

django

Jorge Bastida

me@jorgebastida.com

@jorgebastida

Jaime Irurzun

jaime@irurzun.com

@jaimeirurzun

django

The Web framework for perfectionists with deadlines.

Índice

1. Python

- a. Introducción
- b. Tipos de datos
- c. Operadores
- d. Usos frecuentes
- e. Estructuras
- f. Sentencias
- g. Ejercicio

2. Django

- a. Introducción
- b. URLs y Vistas
- c. Templates
- d. Modelo
- e. Administración
- f. Formularios
- g. Magia avanzada



Proyecto

Evolución de la Web



Desarrollo Web

1ª Generación

HTML
CGI

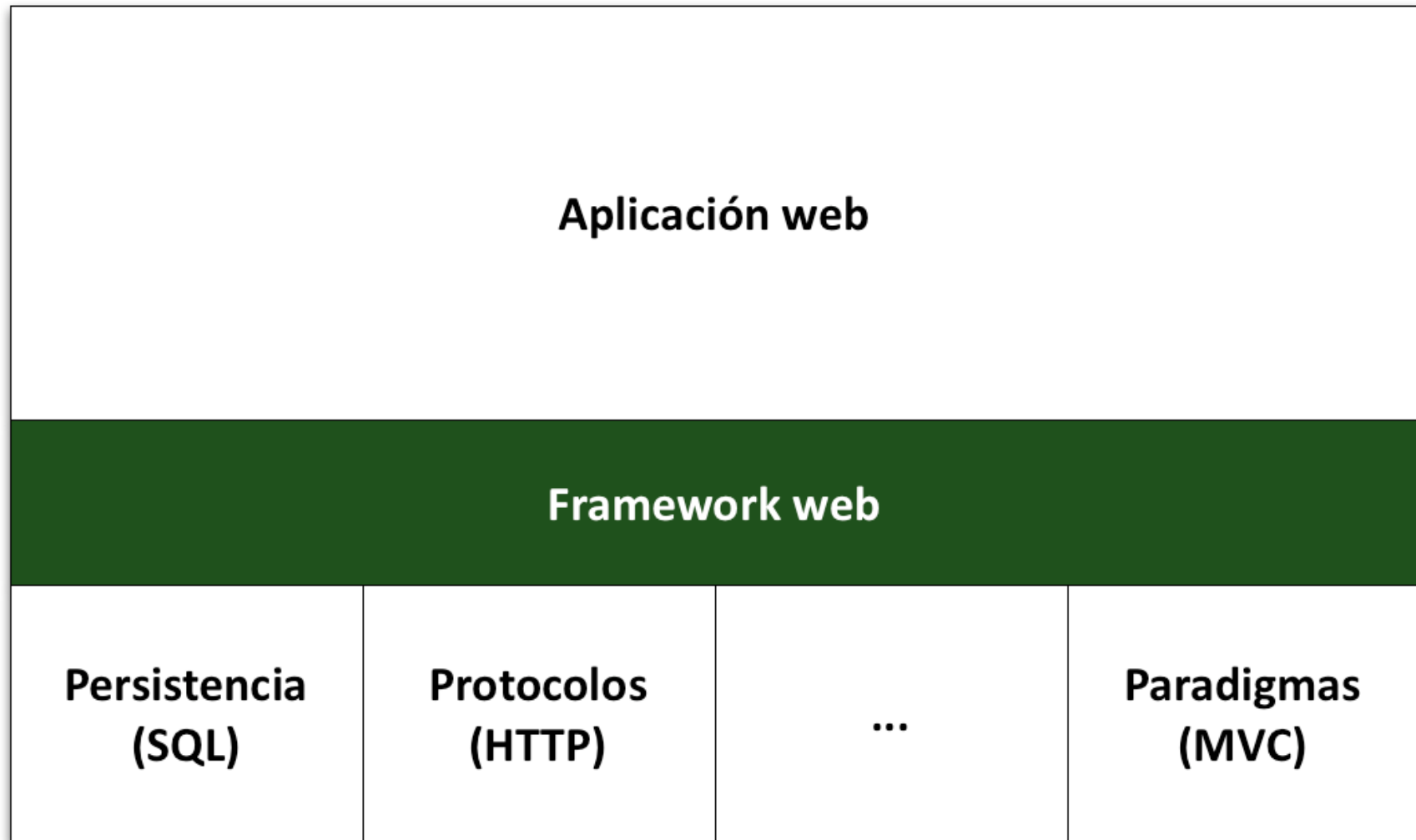
2ª Generación

PHP
ASP
JSP
...

