APÉNDICE SOBRE DJANGO.

Furlán, Ricardo Javier

Martinez, Gustavo Ariel

2014

Índice de contenidos

Introducción a Django

Django, es un framework de desarrollo Web que ahorra tiempo y hace que el desarrollo Web sea divertido. Utilizando Django puedes crear y mantener aplicaciones Web de alta calidad con un mínimo esfuerzo.

En el mejor de los casos, el desarrollo web es un acto entretenido y creativo; en el peor, puede ser una molestia repetitiva y frustrante. Django te permite enfocarte en la parte divertida al mismo tiempo que mitiga el esfuerzo de las partes repetitivas. De esta forma, provee un alto nivel de abstracción de patrones comunes en el desarrollo Web, atajos para tareas frecuentes de programación y convenciones claras sobre cómo solucionar problemas.

Al mismo tiempo, Django intenta no entrometerse, dejándote trabajar fuera del ámbito del framework según sea necesario. El objetivo es incursionar en Django. El enfoque es doble. Primero, explicando en profundidad lo que hace Django, y cómo crear aplicaciones Web con él. Segundo, discutiendo conceptos de alto nivel cuando se considere apropiado.

Al conocer Django, se aprenderá las habilidades necesarias para desarrollar sitios Web poderosos de forma rápida, con código limpio y de fácil mantenimiento.

Historia de Django

Django nació naturalmente de aplicaciones de la vida real escritas por un equipo de desarrolladores Web en Lawrence, Kansas. Nació en el año 2003, cuando los programadores Web del diario Lawrence Journal- World, Adrian Holovaty y Simon Willison, comenzaron a usar Python para crear sus aplicaciones. El equipo de The World Online, responsable de la producción y mantenimiento de varios sitios locales de noticias, prosperaban en un entorno de desarrollo dictado por las fechas límite del periodismo. Para los sitios los periodistas (y los directivos) exigían que se agregaran nuevas características y que aplicaciones enteras se crearan a una velocidad vertiginosa, a menudo con sólo días u horas de preaviso. Es así que Adrian y Simon desarrollaron por necesidad un framework de desarrollo Web que les ahorrara tiempo (era la única forma en que podían crear aplicaciones mantenibles en tan poco tiempo).

En el año 2005, luego de haber desarrollado este framework hasta el punto en que estaba haciendo funcionar la mayoría de los sitios World Online, el equipo de World Online, que ahora incluía a Jacob Kaplan-Moss, decidió liberar el framework como software de código abierto. Lo liberaron en julio de 2005 y lo llamaron Django.

Esta historia es relevante porque ayuda a explicar dos cuestiones clave. La primera es el “punto dulce” de Django. Debido a que Django nació en un entorno de noticias, ofrece varias características (en particular la interfaz 'admin') que son particularmente apropiadas para sitios de “contenido” (sitios como eBay, craigslist.org y washingtonpost.com) que ofrecen información basada en bases de datos .

La segunda cuestión a resaltar es cómo los orígenes de Django le han dado forma a la cultura de su comunidad de código abierto. Debido a que Django fue extraído de código de la vida real, en lugar de ser un ejercicio académico o un producto comercial, está especialmente enfocado en resolver problemas de desarrollo Web con los que los desarrolladores de Django se han encontrado. Como resultado de eso, Django es activamente mejorado casi diariamente. Los desarrolladores del framework tienen un alto grado de interés en asegurarse de que Django les ahorre tiempo a los desarrolladores, produzca aplicaciones que son fáciles de mantener y rindan bajo mucha carga. Aunque existan otras razones, los desarrolladores están motivados por sus propios deseos egoístas de ahorrarse tiempo a ellos mismos y disfrutar de sus trabajos.

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

¿Que es un framework?

Django es un miembro importante de una nueva generación de frameworks Web.

¿Y qué significa ese término exactamente?
Para contestar esa pregunta, consideremos el diseño de una aplicación Web escrita usando el estándar Common Gateway Interface (CGI), una forma popular de escribir aplicaciones Web alrededor del año 1998. En esa época, cuando escribías una aplicación CGI, hacías todo por ti mismo -- el equivalente a hacer una torta desde cero.

