request))


def nuevo_comentario(request):
    if request.method=='POST':
        formulario = ComentarioForm(request.POST)
        if formulario.is_valid():
            formulario.save()
            return HttpResponseRedirect('/recetas')
    else:
        formulario = ComentarioForm()
    return render_to_response('comentarioform.html',{'formulario':formulario}, context_instance=RequestContext(request))
```

Notar que se usa `HttpResponseRedirect` para redireccionar a una URL. Las plantillas `recetaform.html` y `comentarioform.html` serían estas:

```
{% extends 'base.html' %}
{% block contenido %}
    <form id='formulario' method='post' enctype='multipart/form-data' action=''>{% csrf_token %}
        {{formulario.as_p}}
        <p><input type='submit' value='Confirmar'/></p>
    </form>
{% endblock %}
```

```
{% extends 'base.html' %}
{% block contenido %}
    <form id='formulario' method='post' action=''>{% csrf_token %}
        {{formulario.as_p}}
        <p><input type='submit' value='Confirmar'/></p>
    </form>
{% endblock %}
```

Y tendríamos que modificar recetas.html para agregar un enlace para la creación de nuevas recetas:

```
<a href=”/receta/nueva”>Agregar una receta</a>
```

La configuración para la URL es esta:

```
url(r’^receta/nueva/$’,’principal.views.nueva_receta’),
url(r’^comenta/$’,’principal.views.nuevo_comentario’),
```

RESULTADO FINAL

Los archivos: `forms.py`, `views.py`, `urls.py` y `base.html` quedan de la siguiente manera:

FORMS.PY

```python
#encoding:utf-8
from django.forms import ModelForm
from django import forms
from principal.models import Receta, Comentario

class ContactoForm(forms.Form):
    correo = forms.EmailField(label='Tu correo electrónico')
    mensaje = forms.CharField(widget=forms.Textarea)

class RecetaForm(ModelForm):
    class Meta:
        model = Receta

class ComentarioForm(ModelForm):
    class Meta:
        model = Comentario
```

VIEWS.PY

```python
from principal.models import Receta, Comentario
from principal.forms import RecetaForm, ComentarioForm, ContactoForm
from django.contrib.auth.models import User
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response, get_object_or_404
from django.template import RequestContext
from django.core.mail import EmailMessage

def sobre(request):
    html = "<html><body>Proyecto de ejemplo en MDW</body></html>"
    return HttpResponse(html)

def inicio(request):
    recetas = Receta.objects.all()
    return render_to_response('inicio.html',{'recetas':recetas})

def usuarios(request):
    usuarios = User.objects.all()
    recetas = Receta.objects.all()
    return render_to_response('usuarios.html',{'usuarios':usuarios,'recetas':recetas})

def lista_recetas(request):
    recetas = Receta.objects.all()
    return render_to_response('recetas.html',{'datos':recetas}, context_instance=RequestContext(request))

def detalle_receta(request, id_receta):
    dato = get_object_or_404(Receta, pk=id_receta)
    comentarios = Comentario.objects.filter(receta=dato)
    return render_to_response('receta.html',{'receta':dato,'comentarios':comentarios}, context_instance=RequestContext(request))
```

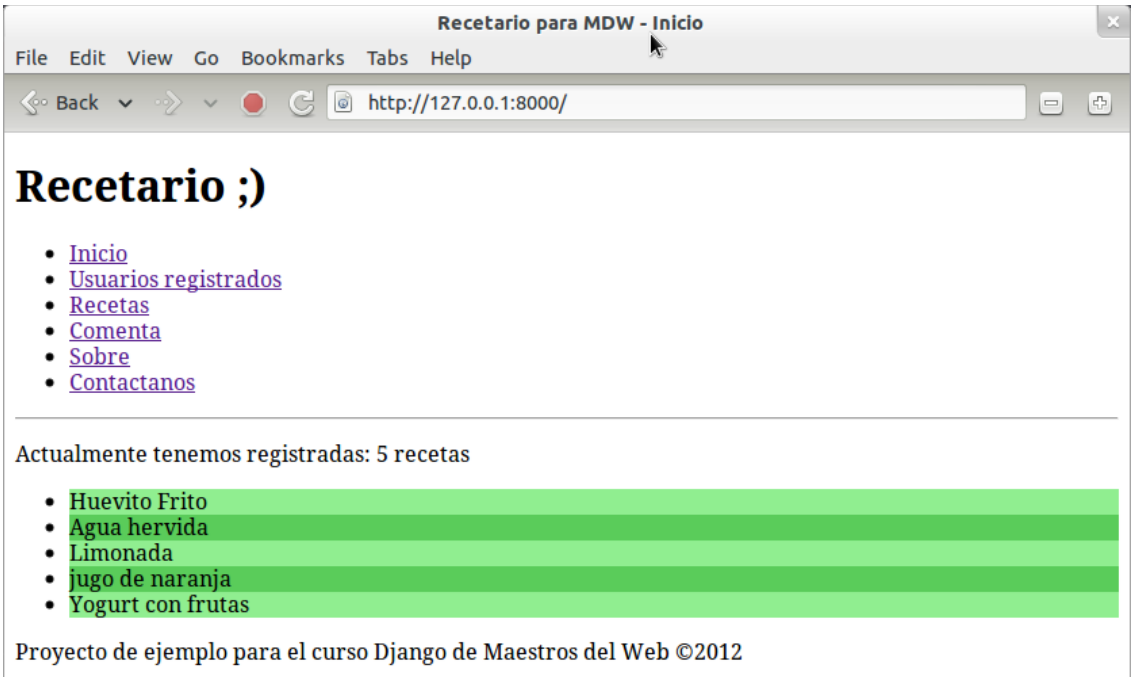
URLS.PY

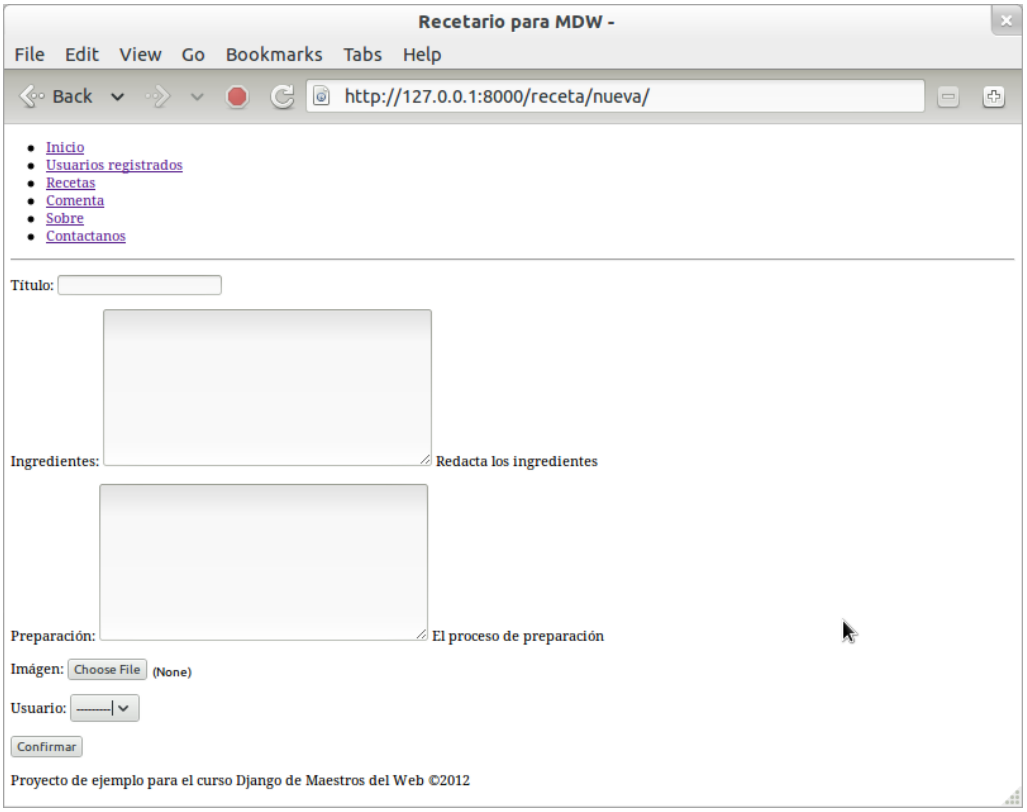
```python
from django.conf.urls import patterns, include, url
from django.contrib import admin
from django.conf import settings