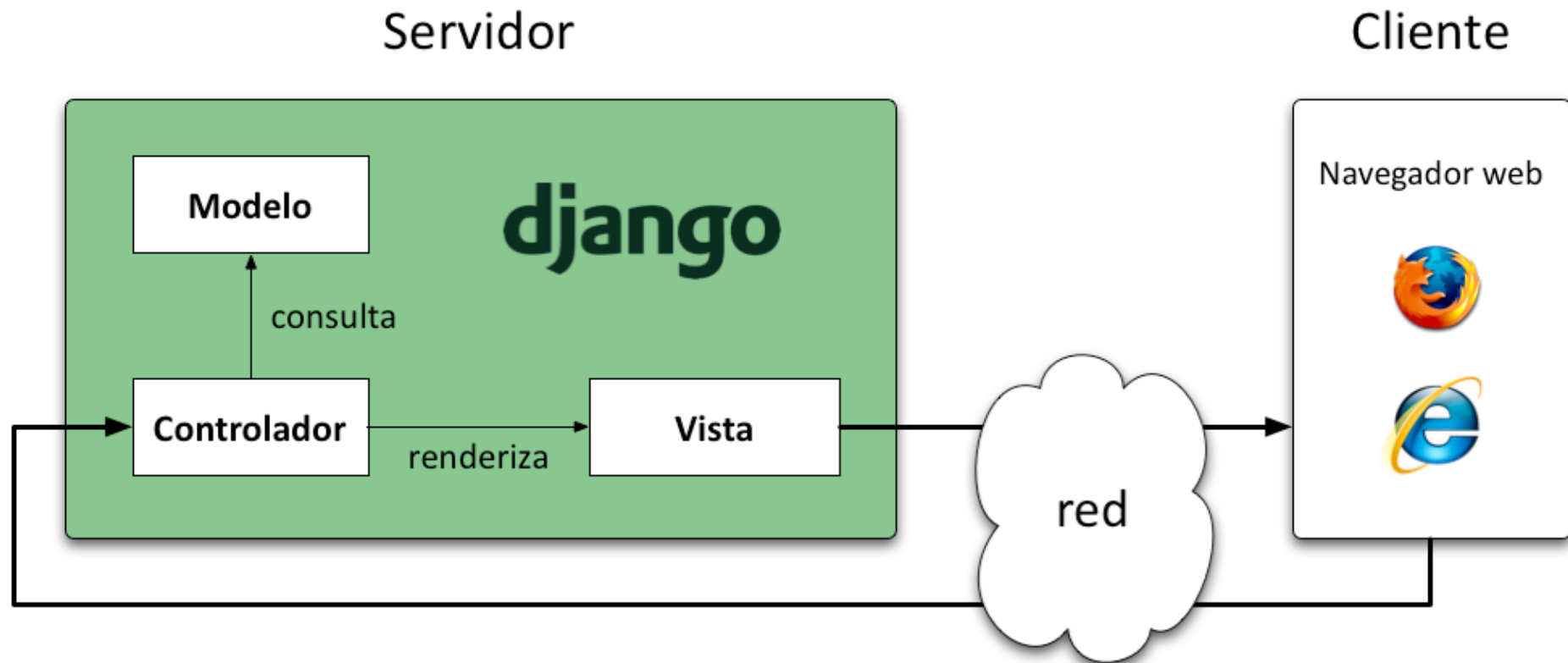
3ª Generación

Django
Rails
Symfony
...

Frameworks web



Django: Qué y dónde



Filosofía

- Loose coupling, Acoplamiento débil.
 - Cada capa es independiente y desconoce completamente a las demás.
- Menos código.
- Rápido desarrollo.
 - Esto es el siglo 21, todo el trabajo tedioso hay que evitarlo.
- Don't Repeat Yourself (DRY)

“Every distinct concept and/or piece of data should live in one, and only one, place. Redundancy is bad. Normalization is good.”

La comunidad

- **django-users**

23.000 miembros

- **django-developers**

7.400 miembros

- **djangoproject.com**

+500.000 visitas únicas mensuales



2. Instalación SGBD

Configurar un motor de Base de Datos:

- Oficiales:
 - PostgreSQL
 - MySQL
 - Oracle
 - **SQLite**
- Comunidad:
 - Sybase SQL Anywhere
 - IBM DB2
 - Microsoft SQL Server 2005
 - Firebird
 - ODBC



3. Instalación Django

A: Paquetes de cada distro

- `apt-get install python-django`

B: Official Release

- <http://www.djangoproject.com/download/>
- `python setup.py install`

C: Github

- `git clone https://github.com/django/django.git`

All Right?

```
>>> import django  
>>> django.VERSION  
(1, 4, 2, 'final', 0)
```



```
>>> import django  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ImportError: No module named django
```



Ficheros y Carpetas

¿Es Django tán simple y fácil de usar?

	Rails	Symfony	Django
Ficheros	149	117	5
Carpetas	35	29	2

* Incluida la carpeta del proyecto



Crear nuestro proyecto

de 5 ficheros ;-)

```
$ django-admin.py startproject dwitter
```

```
.  
├── dwitter  
│   ├── __init__.py  
│   ├── settings.py  
│   ├── urls.py  
│   └── wsgi.py  
└── manage.py
```

- ¿Esto es un proyecto... ? **Si os ve mi jefe...**
- **Sí**, disponemos de un proyecto '*funcional*' en django.

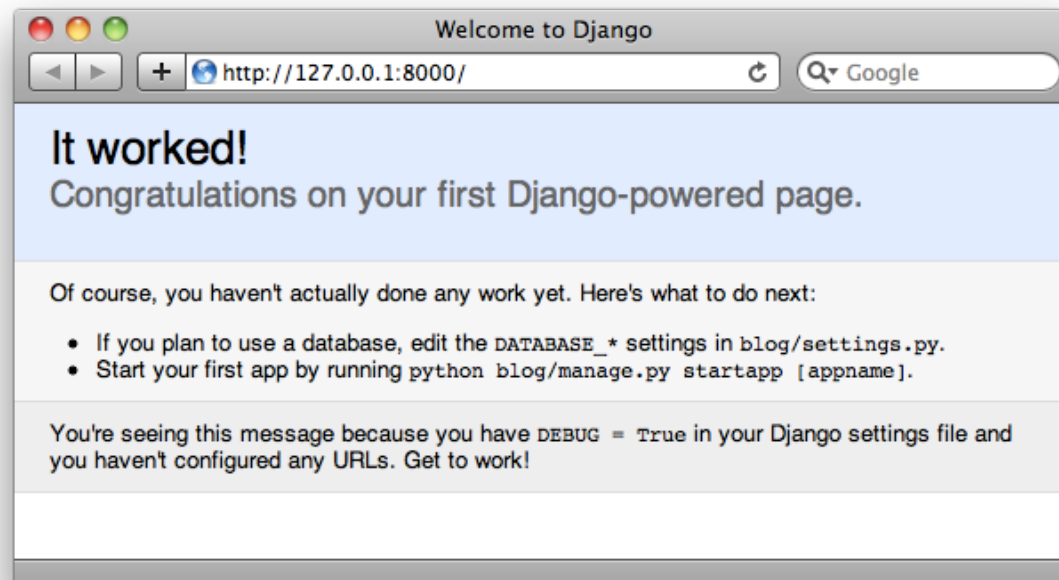
Arrancar nuestro proyecto

de 5 ficheros ;-)

