

---

# **Sistemas Web Desconectados**

***Release 1***

**van Haaster, Diego Marcos; Defossé, Nahuel**

August 10, 2009



---

# Índice general

---

<b>1. Tecnologías del servidor</b>	<b>3</b>
1.1. CGI . . . . .	3
1.2. WSGI . . . . .	3
1.3. Lenguajes interpretados . . . . .	3
1.4. Frameworks web . . . . .	3
1.5. Django . . . . .	3
1.6. El mapeador Objeto-Relacional de Django . . . . .	7
<b>2. Glosario</b>	<b>9</b>
<b>3. Indices, glosario y tablas</b>	<b>11</b>
<b>Índice</b>	<b>13</b>



Índice:



---

# Tecnologías del servidor

---

## 1.1 CGI

CGI es bla

## 1.2 WSGI

WSGI es CGI para Python.

## 1.3 Lenguajes interpretados

Python es un lenguaje interpretado.

## 1.4 Frameworks web

Un framework web es una cosa loca.

## 1.5 Django

Django es un framework web escrito en Python el cual sigue vagamente el concepto de Modelo Vista Controlador. Ideado inicialmente como un administrador de contenido para varios sitios de noticias, los desarrolladores encontraron que su CMS era lo suficientemente genérico como para cubrir un ámbito más amplio de aplicaciones.

En honor al músico Django Reinhardt, fue liberado el código base bajo la licencia *BSD* en Julio del 2005 como Django Web Framework. El slogan del framework fue “Django, El framework para perfeccionistas con fechas límites” <sup>1</sup>.

En junio del 2008 fue anunciada la creación de la Django Software Foundation, la cual se hace cargo hasta la fecha del desarrollo y mantenimiento.

---

<sup>1</sup> Del ingles “The Web framework for perfectionists with deadlines”

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos que puede incluirse como parte de cualquier proyecto; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno (sistema de logging o bitácora), y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

Con Django también se distribuyen aplicaciones que proporcionan un sistema de comentarios, herramientas para syndicar contenido via RSS y/o Atom, “páginas planas” que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs.

Django como framework de desarrollo consiste en un conjunto de utilidades de consola que permiten crear y manipular proyectos y aplicaciones.

### 1.5.1 Estructuración de un proyecto en Django

El framework parte de la base que un proyecto está compuesto por un conjunto de aplicaciones. Un proyecto es un paquete que contiene 3 archivos:

- `manage.py`
- `urls.py`
- `settings.py`

Cuando django se encuentra instalado, existe un comando llamado `django-admin.py` en el PATH del sistema. Mediante este comando se pueden crear proyectos:

```
$ django-admin.py startproject mi_proyecto # Crea el proyecto mi_proyecto
```

El proyecto funciona como un contenedor de aplicaciones que se rigen bajo la misma base de datos, los mismos templates y las mismas clases de middleware.

Una aplicación es un paquete que contiene al menos los módulos:

- `models.py`
- `views.py`

y generalmente suele agregarse un módulo:

- `urls.py`

Para crear una aplicación, se puede bien utilizar el comando `django-admin.py` en la carpeta del proyecto, o el módulo `manage.py` que se explica más abajo.

```
$ django-admin.py startapp mi_app # Crear una aplicacion
```

### Módulo settings

Este módulo define la configuración del proyecto, siendo sus atributos principales la configuración de la base de datos a utilizar, la ruta en la cual se encuentran los medios estáticos, cuál es el nombre del archivo raíz de urls (generalmente `urls.py`). Otros atributos son las clases middleware, las rutas de los templates, el idioma para las aplicaciones que soportan *i18n*, etc.

Al ser un módulo del lenguaje python, la configuración se puede editar muy fácilmente a diferencia de configuraciones realizadas en XML, además de contar con la ventaja de poder configurar en caliente algunos parametros que así lo requieran.



Un parametro fundamental es la lista denominada `INSTALLED_APPS` que contiene los nombres de las aplicaciones instaladas en le proyecto.

## Módulo manage

Esta es la interfase con el framework. Este módulo permite crear aplicaciones, generar el SQL necesario para crear en la base de datos las tablas fruto de la escritura de los modelos, testear que los modelos de una aplicación estén bien definidos (validación), iniciar el servidor de desarrollo, crear volcados de la base de datos y restaurarlos restaurarlos (`emph{fixtures}`), utilizados en casos de pruebas y para precarga de datos conocidos).