Por ejemplo,aquí hay un script CGI sencillo, escrito en Python, que muestra los diez libros más recientemente publicados de una base de datos:

```
#!/usr/bin/python

import MySQLdb

print "Content-Type: text/html"
print
print "<html><head><title>Libros</title></head>"
print
print "<body>"
print "<h1>Los ultimos 10 libros</h1>"
print "<ul>"
conexion = MySQLdb.connect(user='yo', passwd='dejame_entrar', db='mi_base')
cursor = conexion.cursor()
```

```
cursor.execute("SELECT nombre FROM libros ORDER BY fecha_pub DESC LIMIT 10")

for fila in cursor.fetchall():
print "<li> %s</li>" % fila[0]
print "</ul>"
print "</body></html>"
conexion.close()
```

Este código es fácil de entender. Primero imprime una línea de “Content-Type”, seguido de una línea en blanco, tal como requiere CGI. Imprime HTML introductorio, se conecta a la base de datos y ejecuta una consulta que obtiene los diez libros más recientes. Hace un bucle sobre esos libros y genera una lista HTML desordenada. Finalmente imprime el código para cerrar el HTML y cierra la conexión con la base de datos.

Con una única página dinámica como esta, el enfoque desde cero no es necesariamente malo. Por un lado, este código es sencillo de comprender (incluso un desarrollador novato puede leer estas 16 líneas de Python y entender todo lo que hace). No hay más nada que aprender; no hay más código para leer. También es sencillo de utilizar: tan sólo guarda este código en un archivo llamado ultimoslibros.cgi, sube ese archivo a un servidor Web y visita esa página con un navegador.

Pero a medida que una aplicación Web crece más allá de lo trivial, este enfoque se desmorona y te enfrentas a una serie de problemas:

¿Qué sucede cuando múltiples páginas necesitan conectarse a la base de datos?

Seguro que ese código de conexión a la base de datos no debería estar duplicado en cada uno de los scripts CGI, así que la forma pragmática de hacerlo sería refactorizarlo en una función compartida.

¿Debería un desarrollador realmente tener que preocuparse por imprimir la línea de “Content-Type” y acordarse de cerrar la conexión con la base de datos?

Este tipo de código repetitivo reduce la productividad del programador e introduce la

oportunidad para que se cometan errores. Estas tareas de configuración y cierre estarían mejor manejadas por una infraestructura común.

¿Qué sucede cuando este código es reutilizado en múltiples entornos, cada uno con una base de datos y contraseñas diferentes?

En ese punto, se vuelve esencial alguna configuración específica del entorno.

¿Qué sucede cuando un diseñador Web que no tiene experiencia programando en Python desea rediseñar la página?

Lo ideal sería que la lógica de la página (la búsqueda de libros en la base de datos) esté separada del código HTML de la página, de modo que el diseñador pueda hacer modificaciones sin afectar la búsqueda.

En si, un framework web es un software que alivia el sufrimiento derivado de construir páginas web dinámicas. Abstrae problemas comunes al desarrollo web y proporciona atajos para tareas de programación frecuentes.

Para los lectores que desconocen el desarrollo de sitios web dinámicos: un sitio web dinámico es uno en el que las páginas no son simplemente documentos HTML colocados en algún lugar del sistema de ficheros de un servidor.

En cambio, en un sitio web dinámico, cada página la genera un programa de computador (una famosa "aplicación web") que usted, el desarrollador web, crea. Por ejemplo, una aplicación web podría obtener registros de una base de datos o realizar alguna acción basándose en la entrada del usuario.

Un buen framework web resuelve los siguientes problemas:

- Proporciona un método para hacer corresponder peticiones URL con el código que maneja las peticiones. En otras palabras, le otorga una manera de designar qué código se ejecutará para cada URL. Por ejemplo, se le podría decir al framework:

"Para las URLs que se parezcan a /users/joe/, ejecuta el código que muestre el perfil del usuario con ese nombre de usuario".

•Facilita mostrar, validar y volver a mostrar formularios HTML. Los formularios HTML son la principal manera de obtener datos de entrada de los usuarios web, así que más le vale a un framework web facilitar la representación de formularios y el manejo del código tedioso para mostrar y volver a mostrar formularios (resaltando los errores).