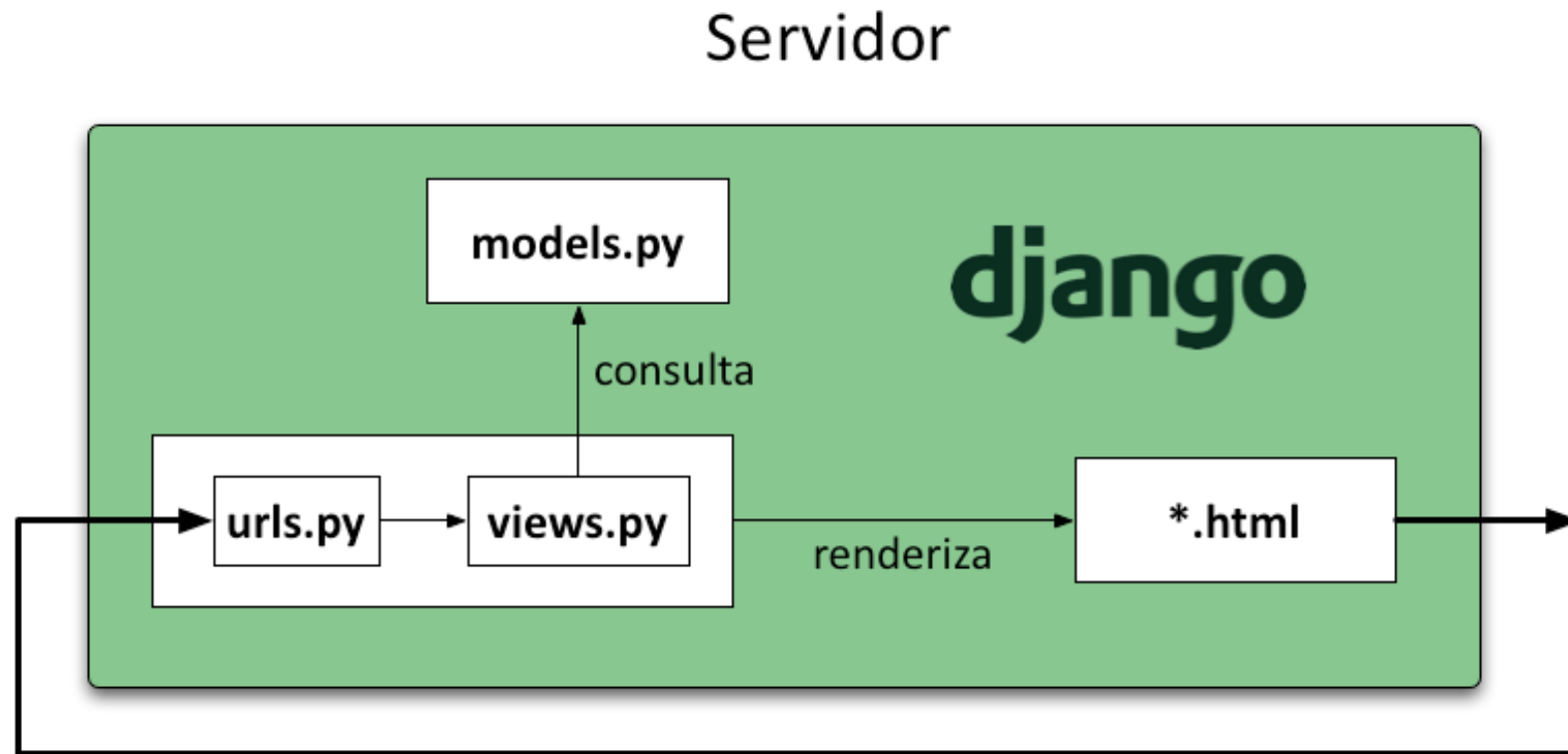
```
$ cd dwitter
$ python manage.py runserver

Validating models...
0 errors found

Django version 1.4, using settings 'dwitter.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```



MVC en Django



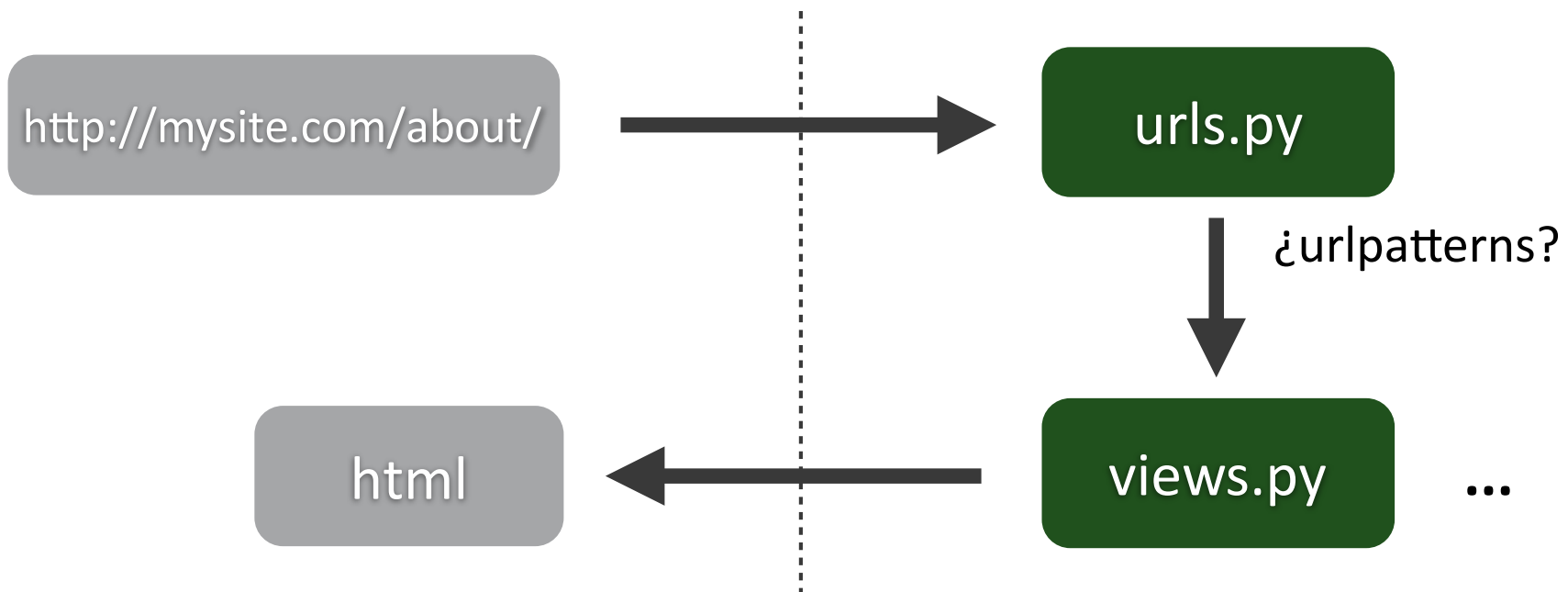
Modelo = Model

Vista = Template

Controlador = URL+View

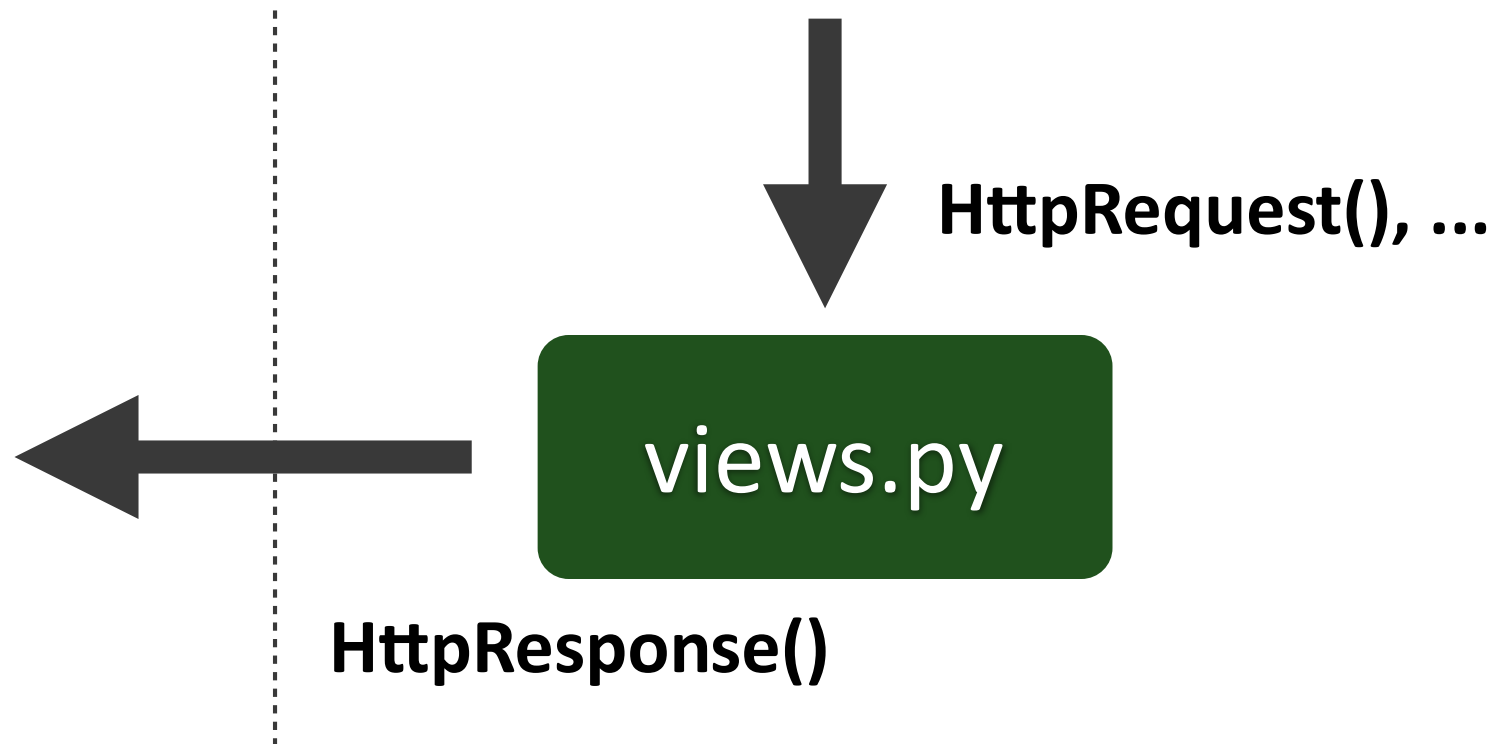
URLs y Vistas

- El fichero **urls.py** actúa como puerta de entrada para las peticiones HTTP
- Se definen URLs elegantes mediante expresiones regulares que redirigen a funciones de **views.py**



URLs y Vistas

- La función de `views.py` recibe como parámetros un objeto **HttpRequest** y todos los parámetros de la URL capturados, teniendo que devolver siempre un objeto **HttpResponse**



URLs y Vistas

Ejemplo 1: `http://mysite.com/time`

urls.py