admin.autodiscover()

urlpatterns = patterns('',
    url(r'^$','principal.views.inicio'),
    url(r'^usuarios/$','principal.views.usuarios'),
    url(r'sobre/$','principal.views.sobre'),
    url(r'^recetas/$','principal.views.lista_recetas'),
    url(r'^receta/(?P<id_receta>\d+)$','principal.views.detalle_receta'),
    url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^media/(?P<path>.*)$','django.views.static.serve',
        {'document_root':settings.MEDIA_ROOT,}
    ),
    url(r'^contacto/$','principal.views.contacto'),
    url(r'^receta/nueva/$','principal.views.nueva_receta'),
    url(r'^comenta/$','principal.views.nuevo_comentario'),
)
```

BASE.HTML

```html
<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='utf-8'>
  <title>Recetario para MDW - {% block titulo %}{% endblock %}</title>
  <style>
  {% block style_css %}{% endblock %}
  </style>
</head>
<body>
  <header>
  {% block encabezado %}{% endblock %}
  <nav>
    <ul>
      <li><a href='/'>Inicio</a></li>
      <li><a href='/usuarios'>Usuarios registrados</a></li>
      <li><a href='/recetas'>Recetas</a></li>
      <li><a href="/comenta">Comenta</a></li>
      <li><a href='/sobre'>Sobre</a></li>
      <li><a href="/contacto">Contactanos</a></li>
    </ul>
  </nav>
  <hr>
  </header>
  <section class='contenido'>
  {% block contenido %}{% endblock %}
  </section>
  <footer>
  <p>Proyecto de ejemplo para el curso Django de Maestros del Web &copy;2012</p>
  </footer>
</body>
</html>
```

Y debemos tener las siguientes interfaces, como resultado:




Recetario para MDW -
File Edit View Go Bookmarks Tabs Help
Back
http://127.0.0.1:8000/receta/nueva/
Inicio
Usuarios registrados
Recetas
Comenta
Sobre
Contactanos
Título:
Ingredientes:
Redacta los ingredientes
Preparación:
El proceso de preparación
Imágen:
Choose File
(None)
Usuario:
Confirmar
Proyecto de ejemplo para el curso Django de Maestros del Web ©2012


Recetario para MDW -
File Edit View Go Bookmarks Tabs Help
Back
http://127.0.0.1:8000/comenta/
Inicio
Usuarios registrados
Recetas
Comenta
Sobre
Contactanos
Receta:
Comentario:
Tu comentario
Confirmar
Proyecto de ejemplo para el curso Django de Maestros del Web ©2012
http://127.0.0.1:8000/comenta


Recetario para MDW -
File Edit View Go Bookmarks Tabs Help
Back
http://127.0.0.1:8000/contacto/
Inicio
Usuarios registrados
Recetas
Comenta
Sobre
Contactanos
Tu correo electrónico:
Mensaje:
Confirmar
Proyecto de ejemplo para el curso Django de Maestros del Web ©2012

DOCUMENTACIÓN OFICIAL DE DJANGO

La mejor forma de enterarse todo lo que puede hacer Django, es revisando su documentación, para el caso de formularios tiene: La API de formularios[1], Campos de formulario[2], Validación de campos y formularios[3], Formularios múltiples[4], Archivos para procesar Formularios[5], Asistente para formularios[6], entre otras cosas más[7]. Si deseas ir más allá con los formularios, pues este es tu punto de partida.

REPOSITORIO DEL PROYECTO Y VÍDEO COMPLEMENTARIO

Si te perdiste de algún detalle, o deseas ver como va quedando el código oficial de nuestro proyecto de ejemplo, no te olvides que siempre puedes revisar el repositorio del proyecto en github[8]. Por otro lado el vídeo Django Form Processing Deep Dive[9], nos da mayores referencias del procesamiento de formularios con Django.

1 https://docs.djangoproject.com/en/dev/ref/forms/api/
2 https://docs.djangoproject.com/en/dev/ref/forms/fields/
3 https://docs.djangoproject.com/en/dev/ref/forms/validation/
4 https://docs.djangoproject.com/en/dev/topics/forms/formsets/
5 https://docs.djangoproject.com/en/dev/topics/forms/media/
6 https://docs.djangoproject.com/en/dev/ref/contrib/formtools/form-wizard/
7 https://docs.djangoproject.com/en/dev/ref/forms/
8 http://neosergio.github.com/recetario_mdw/
9 http://www.youtube.com/watch?v=Wh9a0obtQUQ

CAPÍTULO 9

LOS ARCHIVOS ESTÁTICOS

Ha llegado el momento de agregar los archivos estáticos a nuestro curso, este capítulo es uno de los cortos pero que nos permitirá hacer muchas cosas, dándonos la posibilidad de experimentar más con este estupendo framework.

CONTENIDO ESTÁTICO

Muchos de los desarrolladores que trabajan con Django se quedan sorprendidos con el funcionamiento de las vistas y plantillas, pero eso no es todo, también se deben tener en cuenta las otras partes de una aplicación: como las imágenes, las hojas de estilo, Javascript y otros elementos. Estas partes se les conoce en general como el contenido estático.

Cuando se tienen proyectos pequeños, no es mucho el trabajo al respecto, se pueden incluir este tipo de contenido en las plantillas sin ningún problema. Sin embargo cuando el proyecto deja de ser pequeño y empieza a tener muchas partes, lidiar con este tipo de contenido puede ser un dolor de cabeza.

Para evitar jaquecas innecesarias Django mediante: `django.contrib.staticfiles`, gestiona el contenido estático para las aplicaciones y los ordena en una sola ubicación fácil de referenciar y de usar.

CAMBIOS EN SETTINGS.PY

El primer lugar donde inicia el manejo de los archivos estáticos reside en el archivo de configuraciones del proyecto: `settings.py`, en este archivo tenemos líneas exclusivamente dedicadas al manejo del contenido estático.

En este archivo existen 4 elementos: `STATIC_ROOT`, `STATIC_URL`, `STATICFILES_DIRS` y `STATICFILES_FINDERS` cada uno de ellos con un propósito documentado en el mismo archivo `settings.py` a modo de comentario.

Para empezar con nuestro ejemplo del capítulo, debemos prestar atención a `STATICFILES_DIRS`, este elemento permite declarar la ruta, desde la cual se enlazará el contenido estático, lo dejamos de la siguiente manera:

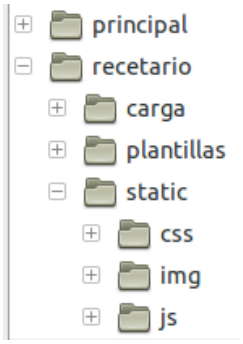
```
STATICFILES_DIRS = (
 # Put strings here, like “/home/html/static” or “C:/www/django/
static”.
 # Always use forward slashes, even on Windows.
 # Don’t forget to use absolute paths, not relative paths.
 os.path.join(RUTA_PROYECTO,’static’),
)
```

El resto de elementos referentes al contenido estático, no se manipulan. Ya que estamos en la versión de desarrollo y aún falta para la etapa de producción. No olvidar guardar el archivo, para proseguir sin errores.

DIRECTORIO STATIC

Ahora procedemos a crear el directorio: `static`, este se debe hacer dentro del directorio del proyecto, al mismo nivel que carga y plantillas. Dentro del directorio static, debemos tener una carpeta por cada tipo de contenido estático que decidamos incluir. Para el ejemplo tendremos tres subdirectorios en static: `css`, `img` y `js`.

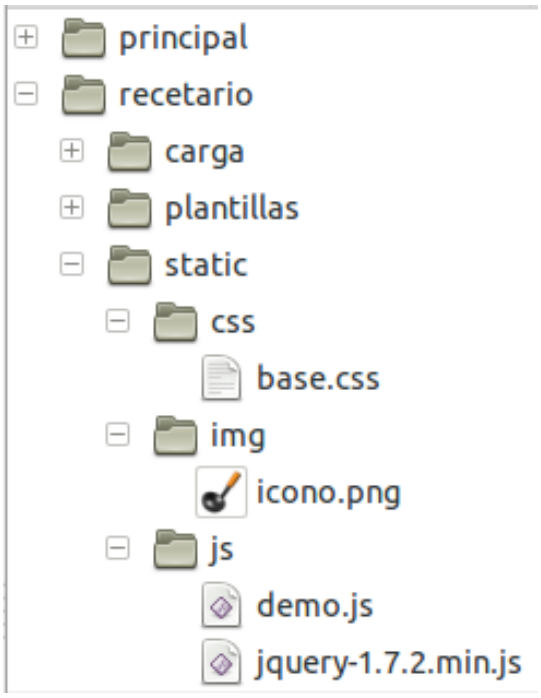
Dentro de cada uno de estos directorios, debemos incluir nuestro contenido estático. Para evitar desenfocar el curso, abarcando otro tipo de temas, usaré instrucciones muy básicas tanto para CSS, como para Javascript y de la misma forma para evitar salirnos del tema, no responderé preguntas referentes a CSS o Javascript, a excepción de alguna pregunta que afecte directamente el entendimiento del curso.



NOTA:
En esta parte se pueden usar diversas herramientas existentes como: Bootstrap, Boilerplate, 960gs, Blueprint, Jquery Mobile o cualquier otro conjunto de archivos que faciliten la construcción de interfaces de usuario. Por supuesto que si desean construir sus interfaces, sin intervención mágica de estas herramientas, también lo pueden hacer.

CONTENIDO ESTÁTICO PARA EL EJEMPLO

En este capítulo el contenido estático que usaré será el siguiente:



El contenido de `base.css`:

```
/* algunos ejemplos con CSS */
body, header, footer, div, h1, h2, ul, li, a {
    margin:0; padding:0;
}
header {
    background-color:#008500;
    color:#ffffff;
}
nav ul li a{
    text-decoration:none;
    color:#fff;
    margin:10px;
}
.activo {
  font-weight: bold;

}
.impar {
    background-color:#90EE90;
}
.par {
    background-color:#5ACC5A;
}
```

El contenido de `demo.js`:

```
/* un ejemplo con jquery */
$(function(){
    $('#menu a[href*="' + location.pathname.split("/")[1] + '"][class!="noactivo"]').addClass('activo');
});
```

Para que el `demo.js` funcione, también se debe agregar **jquery** a nuestro contenido estático.

Dentro del directorio `img` colocaré la siguiente imagen, que servirá de icono de acceso rápido o favicon (como es mayormente conocido):

USO EN LAS PLANTILLAS

Para hacer uso del contenido estático en las plantillas, cada vista debe finalizar con `context_instance=RequestContext(request)` para poder usar `{{STATIC_URL}}` y así en caso de cambiar el nombre o ubicación de la carpeta static en producción, no afecte el proyecto. (Esto hace que Django sea muy dinámico).

Las funciones dentro del archivo `views.py` quedarían así:

```
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response, get_object_or_404
from django.template import RequestContext
from django.core.mail import EmailMessage

def sobre(request):
    html = "<html><body>Proyecto de ejemplo en MDW</body></html>"
    return HttpResponse(html)

def inicio(request):
    recetas = Receta.objects.all()
    return render_to_response('inicio.html',{'recetas':recetas}, context_instance=RequestContext(request))

def usuarios(request):
    usuarios = User.objects.all()
    recetas = Receta.objects.all()
    return render_to_response('usuarios.html',{'usuarios':usuarios,'recetas':recetas}, context_instance=RequestContext(request))

def lista_recetas(request):
    recetas = Receta.objects.all()
    return render_to_response('recetas.html',{'datos':recetas}, context_instance=RequestContext(request))

def detalle_receta(request, id_receta):
    dato = get_object_or_404(Receta, pk=id_receta)
    comentarios = Comentario.objects.filter(receta=dato)
    return render_to_response('receta.html',{'receta':dato,'comentarios':comentarios}, context_instance=RequestContext(request))

def contacto(request):
    if request.method=='POST':
        formulario = ContactoForm(request.POST)
        if formulario.is_valid():
```

La plantilla `base.html` quedaría de la siguiente manera:

```
<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='utf-8'>
  <title>Recetario para MDW - {% block titulo %}{% endblock %}</title>
  <link rel='stylesheet' href='{{STATIC_URL}}css/base.css'>
  <link rel='shortcut icon' href='{{STATIC_URL}}img/icono.png'>
  {% block style_css %}{% endblock %}
  <script src='{{STATIC_URL}}js/jquery-1.7.2.min.js'></script>
  <script src='{{STATIC_URL}}js/demo.js'></script>
</head>
<body>
  <header>
  {% block encabezado %}{% endblock %}
  <nav>
    <ul id='menu'>
      <li><a href='/'>Inicio</a></li>
      <li><a href='/usuarios'>Usuarios registrados</a></li>
      <li><a href='/recetas'>Recetas</a></li>
      <li><a href="/comenta">Comenta</a></li>
      <li><a href='/sobre'>Sobre</a></li>
      <li><a href="/contacto">Contactanos</a></li>
    </ul>
  </nav>
  <hr>
  </header>
  <section class='contenido'>
  {% block contenido %}{% endblock %}
  </section>
  <footer>
  <p>Proyecto de ejemplo para el curso Django de Maestros del Web &copy;2012</p>
  </footer>
</body>
</html>
```

Notar que cada vez que se quiere usar el contenido estático se escribe `{{STATIC_URL}}`, y luego el directorio/archivo que se desee enlazar.

RESULTADO FINAL DEL EJEMPLO

Si se desea se puede observar la ejecución sin archivos estáticos, para fines de depuración, desde el terminal:

```
python manage.py runserver --nostatic
```

En cuestión de minutos ya tienen el contenido estático habitando en el proyecto.

No olviden que si se desea profundizar el tema, pues se tiene información importante en la documentación oficial de Django[1], incluyendo la referencia de la aplicación static[2].

Experimenten, jueguen con Django, prueben, practiquen, usenlo en sus proyectos pequeños que no habrá de qué arrepentirse.

1 https://docs.djangoproject.com/en/dev/howto/static-files/
2 https://docs.djangoproject.com/en/dev/ref/contrib/staticfiles/

CAPÍTULO 10

GESTIÓN DE USUARIOS

La gestión de usuarios es un proceso bastante común en todo proyecto, muchos desarrolladores han programado funcionalidades de autenticación una y otra vez a lo largo de muchos años y siempre funciona de la misma manera. Django quiere simplificarnos la vida y es por ello que viene ya con un sistema de autenticación completo que gestiona cuentas de usuario, grupos, permisos, sesiones de usuario y cookies.

El sistema de autenticación de Django[1], tiene una documentación muy completa incluyendo algunos ejemplos de uso. Abarcarlos todos puede complicar la didáctica del curso, así que voy a implementar solamente estas funcionalidades que son más frecuentes:

- Creación de usuarios
- Autenticación de usuarios
- Acceso restringido
- Cierre de sesión

El sistema de autenticación necesita de `django.contrib.auth`. por lo tanto, es necesario agregar estas líneas a nuestro `views.py`:

```
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login, authenticate, logout
from django.contrib.auth.decorators import login_required
```

Si deseas puedes unir en una sola instrucción a `UsertCreationForm` y `AuthenticationForm`, solamente separados por comas, algo así: `from django.contrib.auth.forms import UserCreationForm, AuthenticationForm`.

CREACIÓN DE USUARIOS

Para crear usuarios (sin la interfaz administrativa de Django), podemos usar el formulario que viene con Django, su nombre es `UserCreationForm` from `django.contrib.auth.forms import UserCreationForm, AuthenticationForm` que pertenece a `django.contrib.auth.forms`.

1 https://docs.djangoproject.com/en/1.4/topics/auth/

Creamos la siguiente vista (nuevo_usuario):

```
def nuevo_usuario(request):
    if request.method=='POST':
        formulario = UserCreationForm(request.POST)
        if formulario.is_valid:
            formulario.save()
            return HttpResponseRedirect('/')
    else:
        formulario = UserCreationForm()
    return render_to_response('nuevousuario.html',{'formulario':formulario}, context_instance=RequestContext(request))
```

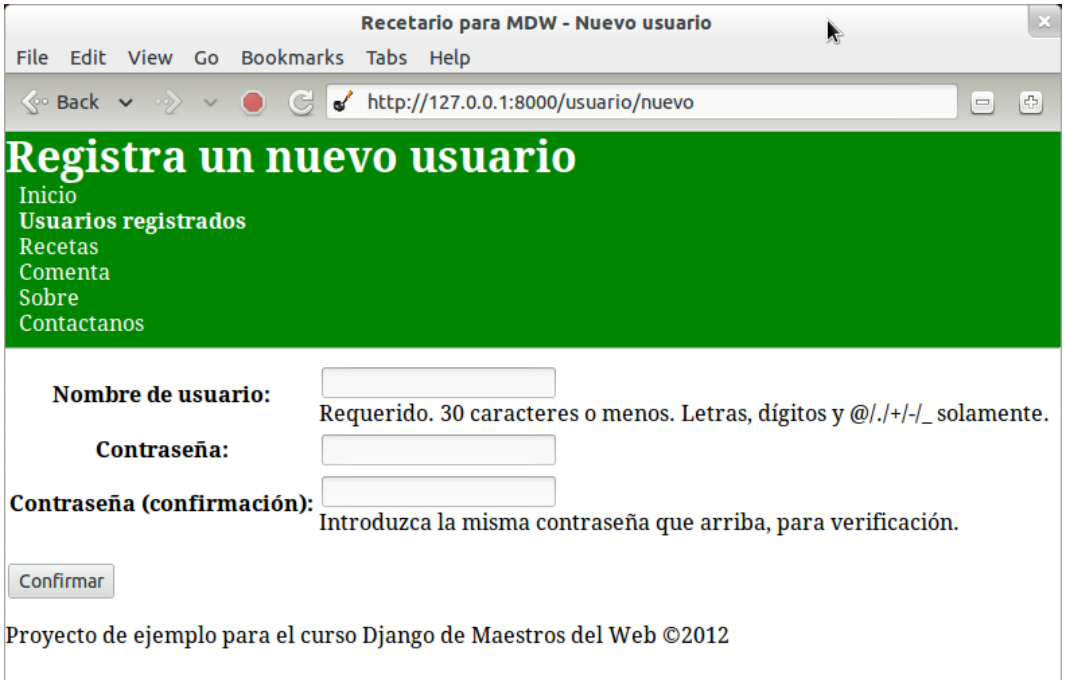
Luego, la siguiente plantilla (nuevousuario.html):

```
{% extends 'base.html' %}
{% block titulo %}Nuevo usuario{% endblock %}
{% block encabezado %}
    <h1>Registra un nuevo usuario</h1>
{% endblock %}
{% block contenido %}
    <form id='formulario' method='post' action=''>{% csrf_token %}
        <table>{{formulario}}</table>
        <p><input type='submit' value='Registrar'/></p>
    </form>
{% endblock %}
```

Y agregamos la siguiente línea a `urls.py`:

```
url(r'^usuario/nuevo$','principal.views.nuevo_usuario'),
```

Si accedemos a la URL: `http://127.0.0.1:8000/usuario/nuevo` veremos lo siguiente:



Podemos probar que crea un usuario nuevo, también puedes agregar un enlace a tus plantillas anteriores, para mejorar la navegación.

AUTENTICACIÓN DE USUARIOS

Ahora vamos a crear la interfaz para el ingreso al sistema, para ello usaremos el formulario `AuthenticationForm`, que también pertenece a `django.contrib.auth.forms` y usaremos `authenticate` y `login` de `django.contrib.auth`.

Agregamos la siguiente vista(ingresar) :