Por ejemplo, para generar en la base de datos a patir de las aplicaciones inestalladas se ejecuta el siguiente comando:

```
manage.py syncdb # Generar las tablas
```

## Módulo urls

Este nombre de módulo aparece a nivel proyecto, pero también puede aparecer a nivel aplicación. Su misión es definir las asociaciones entre URLs y vistas, de manera de que el framework sepa que vista utilizar en función de la URL que está requiriendo el cliente. Las URLs se escriben mediante expresiones regulares. Se suele aprovechar la posibilidad del modulo de expresiones regulares del lenguaje python, que permite recuperar grupos nombrados (en contraposición al enfoque ordinal tradicional).

La asociación url-vistas se define en el módulo bajo el nombre *urlpatterns*. También es posible derivar el tratado de una parte de la expresión regular a otro módulo de urls. Generalmente esto ocurre cuando se desea delegar el tratado de las urls a una aplicación particular.

**Ej:** Derivar el tratado de todo lo que comience con la cadena `personas` a al módulo de urls de la aplicación `personas`.

```
(r'^personas', include('mi_proyecto.personas.urls'))
```

## Módulo models

Cada vez que se crea una aplicación, se genera un módulo `models.py`, en el cual se le permite al programador definir modelos de objetos, que luego son transformados en tablas relacionales <sup>2</sup>.

## Módulo views

Cada aplicacion posee un módulo `views`, donde se definen las funciones que atienden al cliente y son activadas gracias a el mapeo definido en el módulo `urls` del proyecto o de la aplicación.

Las funciones que trabajan como vistas deben recibir como primer parámetro el `request` y opcionalmente parámetros que pueden ser recuperados del mapeo de urls.

Dentro del módulo de urls

```
# Tras un mapeo como el siguiente
(r'^persona/(?P<id_persona>\d)/$', mi_vista)
# la vista se define como
def mi_vista(request, id_persona):
    persona = Personas.objects.get(id = id_persona)
```

<sup>2</sup> Mediante el comando `syncdb` del módulo `manage` del proyecto

```
datos = {'persona': persona, }  
return render_to_response('plantilla.html', datos)
```

### El ciclo de una petición

Cada vez que un browser realiza una petición a un proyecto desarrollado en django, la petición HTTP pasa por varias capas.

Inicialmente atraviesa los Middlewares, en la cual, el middleware de Request, empaqueta las variables del request en una instancia de la clase Request.

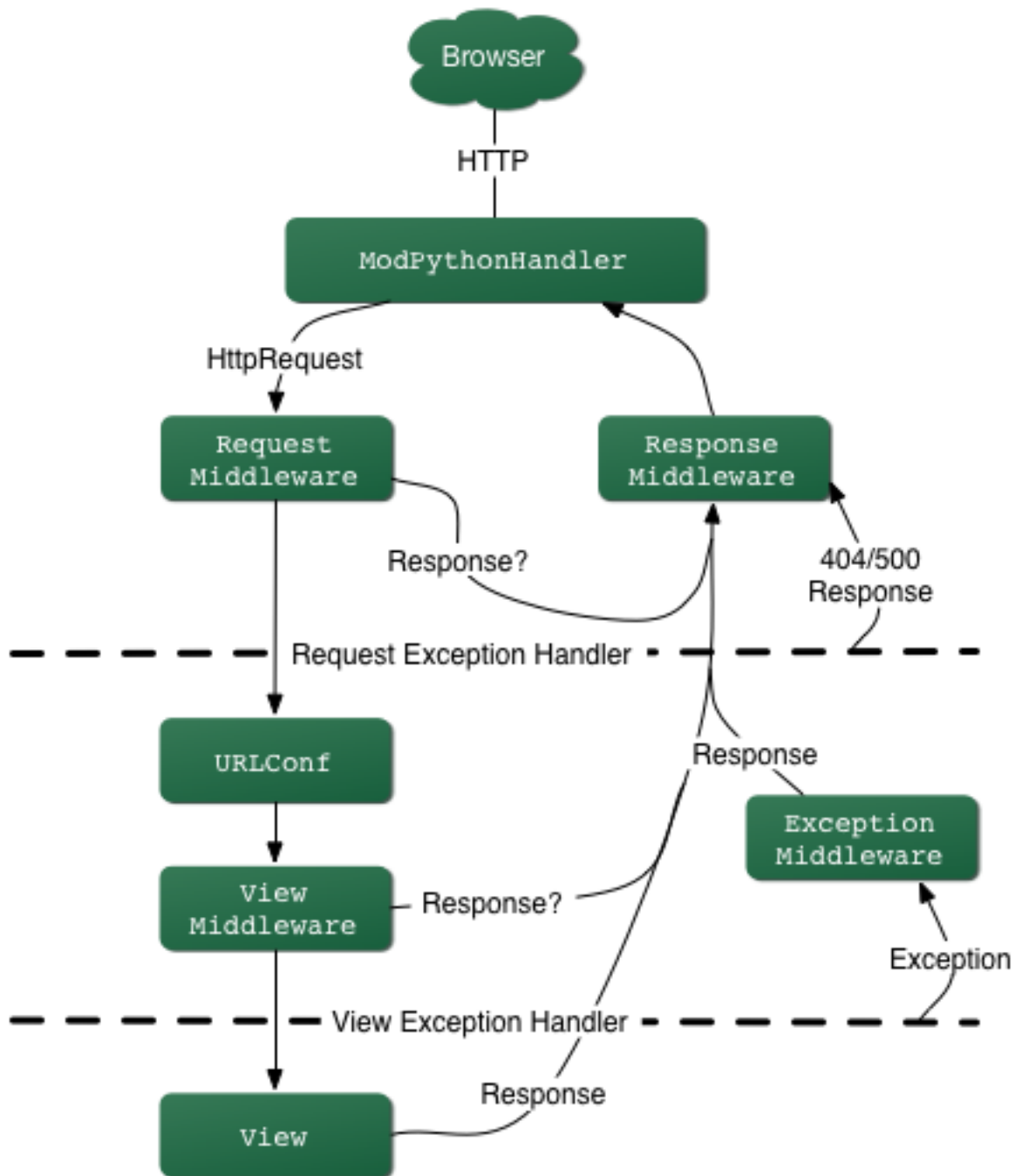
Luego de atravesar los middlewares de request, mediante las definiciones de URLs, se selecciona la vista a ser ejecutada.