```
from django.conf.urls.defaults import *
from mysite.views import hora_actual

urlpatterns = patterns('',
    url(r'^time/$', hora_actual),
)
```

views.py

```
from django.http import HttpResponse
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    html = "Son las %s." % now
    return HttpResponse(html)
```

URLs y Vistas

Ejemplo 2: `http://mysite.com/time/plus/2`

urls.py

```
from django.conf.urls.defaults import *
from mysite.views import dentro_de

urlpatterns = patterns('',
    url(r'^time/plus/(\d{1,2})/$', dentro_de),
)
```

views.py

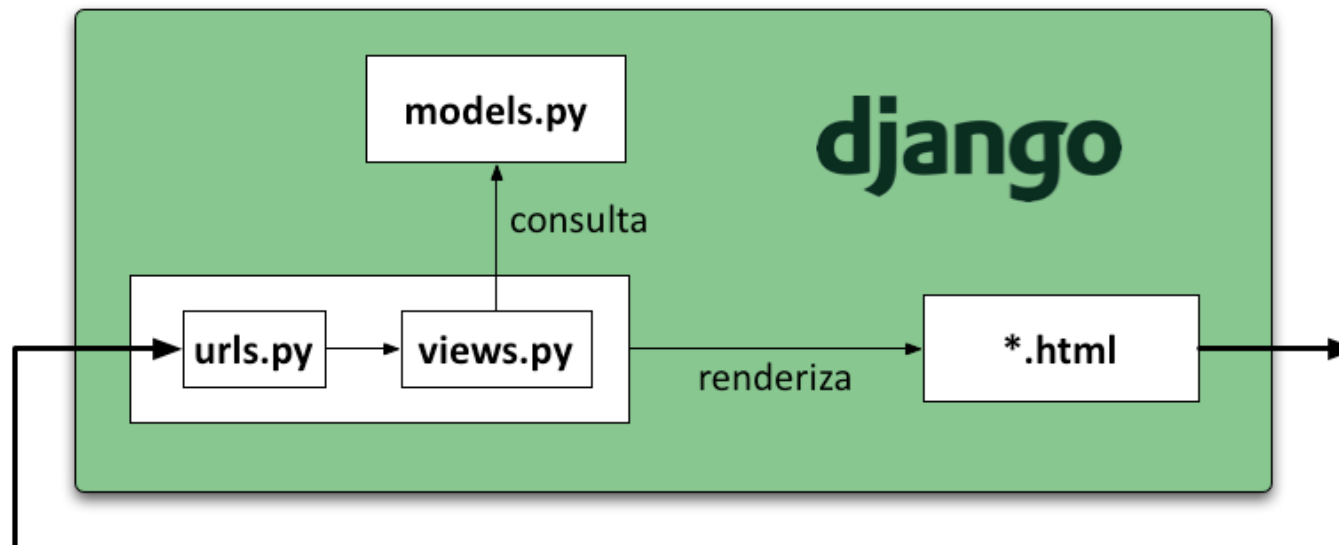
```
from django.http import HttpResponse
from datetime import datetime, timedelta

def dentro_de(request, offset):
    offset = int(offset)
    dt = datetime.now() + timedelta(hours=offset)
    html = "En %i hora(s), serán las %s." % (offset, dt)
    return HttpResponse(html)
```

Templates

- Separan la lógica de **presentación** a una capa independiente.
 - Ficheros independientes (.html)
 - Lenguaje independiente (¡para diseñadores!)

Servidor



Templates

- Se basan en dos tipos de objetos: `Template()` y `Context()`.
 - Un objeto **`Template()`** contiene el **string** de salida que queremos devolver en el `HttpResponse` (normalmente HTML), pero incluyendo etiquetas especiales de Django.
 - Un objeto **`Context()`** contiene un **diccionario** con los valores que dan contexto a una plantilla, los que deben usarse para renderizar un objeto `Template()`.

Template:

```
"Bienvenido, {{ user }}."
```

Context:

```
{ 'user': 'alatar' }
```



```
"Bienvenido, alatar."
```

Templates

- Primera aproximación al objetivo: **Template + Context**

```
from django.http import HttpResponse
from django.template import Template, Context
from datetime import datetime

PLANTILLA = """<html><body>
Son las {{ hora }}.
</body></html>"""

def hora_actual(request):
    now = datetime.now()
    t = Template(PLANTILLA)
    c = Context({'hora': now})
    html = t.render(c)
    return HttpResponse(html)
```

Templates

- Segunda aproximación al objetivo: **open()**, **read()**, **close()**

```
from django.http import HttpResponse
from django.template import Template, Context
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    fp = open('/home/django/templates/hora.html')
    t = Template(fp.read())
    fp.close()
    c = Context({'hora': now})
    html = t.render(c)
    return HttpResponse(html)
```

Templates

- Segunda aproximación al objetivo: `open()`, `read()`, `close()`

```
from django.http import HttpResponse
from django.template import Template, Context
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    fp = open('/home/django/templates/hora.html')
    t = Template(fp.read())
    fp.close()
    c = Context({'hora': now})
    html = t.render(c)
    return HttpResponse(html)
```


Templates

- Tercera aproximación al objetivo: `get_template()`

settings.py



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

```
from django.http import HttpResponse  
from django.template.loader import get_template  
from django.template import Context  
from datetime import datetime  
  
def hora_actual(request):  
    now = datetime.now()  
    t = get_template('hora.html')  
    c = Context({'hora': now})  
    html = t.render(c)  
    return HttpResponse(html)
```

Templates

- Tercera aproximación al objetivo: `get_template()`

settings.py



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

```
from django.http import HttpResponseRedirect  
from django.template.loader import get_template  
from django.template import Context  
from datetime import datetime  
  
def hora_actual(request):  
    now = datetime.now()  
    t = get_template('hora.html')  
    c = Context({'hora': now})  
    html = t.render(c)  
    return HttpResponseRedirect(html)
```

Templates

- Objetivo alcanzado: **render()**

```
from django.shortcuts import render
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    return render(request, 'hora.html', {'hora': now})
```



Templates

- Objetivo alcanzado: **render()**

```
from django.shortcuts import render
from datetime import datetime

def hora_actual(request):
    now = datetime.now()
    return render(request, 'hora.html', {'hora': now})
```



AWESOME!

Templates: Tip

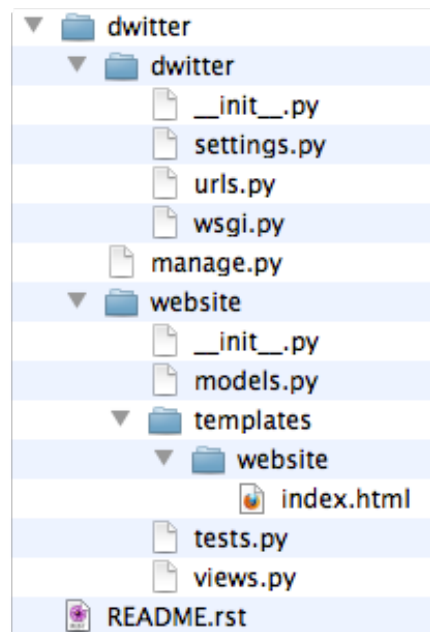
a) settings.py



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

Reusable
apps?

b)



Alternativa reutilizable:

```
TEMPLATE_LOADERS = (  
    'django.template.loaders.filesystem.Loader',  
    'django.template.loaders.app_directories.Loader',  
)
```

- La carpeta **/templates** es buscada dentro de cada app.
- Conviene incluir una carpeta con el nombre de la app por claridad.

```
return render(request, 'website/index.html')
```

Templates en detalle

- **Si**, otro sistema de templates
 - Smarty, Tiles, ClearSilver ...
- Describen cuál va a ser el resultado que ven los usuarios.
 - Desacoplado de Python (Diseñadores muy lejos de Python)
- HTML (o no)... con **esteroides**.
- Muy sencillo de aprender
 - KISS: Keep It Simple, Stupid
- Muy sencillo de extender

Filosofía y Limitaciones

- La sintaxis debe estar desacoplada del HTML/XML.
- Los diseñadores saben HTML.
- Los diseñadores no saben Python.
- No consiste en inventarse un lenguaje.
- Una variable no puede cambiar el valor de una variable.
- Una template no puede ejecutar código Python.

Templates: {{}}

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, {{ username }}.
  </body>
</html>
```

```
{'username': 'juan'}
```

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, juan.
  </body>
</html>
```


Filters y Tags

filter

```
{{ variable|filter }}
```

inline tag

```
{% tag var1 var2 %}
```

block tag

```
{% tag var1 %}  
...  
{% endtag %}
```

Templates: tags {% %}

comment

```
{% comment %} Bu! {% endcomment %}
```

for

```
{% for elemento in lista %}  
    <li>{{ elemento }}</li>  
{% endfor %}
```

if

```
{% if username == "Juan" %}  
    Hola Juan, me gustas!  
{% else %}  
    Hola {{ username }},  
{% endif %}
```

```
== != > < >= <=  
in and or not
```

Templates: tags {% %}

cycle

```
{% for elemento in lista %}  
    <li class="{% cycle 'rojo' 'azul' %}">{{ elemento }}</li>  
{% endfor %}
```

include

```
{% include "foo/bar.html" %}
```

forloop

```
{% for elemento in lista %}  
    <li>{{ forloop.counter }}.{{ elemento }}</li>  
{% endfor %}
```

empty

```
{% for elemento in lista %}  
    <li class="{% cycle 'rojo' 'azul' %}">{{ elemento }}</li>  
{% empty %}  
    Sin elementos.  
{% endfor %}
```

Templates: Filters

title

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, {{ username|title }}.
  </body>
</html>
```

```
{ 'username': 'juan' }
```

```
<html>
  <head>Ejemplo templates</head>
  <body>
    Hola, Juan.
  </body>
</html>
```

Templates: Filters

```
{'username': 'Juan es majo'}
```

length

```
{{ username|length }}
```

12

cut

```
{{ username|cut }}
```

Juanesmaj

slugify

```
{{ username|slugify }}
```

juan-es-majo

wordcount

```
{{ username|wordcount }}
```

3

upper

```
{{ username|upper }}
```

JUAN ES MAJO

```
{'username': None}
```

default

```
{{ username|default:"Desconocido" }}
```

Desconocido

Templates: Filters

```
{'username': 'Juan es <b>majo, guapo y <em>listo</em></b>'}
```

striptags

```
{{ username|striptags }}
```

```
Juan es majo guapo y listo
```

truncatewords_html

```
{{ username|truncatewords_html:4 }}
```

```
Juan es <b>majo guapo</b> ...
```

removetags

```
{{ username|removetags:"em a br" }}
```

```
Juan es <b>majo guapo y listo</b>
```