```
def ingresar(request):
    if request.method == 'POST':
        formulario = AuthenticationForm(request.POST)
        if formulario.is_valid:
            usuario = request.POST['username']
            clave = request.POST['password']
            acceso = authenticate(username=usuario, password=clave)
            if acceso is not None:
                if acceso.is_active:
                    login(request, acceso)
                    return HttpResponseRedirect('/privado')
                else:
                    return render_to_response('noactivo.html', context_instance=RequestContext(request))
            else:
                return render_to_response('nousuario.html', context_instance=RequestContext(request))
    else:
        formulario = AuthenticationForm()
    return render_to_response('ingresar.html',{'formulario':formulario}, context_instance=RequestContext(request))
```

Notar que existe una condicional que menciona “is_active”, eso nos indica que el usuario puede que exista en el sistema, pero también debe estar activo para poder ingresar.

Crearemos las plantillas (`noactivo.html`, `nousuario.html` e `ingresar.html`):

```
{% extends 'base.html' %}
{% block titulo %}Usuario desactivado{% endblock %}
{% block encabezado %}
    <h1>El usuario no esta activo</h1>
{% endblock %}
{% block contenido %}
    <p>Contactarse con el administrador, para resolver el problema.</p>
{% endblock %}
```

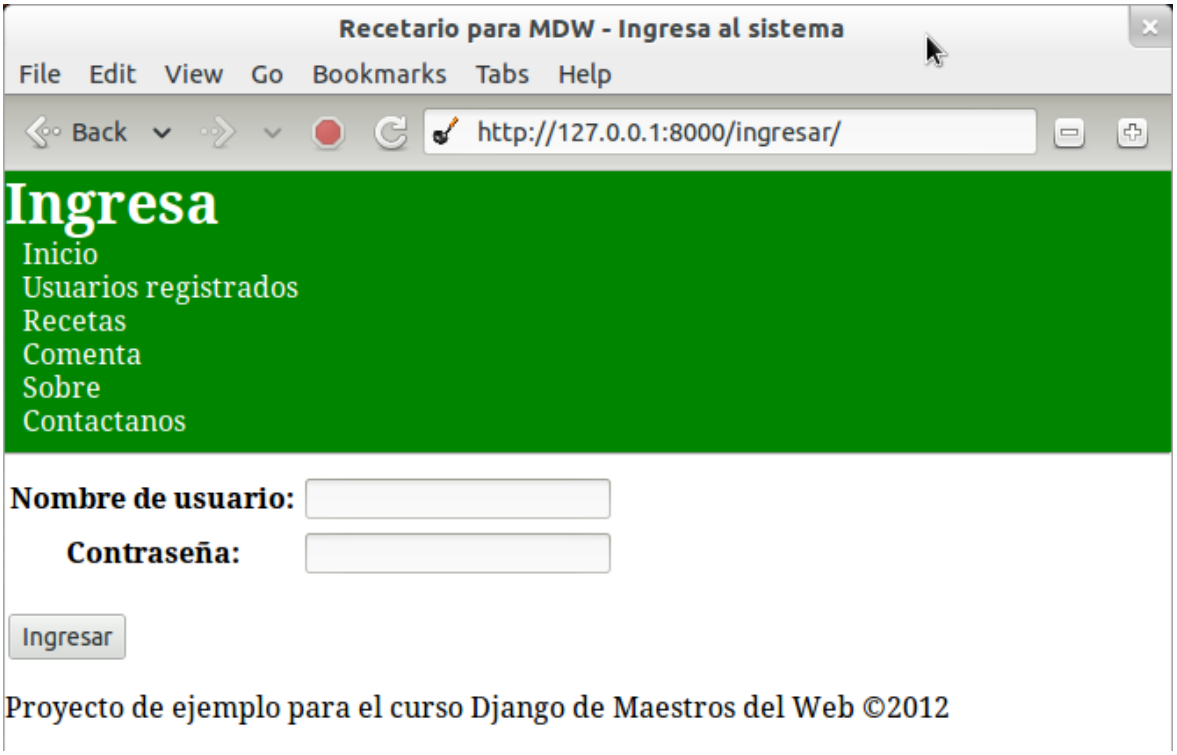
```
{% extends 'base.html' %}
{% block titulo %}Error de acceso{% endblock %}
{% block encabezado %}
    <h1>Error de acceso</h1>
{% endblock %}
{% block contenido %}
    <p>El usuario y la contraseña no coinciden o no existen.</p>
{% endblock %}
```

```
{% extends 'base.html' %}
{% block titulo %}Ingresa al sistema{% endblock %}
{% block encabezado %}
    <h1>Ingresa</h1>
{% endblock %}
{% block contenido %}
    <form id='formulario' method='post' action=''>{% csrf_token %}
        <table>{{formulario}}</table>
        <p><input type='submit' value='Ingresar'/></p>
    </form>
{% endblock %}
```

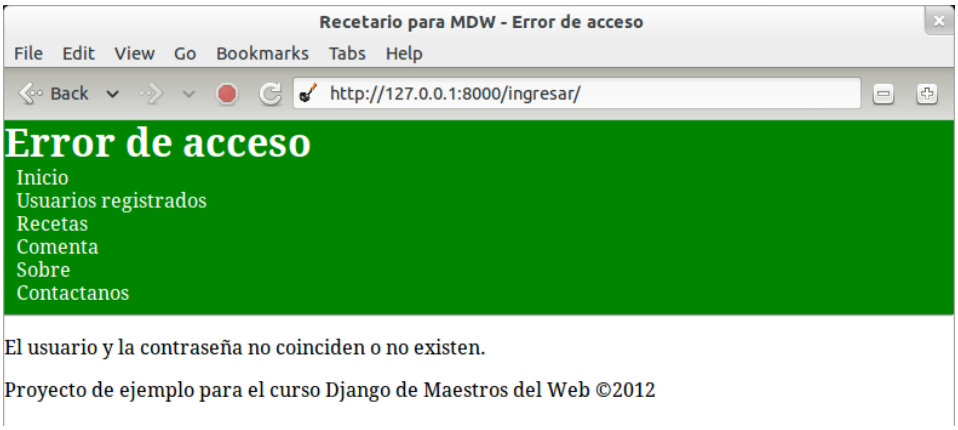
Agregamos la siguiente línea al `urls.py`:

```
url(r'^ingresar/$','principal.views.ingresar'),
```

Al probarlo ingresamos a la dirección `http://127.0.0.1:8000/ingresar/`, si tenemos un usuario inactivo (se puede activar o desactivar a un usuario desde la interfaz administrativa de Django), nos saldrá el contenido de la plantilla `noactivo.html`, si nos equivocamos de credenciales saldra nousuario.html y si le damos los datos adecuados, nos saldrá el error 404, porque aún no hemos creado la vista para (`/privado`).




Recetario para MDW - Usuario desactivado
File Edit View Go Bookmarks Tabs Help
Back
http://127.0.0.1:8000/ingresar/
El usuario no esta activo
Inicio
Usuarios registrados
Recetas
Comenta
Sobre
Contactanos
Contactarse con el administrador, para resolver el problema.
Proyecto de ejemplo para el curso Django de Maestros del Web ©2012


Recetario para MDW - Error de acceso
File Edit View Go Bookmarks Tabs Help
Back
http://127.0.0.1:8000/ingresar/
Error de acceso
Inicio
Usuarios registrados
Recetas
Comenta
Sobre
Contactanos
El usuario y la contraseña no coinciden o no existen.
Proyecto de ejemplo para el curso Django de Maestros del Web ©2012


Page not found at /privado
File Edit View Go Bookmarks Tabs Help
Back
http://127.0.0.1:8000/privado
Page not found (404)
Request Method: GET
Request URL: http://127.0.0.1:8000/privado
Using the URLconf defined in recetario.urls, Django tried these URL patterns, in this order:
1. ^$
2. ^usuarios/$
3. ^sobre/$
4. ^recetas/$
5. ^receta/(?P<id_receta>\d+)$
6. ^admin/doc/
7. ^admin/
8. ^media/(?P<path>.*)$
9. ^contacto/$
10. ^receta/nueva/$
11. ^comenta/$
12. ^usuario/nuevo$
13. ^ingresar/$
The current URL, privado, didn't match any of these.
You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard 404 page.

ACCESO RESTRINGIDO

Para complementar nuestro ejemplo anterior (`/privado`) vamos a crear una vista que permita manejar esto, esta vista tendrá la restricción de autenticación, quiere decir que se necesita ingresar al sistema para poder ver su contenido, para esto vamos a usar `login_required` que pertenece a `django.contrib.auth.decorators`, se usa mediante la sintaxis:

```
@login_required(login_url='/ingresar')
```

Esta línea se debe agregar antes de cada vista, para poder activar esta restricción, nuestra vista para privado quedaría de esta manera:

```
@login_required(login_url='/ingresar')
def privado(request):
    usuario = request.user
    return render_to_response('privado.html', {'usuario':usuario}, context_instance=RequestContext(request))
```

Notar que ahora usaremos los datos del modelo User de Django, la documentación de estos campos de usuario es de mucha ayuda[1].

Podríamos modificar nuestra vista ingresar para que no vuelva aparecer el formulario de registro, mientras el usuario ya se encuentra dentro del sistema, quedaría de la siguiente manera:

```
def ingresar(request):
    if not request.user.is_anonymous():
        return HttpResponseRedirect('/privado')
    if request.method == 'POST':
        formulario = AuthenticationForm(request.POST)
        if formulario.is_valid:
            usuario = request.POST['username']
            clave = request.POST['password']
            acceso = authenticate(username=usuario, password=clave)
            if acceso is not None:
                if acceso.is_active:
                    login(request, acceso)
                    return HttpResponseRedirect('/privado')
                else:
                    return render_to_response('noactivo.html', context_instance=RequestContext(request))
            else:
                return render_to_response('nousuario.html', context_instance=RequestContext(request))
    else:
        formulario = AuthenticationForm()
    return render_to_response('ingresar.html',{'formulario':formulario}, context_instance=RequestContext(request))
```

Fijarse que estas líneas fueron agregadas:

```
if not request.user.is_anonymous():
return HttpResponseRedirect('/privado')
```

1 https://docs.djangoproject.com/en/1.4/topics/auth/#fields

Agregaríamos al `urls.py`:

```
url(r'^privado/$','principal.views.privado'),
```

Y crearíamos la plantilla (`privado.html`):

```
{% extends 'base.html' %}
{% block titulo %}Bienvenido{% endblock %}
{% block encabezado %}
    <h1>Bienvenido</h1>
{% endblock %}
{% block contenido %}
    <p>Bievenido {{usuario.username|upper}},
        la última vez que te vimos por aquí fue el {{usuario.last_login}}.</p>
    <p>Eres miembro desde el {{usuario.date_joined}}</p>
{% endblock %}
```

Nuestro resultado sería:

Esto nos obligará a tener una manera de cerrar la sesión.

CIERRE DE SESIÓN

Para el cierre de sesión necesitamos logout que se encuentra en `django.contrib.auth`, la vista sería tan simple como esto:

```
@login_required(login_url='/ingresar')
def cerrar(request):
    logout(request)
    return HttpResponseRedirect('/')
```

Y en `urls.py` está línea:

```
url(r'^cerrar/$', 'principal.views.cerrar'),
```

Solo basta con ingresar a `http://127.0.0.1:8000/cerrar/` y se cerrará la sesión.

El archivo `urls.py` quedaría, al final de todas las modificaciones, de la siguiente manera:

```
1  from django.conf.urls import patterns, include, url
2  from django.contrib import admin
3  from django.conf import settings
4
5  admin.autodiscover()
6
7  urlpatterns = patterns('',
8      url(r'^$','principal.views.inicio'),
9      url(r'^usuarios/$','principal.views.usuarios'),
10     url(r'^sobre/$','principal.views.sobre'),
11     url(r'^recetas/$','principal.views.lista_recetas'),
12     url(r'^receta/(?P<id_receta>\d+)$','principal.views.detalle_receta'),
13     url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
14     url(r'^admin/', include(admin.site.urls)),
15     url(r'^media/(?P<path>.*)$','django.views.static.serve',
16         {'document_root':settings.MEDIA_ROOT,}
17     ),
18     url(r'^contacto/$','principal.views.contacto'),
19     url(r'^receta/nueva/$','principal.views.nueva_receta'),
20     url(r'^comenta/$','principal.views.nuevo_comentario'),
21     url(r'^usuario/nuevo$','principal.views.nuevo_usuario'),
22     url(r'^ingresar/$','principal.views.ingresar'),
23     url(r'^privado/$','principal.views.privado'),
24     url(r'^cerrar/$', 'principal.views.cerrar'),
25 )
```

REPOSITORIO DEL PROYECTO E INDICACIONES FINALES

Si desean revisar el código completo del proyecto pueden hacerlo desde el repositorio en github[1], para comparar y resolver cualquier duda, eviten solo copiar y pegar, mejor escriban el código, será más educativo.

1 http://neosergio.github.com/recetario_mdw/

CAPÍTULO 11

DESPLIEGUE DEL SERVIDOR WEB

Existen muchas formas de llevar un proyecto de Django a producción y justamente es uno de los motivos que generan mayor confusión. La documentación del proyecto para el despliegue de Django[1] nos sugiere tres formas:

- Usando WSGI[2]
- Usando FastCGI, SCGI o AJP[3]
- mod_python (obsoleto)[4]

De estas tres opciones, he visto por conveniente usar la primera de ellas (WSGI), me parece más simple de replicar sin entrar en mucho detalle. Sin embargo no sólo es eso, también existen otras tres opciones para trabajar con WSGI, estas son:

- Apache y mod_wsgi[5]
- uWSGI[6]
- Gunicorn[7]

Y de la misma manera elegí la primera, Apache es un servidor que muchos han instalado previamente y que de seguro, ya saben hacerlo y configurarlo independientemente del sistema operativo que usen, si no lo sabes hacer aún, hay mucha información en Internet de como hacerlo, te comparto un vídeo que te puede servir: Installing LAMP On Ubuntu[8].

Es necesario que tengas por lo menos conocimientos básicos de como levantar un servidor Apache, un poco de virtualhost y como configurarlo para que puedas abrir otros puertos, si no sabes estos temas puedes saltarte a la parte de proveedores o tomar una pausa en el curso para aprender un poco más de lo que te he mencionado y entiendas mejor lo que sigue.

1 https://docs.djangoproject.com/en/1.4/howto/deployment/
2 https://docs.djangoproject.com/en/1.4/howto/deployment/wsgi/
3 https://docs.djangoproject.com/en/1.4/howto/deployment/fastcgi/
4 https://docs.djangoproject.com/en/1.4/howto/deployment/modpython/
5 https://docs.djangoproject.com/en/1.4/howto/deployment/wsgi/modwsgi/
6 https://docs.djangoproject.com/en/1.4/howto/deployment/wsgi/uwsgi/
7 https://docs.djangoproject.com/en/1.4/howto/deployment/wsgi/gunicorn/
8 http://www.youtube.com/watch?v=BnCt8-YaPJo

CONFIGURACIÓN EN EL SERVIDOR

Una vez con Apache corriendo, para nuestro ejemplo vamos a usar el puerto 90, para ello debemos asegurarnos tener, en nuestro `httpd.conf` la línea:

```
Listen 90
```

Luego es necesario tener instalado y activado `mod_wsgi` para que trabaje con Apache, de la misma manera, si no sabes hacerlo, es sencillo buscar más información sobre esto por Internet, no te tomará mucho tiempo.

Adicionalmente también debes tener la siguiente línea en nuestro `httpd.conf`:

```
WSGIPythonPath /ruta_del_directorio_del_proyecto/recetario/
```

Es la ruta **absoluta** de tu directorio del proyecto (ese directorio que tiene el `manage.py`), si usas Windows no olvides que la ruta debe ir entre comillas para evitar inconvenientes, algo como:

```
WSGIPythonPath “letra://ruta_del_directorio_del_proyecto/recetario/”
```

Una vez con estas dos líneas configuradas, es momento de crear nuestro virtualhost, debe quedar de la siguiente manera:

```
<VirtualHost *:90>

    WSGIScriptAlias / /ruta_del_directorio_del_proyecto/recetario/recetario/wsgi.py

    Alias /static/ /ruta_del_directorio_del_proyecto/recetario/recetario/static/
    Alias /media/ /ruta_del_directorio_del_proyecto/recetario/recetario/carga/

    <Directory /ruta_del_directorio_del_proyecto/recetario>
    <Files wsgi.py>
    Order deny,allow
    Allow from all
    </Files>
    </Directory>

    <Directory /ruta_del_directorio_del_proyecto/recetario/recetario/static/>
    Order deny,allow
    Allow from all
    </Directory>

    <Directory /ruta_del_directorio_del_proyecto/recetario/recetario/carga/>
    Order deny,allow
    Allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Si se dan cuenta hay dos alias uno para static y el otro para media, aquí una aclaración, se recomienda utilizar un servidor web diferente para servir los archivos estáticos (static y media), es por ello que para eso Django puede funcionar usualmente con:

- Nginx[1]
- lighttpd[2]
- TUX web server[3]
- Cherokee[4]

Pero, para no hacer complejo este capítulo, vamos a usar el mismo Apache para este propósito, por eso declaré dos alias y dos directorios más en el virtualhost, como se puede ver en la imagen anterior.

Ahora es momento de cambiar las configuraciones de nuestro proyecto, necesitamos poner rutas absolutas, para que todo se enlace correctamente. Abrimos nuestro archivo settings.py y modificamos las siguientes líneas:

```
DEBUG = False
```

En el nombre de la base de datos, se debe incluir la ruta absoluta completa, así (en Windows debe ir entre comillas, no lo olvides):

```
'NAME': '/ruta_del_directorio_del_proyecto/recetario/recetario.db',
```

El media url debe ir así:

```
MEDIA_URL = 'http://localhost:90/media/'
```

El static url debe ser así:

```
STATIC_URL = 'http://localhost:90/static/'
```

Como el modo de depuración esta desactivado (`DEBUG = False`), necesitamos dos plantillas más, una que maneje el error 404 y otra el error 500, sus nombres deben ser: `404.html` y `500.html`. Las mías son estas y lucen así al ejecutarse:

```
1  {% extends 'base.html' %}
2
3  {% comment %} Aqui van comentarios {% endcomment %}
4
5  {% block titulo %} 404 {% endblock %}
```

1 http://wiki.nginx.org/Main
2 http://www.lighttpd.net/
3 http://en.wikipedia.org/wiki/TUX_web_server
4 http://www.cherokee-project.com/

```
6
7   {% block encabezado %}
8       <h1>Recetario :( </h1>
9   {% endblock %}
10
11  {% block contenido %}
12    <h1>Lo sentimos no encontramos lo que estas buscando :( </h1>
13    <p>
14      <img src='{{STATIC_URL}}img/kernel.jpg'>
15    </p>
16  {% endblock %}
```

```
{% extends 'base.html' %}

{% comment %} Aqui van comentarios {% endcomment %}

{% block titulo %} 500 {% endblock %}
{% block style_css %}
    <link href='http://127.0.0.1:90/static/css/base.css' rel='stylesheet'>
    <link rel='shortcut icon' href='http://127.0.0.1:90/img/icono.png'>
{% endblock %}
{% block encabezado %}
    <h1>Recetario :( </h1>
{% endblock %}

{% block contenido %}
  <h1>Lo sentimos nuestro servidor se quedo dormido, hacemos todos los esfuerzos
  por despertarlo. Un poquito de paciencia. </h1>
  <p>
    <img src='http://127.0.0.1:90/static/img/leono.jpg'>
  </p>
{% endblock %}
```

La 404 sale cuando no encuentra la ruta y el 500 cuando hay un error interno (de sintaxis o de mal funcionamiento de un componente).

Si deseamos que nuestro admin, también se vea de la misma forma que en desarrollo, debemos agregar la carpeta de archivos estáticos del admin a nuestra carpeta 'static', para hacer esto debemos tener a la mano los archivos con los cuales instalamos Django (si no los tienes descargalo de la página del proyecto[1]) y debemos ir a la ruta (para encontrarlos):

```
/carpeta_de_descargas/Django-1.4/django/contrib/admin/static
```

Copiar toda la carpeta '`admin/`' a nuestro directorio '`static/`' y eso es todo.

> **NOTA:**
> No se debe olvidar agregar los permisos necesarios a los directorios, sobre todo si no estas usando Windows para seguir el curso.

Ahora ya podemos ver el resultado ejecutándose en el servidor (sin el `python manage.py runserver` y de frente en `http://127.0.0.1:90/`), misión cumplida, nuestro proyecto está en un servidor de producción, hay muchas configuraciones que hacer, el trabajo no queda ahí, hay que optimizar constantemente y estar pendiente de lo que sucede para mantener nuestra aplicación ejecutándose sin problemas.

1 https://www.djangoproject.com/download/1.4/tarball/

Jacob Kaplan-Moss (@jacobian[1]), es uno de los creadores de Django y dió una charla sobre Django en el mundo real (en producción), sería estupendo que revisarán las diapositivas de su charla: Django in the Real World Presentation[2], para que entiendan a que me refiero.