• Convierte la entrada que envía el usuario en estructuras de datos que se pueden manipular cómodamente. Por ejemplo, el framework podría convertir los datos de un formulario HTML en tipos de datos nativos al lenguaje de programación que se esté utilizando.

• Ayuda a separar el contenido de la presentación mediante un sistema de plantillas, de manera que se pueda cambiar el aspecto de un sitio web sin afectar al contenido, y viceversa.

• Se integra cómodamente con las capas de almacenamiento (como las bases de datos) pero no exige estrictamente el uso de una base de datos.

• Le permite trabajar más productivamente, a un nivel de abstracción mayor que si estuviera programando usando, digamos, HTTP. Pero no le prohíbe ir un nivel de abstracción "hacia abajo" cuando sea necesario.

• Se aparta de su camino, evitando llenarle la aplicación de manchas sucias, como URLs que contengan ".aspx" o ".php".

Django hace todas estas cosas correctamente; y presenta una serie de características que elevan el listón de lo que debería ser un framework web.

El framework está escrito en Python, un lenguaje de programación bonito, conciso,

potente y de alto nivel. Para desarrollar un sitio utilizando Django hay que escribir código Python que utiliza las bibliotecas de Django.

El modelo MVC

El modelo–vista–controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

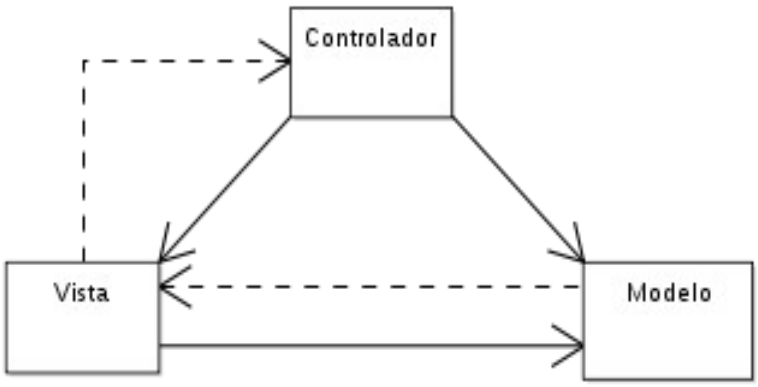

Figura 1.1

El patrón MVC fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones.

El MVC fue introducido por Trygve Reenskaug en Smalltalk-76 en los años 70 y, seguidamente, en los años 80, Jim Althoff y otros implementaron una versión de MVC para la biblioteca de clases de Smalltalk-80.Sólo más tarde, en 1988, MVC se expresó

como un concepto general en un artículo sobre Smalltalk-80.

En esta primera definición de MVC el controlador se definía como "*el módulo que se ocupa de la entrada*" (de forma similar a como la vista "*se ocupa de la salida*"). Esta

definición no tiene cabida en las aplicaciones modernas en las que esta funcionalidad es asumida por una combinación de la 'vista' y algún framework moderno para desarrollo. El 'controlador', en las aplicaciones modernas de la década de 2000, es un módulo o una sección intermedia de código, que hace de intermediario de la comunicación entre el 'modelo' y la 'vista', y unifica la validación (utilizando llamadas directas o el "*observer*" para desacoplar el 'modelo' de la 'vista' en el 'modelo').

De manera genérica, los componentes de MVC se podrían definir como sigue:

• **El Modelo:** Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

• **El Controlador:** Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.