Templates: Filters

```
{'url': 'Visitad http://www.djangoproject.com'}
```

urlize

```
{{ url|urlize }}
```

```
Visitad <a href="http://www.djangoproject.com"> http://  
www.djangoproject.com </a>
```

urlizetrunc

```
{{ url|urlizetrunc:16 }}
```

```
Visitad <a href="http://www.djangoproject.com"> http://  
www.djang... </a>
```

Templates: Filters

```
{'lista': ['States', ['Kansas', ['Lawrence', 'Topeka'], 'Illinois']]}
```

unordered_list

```
{{ lista|unordered_list }}
```

```
<li>States
  <ul>
    <li>Kansas
      <ul>
        <li>Lawrence</li>
        <li>Topeka</li>
      </ul>
    </li>
    <li>Illinois</li>
  </ul>
</li>
```


Templates: Filters

```
{'value': 123456789}
```

add

```
{{ value|add:"1" }}
```

123456790

filesizeformat

```
{{ value|filesizeformat }}
```

117.7MB

```
{'date': datetime.datetime(2010, 9, 11, 17, 1, 59, 385323) }
```

date

```
{{ date|date:"d M Y" }}
```

11 Sep 2010

timesince

```
{{ date|timesince }}
```

4 days, 6 hours

timeuntil

```
{{ date|timeuntil }}
```

1 days, 6 hours

Herencia de Templates

base.html

```
<html>
  <head>
    <title>Mi página personal</title>
  </head>
  <body>
    {% block content %}
      Contenido por defecto.
    {% endblock %}
  </body>
</html>
```

base.html



hija.html

hija.html

```
{% extends "base.html" %}
{% block content %}
  Hola desde la portada.
{% endblock %}
```

Ejemplo SQL

```
def book_list(request):  
    try:  
        db = MySQLdb.connect(user='me', db='mydb',  
                               passwd='secret', host='localhost')  
        cursor = db.cursor()  
        cursor.execute('SELECT nama FROM books ORDER BY name')  
        names = []  
        for row in cursor.fetchall():  
            names.append(row[0])  
        db.close()  
    except:  
        return render_to_response('500.html')  
    return render_to_response('book_list.html', {'names':names})
```

Ejemplo ORM

```
def book_list(request):  
    1 names = Books.objects.all().order_by('name')  
    2  
    return render(request, 'book_list.html', {'names':names})
```

Modelo

```
from django.db import models 1
class Books(models.Model): 2
    name = models.CharField(blank=True, max_length=100) 3 4
    created = models.DateTimeField(blank=False) 5
    available = models.BooleanField(default=True) 6
```

- **Independencia SGBD!**
 - Definimos estructuras de información **genéricas**.
 - Definimos **restricciones** (*notnull, blank, max_lenght...*)
- **Única** definición del modelo (*configuración, mapeo a db*)

Propiedades de las Field

- `null` (`True|False`)
- `blank` (`True|False`)
- `choices` (`lista`)
- `default` (`valor`)
- `editable` (`True|False`)
- `help_text` (`String`)
- `unique` (`True|False`)
- `primary_key`
- `unique_for_date`
- `unique_for_month`
- `unique_for_year`

¿Es magia? No.

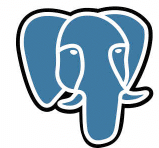
```
$ python manage.py sqlall website
```

```
BEGIN;  
CREATE TABLE "website_books" (  
    "id" integer NOT NULL PRIMARY KEY,  
    "name" varchar(100) NOT NULL,  
    "created" datetime NOT NULL,  
    "available" bool NOT NULL  
);  
COMMIT;
```

SQLite

```
BEGIN;  
CREATE TABLE "website_books" (  
    "id" serial NOT NULL PRIMARY KEY,  
    "name" varchar(100) NOT NULL,  
    "created" timestamp with time zone NOT NULL,  
    "available" boolean NOT NULL  
);  
COMMIT;
```

PostgreSQL



¿Es magia? No.

- Nombres de tablas generados automáticamente.
 - `<app_name>_lower(<model_name>)`
- **id** como Primary Key (*Personalizable*)
- Las Foreign Key terminan en **_id** (*Personalizable*)
- Los tipos de datos se ajustan en función del SGBD

Configurar settings.py

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

'postgresql_psycopg2',
'postgresql', 'mysql',
'sqlite3' or 'oracle'.

Creando Tablas

- Crea las tablas para **todos** los modelos de las **apps instaladas** en el fichero settings.py
- **No actualiza esquemas** si la tabla existe.
 - Eliminar tabla y volver a ejecutar **syncdb**

Creando Tablas

```
$ python manage.py syncdb
```

- Crea las tablas para **todos** los modelos de las **apps instaladas** en el fichero settings.py
- **No actualiza esquemas** si la tabla existe.
 - Eliminar tabla y volver a ejecutar **syncdb**

syncdb

```
$ python manage.py syncdb
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table website_tweet
```

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 'neo'): admin

E-mail address: user@example.com

Password:

Password (again):

Superuser created successfully.

Installing index for auth.Permission model

Installing index for auth.Message model

Installing index for website.Tweet model

Previo: Clases del ORM

```
ts = Publisher.objects.all()
```

Model

Manager

QuerySet

INSERT

```
o = Model(...)    o.save()
```

a)

```
>>> p = Publisher(  
...     name='Apress',  
...     address='2855 Telegraph Avenue',  
...     city='Berkeley',  
...     state_province='CA',  
...     country='U.S.A.',  
...     website='http://www.apress.com/')  
>>> p.save()
```

```
manager.create(...)
```

b)

```
>>> p = Publisher.objects.create(  
...     name='O'Reilly',  
...     address='10 Fawcett St.',  
...     city='Cambridge',  
...     state_province='MA',  
...     country='U.S.A.',  
...     website='http://www.oreilly.com/')
```

UPDATE

```
o.save()
```

1

```
>>> ...  
>>> p.id  
52  
>>> p.name = 'Apress Publishing'  
>>> p.save()
```

```
queryset.update(...)
```

n

```
>>> Publisher.objects.all().update(country='USA')  
2
```

DELETE

```
o.delete()
```

1

```
>>> ...  
>>> p.id  
52  
>>> p.delete()
```

```
queryset.delete()
```

n

```
>>> ps = Publisher.objects.all()  
>>> ps.delete()
```


SELECT de 1 resultado

```
.get(...)
```

```
>>> Publisher.objects.get(name="Apress")  
<Publisher: Apress>
```

```
>>> Publisher.objects.get(name="Anaya")  
Traceback (most recent call last):  
...  
DoesNotExist: Publisher matching query does not exist.
```

```
>>> Publisher.objects.get(country="U.S.A.")  
Traceback (most recent call last):  
...  
MultipleObjectsReturned: get() returned more than one Publisher --  
it returned 2! Lookup parameters were {'country': 'U.S.A.'}
```