PROVEEDORES PARA DJANGO

Para publicar un proyecto en Django, va a depender mucho del alcance del mismo, como hemos leído hace un momento, se puede publicar desde un servidor local dentro de una organización, pero no es lo único se puede usar un servicio en la nube para aplicaciones de muchas visitas diarias. En pocas palabras: hay para todos los gustos y precios. Por ahora lo importante es conocer las opciones generales para publicar proyectos en Internet con Django:

- Always Data[3]: Ofrece diversos planes desde los 10 MB de almacenamiento (gratuito), hasta los 50 GB. Lo resaltante es que configurar una cuenta para trabajar con Django es muy fácil y rápido. Además Always Data ha sido desarrollada completamente usando Django.
- Webfaction[4]: Ofrece construcción de un proyecto rápidamente a través de su panel de configuración web. Se pueden instalar otras bibliotecas de Python de ser necesario. Sus planes de servicio tienen buena relación costo/beneficio.
- DreamHost[5]: Para los que ya están acostumbrados a DreamHost, es la opción ideal. Sin embargo, se debe tener en cuenta que su uso sólo se recomienda para proyectos pequeños, donde las visitas no sean altas. Actualmente se puede usar el cupón DJANGOHOSTING para ahorrar 49 dólares, en planes de uno o más años.
- Linode[6]: Para aquellos que buscan un VPS como solución de despliegue de sus proyectos, está opción es muy buena, ya que brinda soporte a los usuarios[7] para desplegar sus proyectos eficientemente.
- Gondor[8]: Es un servicio en la nube, construido especialmente para proyectos con Django. El beneficio de este servicio es que permite escalar, es ideal para Startups.
- Heroku[9]: Ideal para trabajar con proyectos que usan GIT como control de versiones, excelente seguimiento de incidencias, mediante reportes especializados, gratuitamente se pueden usar 5MB de almacenamiento.

1 https://twitter.com/#%21/jacobian
2 http://www.maestrosdelweb.com/images/2012/06/Django-in-the-Real-World-Presentation.pdf
3 https://www.alwaysdata.com/
4 http://www.webfaction.com/
5 http://dreamhost.com/
6 http://www.linode.com/
7 http://library.linode.com/frameworks/django-apache-mod-wsgi
8 https://gondor.io/
9 http://www.heroku.com/

- Google App Engine[1]: No necesita presentaciones, es un servicio estupendo para publicar tus proyectos con el soporte de Google.

Existen otras opciones para publicar un proyecto con Django, afortunadamente existe una lista de recursos, que son evaluados de manera comunitaria y que exclusivamente se enfocan en alojamiento de sitios desarrollados con Django. Este sitio se llama DjangoFriendly[2]. También se puede dar un vistazo al wiki[3], del propio proyecto, dedicado a recursos de alojamiento.

AYUDA FINAL

Encontrar ayuda en Django por Internet se hace cada vez más fácil, se tiene la documentación oficial[4], con un a gran comunidad detrás de Django[5], donde también se pueden encontrar recursos interesantes. Incluyendo las comunidades en español reunidas en Django.es[6] y en el grupo de google de Django en Español[7].

Por supuesto debo mencionar además a forosdelweb.com donde podemos encontrar sugerencias, recomendaciones y respuestas a nuestras preguntas con respecto a desarrollo web, Django y donde alojar nuestros trabajos por ejemplo[8], entre otros temas interesantes.

Les dejo las chuletas (cheat sheets) del curso, recopiladas en un sólo archivo: Django 1.4 CheatSheet – Maestros del Web[9]. Les recomiendo también usar Twitter (si es que aún no lo hacen), es una gran red de información para estar al tanto de lo que sucede en el mundo Django.

El repositorio del proyecto de ejemplo[10], está quedando con la parte de producción comentada y explicada, para que al momento de que lo descarguen aún puedan ejecutarlo en modo de desarrollo y cuando deseen pasarlo a producción puedan descomentar las líneas de producción y comentar las de desarrollo. También he incluido un directorio nuevo llamado: apache_demo con algunos archivos de demostración y explicación (descubrir cómo usarlos será sencillo).

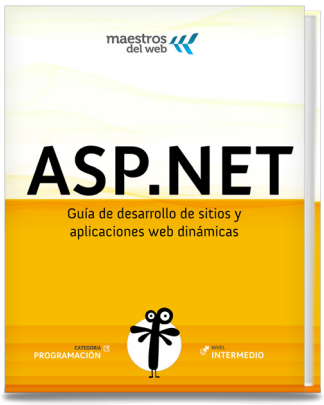
1 http://code.google.com/intl/es-AR/appengine/
2 http://djangofriendly.com/hosts/
3 https://code.djangoproject.com/wiki/DjangoFriendlyWebHosts
4 https://docs.djangoproject.com/en/1.4/
5 https://www.djangoproject.com/community/
6 http://django.es/
7 https://groups.google.com/forum/?fromgroups#%21forum/django-es
8 http://www.forosdelweb.com/f19/paas-platforma-service-experiencias-915469/
9 http://www.maestrosdelweb.com/images/2012/06/django-1.4-cheatsheet-maestros-del-web.pdf
10 http://neosergio.github.com/recetario_mdw/

MÁS GUÍAS DE MAESTROS DEL WEB

GUÍA PYTHON

Python es uno de los lenguajes de programación multiparadigma, más potente y que menor curva de aprendizaje demanda.

Visita la Guía Python
http://bit.ly/QgE4Kk



GUÍA ASP.NET

ASP.NET es un modelo de desarrollo Web unificado creado por Microsoft para el desarrollo de sitios y aplicaciones web dinámicas con un mínimo de código.

Visita la Guía ASP.NET
http://mdw.li/guiaaspnet

GUÍA ZEND

Zend Framework es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5.

Visita la Guía Zend
http://mdw.li/guiazend

maestros del web

CURSO ANDROID

Actualiza tus conocimientos con el curso sobre Android para el desarrollo de aplicaciones móviles.

[Visita el Curso Android](http://mdw.li/lmlydX)
http://mdw.li/lmlydX

GUÍA IOS

Desarrollo móvil enfocado a iphone e iPad, ahora actualizada a la versión iOS5.

[Visita la Guía IOS](http://j.mp/iWFI9v)
http://j.mp/iWFI9v

GUÍA COMMUNITY MANAGER

Te brindará un panorama sobre las características y estrategias que debes conocer para desarrollarte en esta profesión.

[Visita la Guía Community Manager](http://j.mp/ACDQ1P)
http://j.mp/ACDQ1P

maestros del web

ADICTOS A LA COMUNICACIÓN

Utiliza las herramientas sociales en Internet para crear proyectos de comunicación independientes.

[Visita Adictos a la comunicación](http://mdw.li/guiacomunica)
http://mdw.li/guiacomunica

GUÍA STARTUP

Aprende las oportunidades, retos y estrategias que toda persona debe conocer al momento de emprender.

[Visita la Guía Startup](http://mdw.li/gkTDom)
http://mdw.li/gkTDom

LOS MAESTROS DEL WEB

Una serie de perfiles de personas y proyectos que nos inspiran a permanecer en el medio, aprender y seguir evolucionando.

[Visita Los Maestros del Web](http://j.mp/spAncK)
http://j.mp/spAncK

maestros del web