Una vista es una función que recibe como primer argumento el request y opcionalmente una serie de parámetros que puede recuperar de la propia URL.

Dentro de la vista se suelen hacer llamadas al ORM, para realizar consultas sobre la base de datos. Una vez que la vista ha completado la lógica, genera un mapeo que es transferido a la capa de templates.

El template rellena sus comodines en función de los valores del mapeo que le entrega la vista. Un template puede poseer lógica muy básica (bifurcaciones, bucles de repetición, formateo de datos, etc).

El template se entrega como un HttpResponse. La responsabilidad de la vista es entregar una instancia de esta clase.



## 1.6 El mapeador Objeto-Relacional de Django

Diego



---

# Glosario

---

**API** [Application-Programming-Interface](#); conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

**DOM** [Document-Object-Model](#); interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

**JSON** [JavaScript-Object-Notation](#); formato ligero para el intercambio de datos.

**RPC** [Remote-Procedure-Call](#); es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

**field** An attribute on a [model](#); a given field usually maps directly to a single database column.

**generic view** A higher-order [view](#) function that abstracts common idioms and patterns found in view development and abstracts them.

**model** Models store your application's data.

**MTV** hola

**MVC** [Model-view-controller](#); a software pattern.

**project** A Python package – i.e. a directory of code – that contains all the settings for an instance of Django. This would include database configuration, Django-specific options and application-specific settings.

**property** Also known as “managed attributes”, and a feature of Python since version 2.2. From [the property documentation](#):

Properties are a neat way to implement attributes whose usage resembles attribute access, but whose implementation uses method calls. [...] You could only do this by overriding `__getattr__` and `__setattr__`; but overriding `__setattr__` slows down all attribute assignments considerably, and overriding `__getattr__` is always a bit tricky to get right. Properties let you do this painlessly, without having to override `__getattr__` or `__setattr__`.

**queryset** An object representing some set of rows to be fetched from the database.

**slug** A short label for something, containing only letters, numbers, underscores or hyphens. They're generally used in URLs. For example, in a typical blog entry URL:

`http://www.djangoproject.com/weblog/2008/apr/12/spring/`

the last bit (`spring`) is the slug.

**template** A chunk of text that separates the presentation of a document from its data.

**view** A function responsible for rendering a page.

**BSD** ve ese de

**i18n** La internacionalización es el proceso de diseñar software de manera tal que pueda adaptarse a diferentes idiomas y regiones sin la necesidad de realizar cambios de ingeniería ni en el código. La localización es el proceso de adaptar el software para una región específica mediante la adición de componentes específicos de un locale y la traducción de los textos, por lo que también se le puede denominar regionalización. No obstante la traducción literal del inglés es la más extendida.

---

# Indices, glosario y tablas

---

- *Index*
- *Module Index*
- *Glosario*





---

# Índice

---

## A

API, [9](#)

## B

BSD, [10](#)

## D

DOM, [9](#)

## F

field, [9](#)

## G

generic view, [9](#)

## I

i18n, [10](#)

## J

JSON, [9](#)

## M

model, [9](#)

MTV, [9](#)

MVC, [9](#)

## P

project, [9](#)

property, [9](#)

## Q

queryset, [9](#)

## R

RPC, [9](#)

## S

slug, [9](#)

## T

template, [10](#)

## V

view, [10](#)