SELECT de N resultados

¡Devuelven
QuerySets, no
listas!

```
.all()
```

```
>>> Publisher.objects.all()
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

```
.filter(...)
```

```
>>> Publisher.objects.filter(
    country="U.S.A.", state_province="CA")
[<Publisher: Apress>]
```

```
.exclude(...)
```

```
>>> Publisher.objects.exclude(
    country="U.S.A.", state_province="CA")
[<Publisher: O'Reilly>]
```

get(), filter() y exclude()

```
Modelo.objects.filter(campo1="valor1", campo2="valor2")
```

Los parámetros pueden indicar mucho más que igualdad (=)



campo__exact=''
campo__iexact=''
campo__contains=''
campo__icontains=''
campo__isnull=T|F
campo__day=31

campo__gt=0
campo__gte=0
campo__lt=0
campo__lte=0
campo__in=[,]
campo__month=12

campo__startswith=''
campo__istartswith=''
campo__endswith=''
campo__iendswith=''
campo__range=(,)
campo__year=2010

ORDER BY

¡También
devuelve
QuerySets!

```
.order_by(...)
```



```
>>> Publisher.objects.order_by("name")  
[<Publisher: Apress>, <Publisher: O'Reilly>]
```



```
>>> Publisher.objects.order_by("-name")  
[<Publisher: O'Reilly>, <Publisher: Apress>]
```

Múltiples campos:

```
>>> Publisher.objects.order_by("-name", "country")  
[<Publisher: O'Reilly>, <Publisher: Apress>]
```

Slicing

[n:m]

```
>>> Publisher.objects.order_by("name")[0]  
<Publisher: Apress>
```

```
>>> Publisher.objects.order_by("name")[:2]  
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

LIMIT

```
>>> Publisher.objects.order_by("name")[1:2]  
[<Publisher: O'Reilly>]
```

OFFSET

Related Objects

OneToOneField

```
1 class Coche(models.Model):  
    motor = OneToOneField(Motor)
```

```
1 class Motor(models.Model):  
    ...
```

¿Cómo usamos la relación desde las instancias?

Gracias a nosotros

```
>>> c.motor  
<Motor: Motor object>
```

Gracias a **Django**

```
>>> m.coche  
<Coche: Coche object>
```

Related Objects

ForeignKeyField

```
1 class Blog(models.Model):  
    ...
```

```
n class Post(models.Model):  
    blog = ForeignKeyField(Blog)
```

¿Cómo usamos la relación desde las instancias?

Gracias a nosotros

```
>>> p.blog  
<Blog: Blog object>
```

Gracias a **Django**

```
>>> b.post_set.all()  
[<Post: Post object>, ...]
```

Related Objects

ManyToManyField

n

```
class Post(models.Model):  
    tags = ManyToManyField(Tag)
```

m

```
class Tag(models.Model):  
    ...
```

¿Cómo usamos la relación desde las instancias?

Gracias a nosotros

```
>>> p.tags.add(t1, t2)  
>>> p.tags.all()  
[<Tag: Tag object>, ...]
```

Gracias a **Django**

```
>>> t.post_set.add(p1, p2)  
>>> t.post_set.all()  
[<Post: Post object>, ...]
```


Related Objects

En todas ellas podemos **renombrar** el puntero inverso

```
1 class Blog(models.Model):  
    ...
```

```
n class Post(models.Model):  
    blog = ForeignKeyField(Blog, related_name='posts')
```

Gracias a nosotros

```
>>> p.blog  
<Blog: Blog object>
```

Gracias a **Django**

```
>>> b.posts.all()  
[<Post: Post object>, ...]
```

Cuando haya **2 relaciones** entre **2 modelos**, será **obligatorio**

Laziness

- Las consultas **sólo** se ejecutarán cuando realmente se necesite obtener los objetos. En las siguientes situaciones:

- Iteraciones

```
for p in Publisher.objects.all():
```

- Slicing

```
Publisher.objects.filter(country='USA')[0]
```

- Serialización

[Caché]

- `repr()`

```
[<Publisher: Publisher object>]
```

- `len()` !!!

```
len(Publisher.objects.all())
```

- `list()`


```
list(Publisher.objects.all())
```

- `bool()`

```
if Publisher.objects.filter(country='USA'):
```

Nota: `__unicode__()`

```
>>> Publisher.objects.all()  
[<Publisher: Publisher object>]
```



```
class Publisher(models.Model):  
    ...  
  
    def __unicode__(self):  
        return self.name
```



```
>>> Publisher.objects.all()  
[<Publisher: Apress>]
```

Profile

- **Problema:** El modelo *User* de *django.contrib.auth* no puede contener toda la información que necesitamos.
 - Username, Password, Name.... y poco más.
- **Solución:** Definir un **Profile** (Un Modelo Agregado) para guardar esa información.

Profile

```
class Profile(models.Model):  
  
    user = models.OneToOneField(User, unique=True)  
    bio = models.CharField(blank=True, max_length=200)  
    ...
```

models.py

- Cada objeto *User* de *django.contrib.auth* dispondrá de un atributo *profile* que nos permitirá acceder al *Profile*.