• **La Vista:** Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

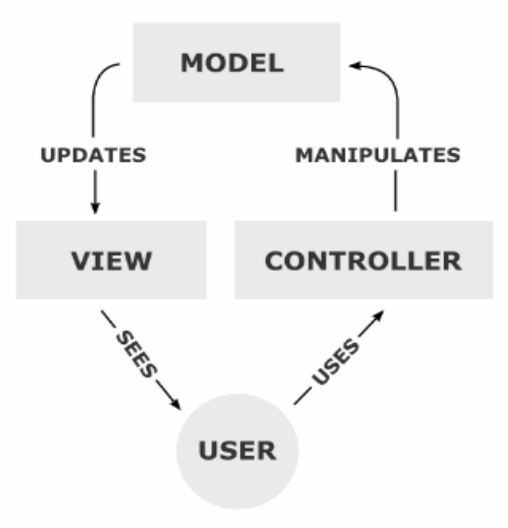

Figura 1.2

MVC y bases de datos

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos que debe utilizar la aplicación; en líneas generales del MVC dicha gestión corresponde al modelo. La unión entre capa de presentación y capa de negocio conocido en el paradigma de la Programación por capas representaría la integración entre la Vista y su correspondiente Controlador de eventos y acceso a datos, MVC no pretende discriminar entre capa de negocio y capa de presentación pero si pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la Vista y el Controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

Django y el “Patrón MTV”

Cabe destacar que Django fue diseñado para promover el acoplamiento débil y la estricta separación entre las piezas de una aplicación. Siguiendo esta filosofía es fácil hacer cambios en un lugar particular de la aplicación sin afectar otras piezas.

Existen tres piezas (la lógica de acceso a la base de datos, la lógica de negocios, y la lógica de presentación) las cuales, unidas, comprenden un concepto que a veces es llamado el patrón de arquitectura de software Modelo-Vista-Controlador (MVC). En este patrón, el “Modelo” hace referencia al acceso a la capa de datos, la “Vista” se refiere a la parte del sistema que selecciona qué mostrar y cómo mostrarlo, y el “Controlador” implica la parte del sistema que decide qué vista usar, dependiendo de la entrada del usuario, accediendo al modelo si es necesario.

Django sigue el patrón MVC tan al pie de la letra que puede ser llamado un framework MVC. Someramente, la M, V y C se separan en Django de la siguiente manera:

• M, la porción de acceso a la base de datos, es manejada por la capa de la base de datos de Django.

• V, la porción que selecciona qué datos mostrar y cómo mostrarlos, es manejada por la vista y las plantillas.

• C, la porción que delega a la vista dependiendo de la entrada del usuario, es manejada por el framework mismo siguiendo tu URLconf y llamando a la función apropiada de Python para la URL obtenida.

Debido a que la “C” es manejada por el mismo framework y la parte más emocionante se produce en los modelos, las plantillas y las vistas, Django es conocido como un framework MTV. En el patrón de diseño MTV,

• M significa “Model” (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.

• T significa “Template” (Plantilla), la capa de presentación. Esta capa contiene las decisiones relacionadas a la presentación: es básicamente una página HTML con algunas etiquetas extras propias de Django, en si no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc).

• V significa “View” (Vista): La vista se presenta en forma de funciones en Python, su propósito es determinar que 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 externos y la validación de datos a través de formularios. Lo mas 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.

Además, Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, esta 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 y sigue la filosofia de Python: “Explicito es mejor que implícito”. Este URLConf permite que las rutas que maneje Django sean agradables y entendibles para el usuario.

Figura 1.3

Django y Python

En esencia, Django es sencillamente una colección de bibliotecas escritas en el lenguaje de programación Python.

Para desarrollar un sitio usando Django se escribe código Python que utiliza esas bibliotecas. Aprender Django, entonces, es sólo cuestión de aprender a programar en Python y comprender cómo funcionan las bibliotecas Django.

Si se tiene experiencia programando en Python, no debería haber mayores problema en meterse de lleno. En conjunto, el código Django no produce “magia negra” (es decir, trucos de programación cuya implementación es difícil de explicar o entender). Para el desarrollador, aprender Django será sólo cuestión de aprender las convenciones y APIs de Django.

Si no se tiene experiencia programando en Python, le espera una grata sorpresa. Es fácil de aprender y muy divertido de usar.

Bibliografía

- “El libro de Django”, Adrian Holovaty y Jacob Kaplan-Moss, revisión 757, 2008.
- http://www.maestrosdelweb.com/editorial/python-django-mejorandola/
- http://www.maestrosdelweb.com/editorial/curso-django-entendiendo-como-trabaja-django/
- http://es.wikipedia.org/wiki/Django_(framework)
- http://www.maestrosdelweb.com/editorial/curso-django-introduccion/