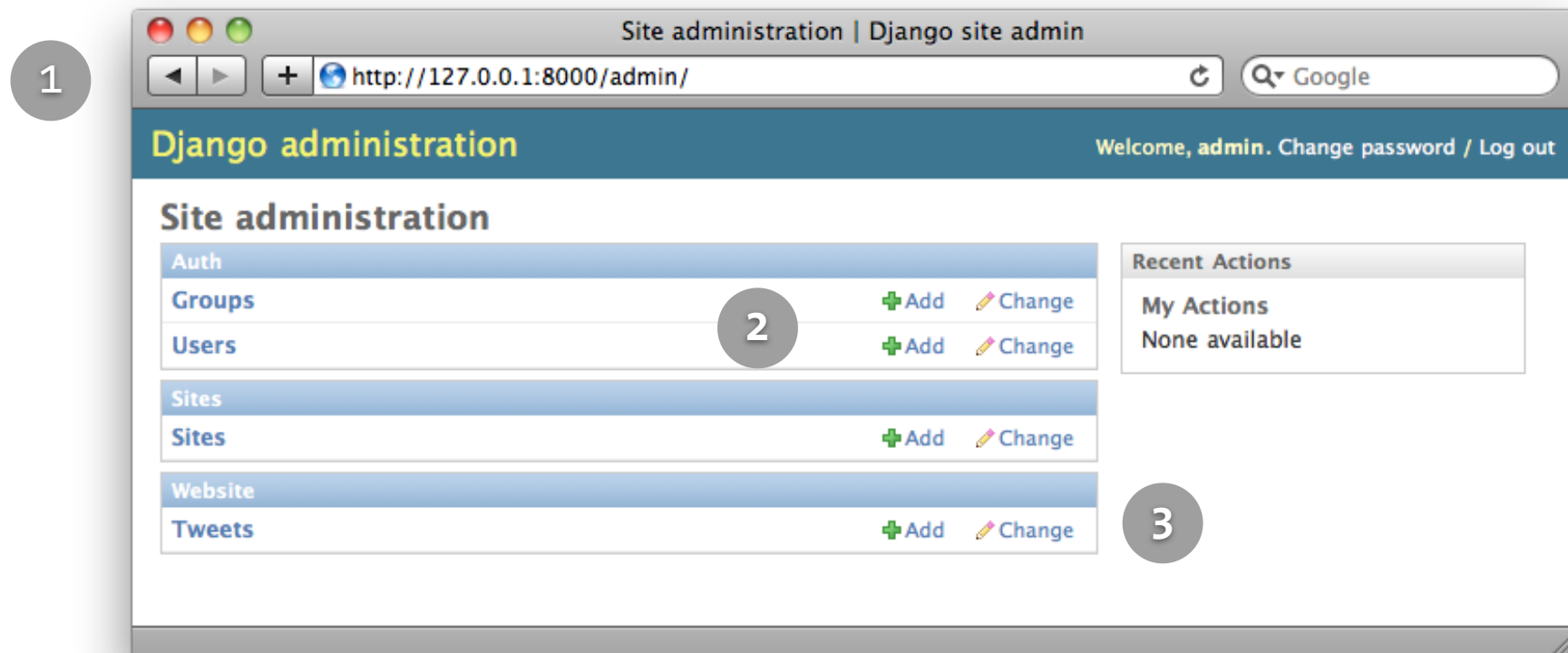
Profile

```
>>> from django.contrib.auth.models import User
>>> u = User.objects.get(id=1)
>>> type(u)
<class 'django.contrib.auth.models.User'>

>>> type(u.profile)
<class 'website.models.Profile'>
```

- Donde tengamos el User tendremos el Profile.
- Donde tengamos el Profile, tendremos el User.

django.contrib.admin



django.contrib.admin

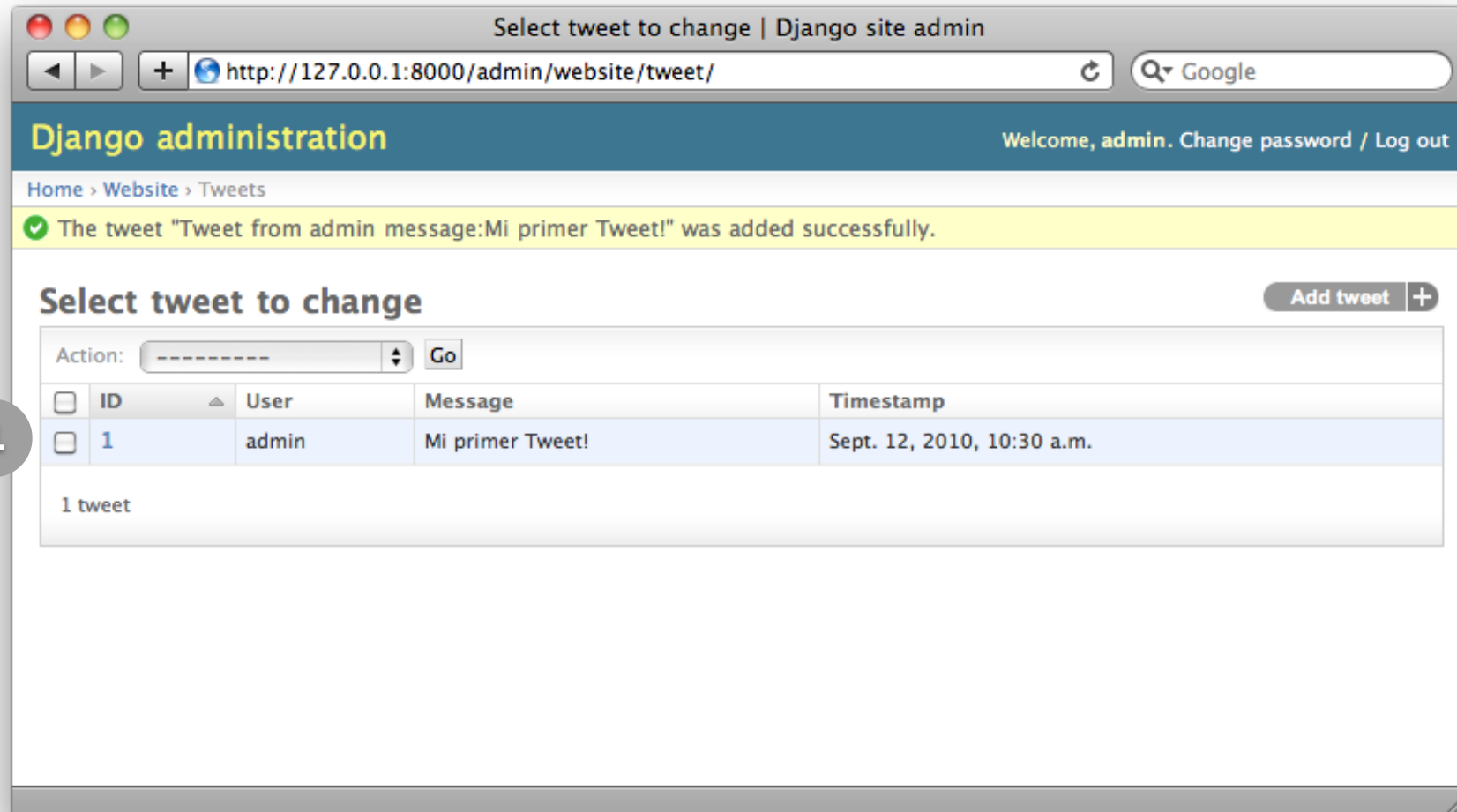
The screenshot shows a web browser window with the title "Add tweet | Django site admin". The address bar shows the URL "http://127.0.0.1:8000/admin/website/tweet/add/". The page header is "Django administration" with a welcome message "Welcome, admin. Change password / Log out". The breadcrumb trail is "Home > Website > Tweets > Add tweet". The main heading is "Add tweet".

1. The "Message:" field contains the text "Mi primer Tweet!".

2. The "User:" field shows a dropdown menu with "admin" selected and a plus sign to add a new user.

3. The bottom right corner contains three buttons: "Save and add another", "Save and continue editing", and "Save".

django.contrib.admin



Django admin: Instalación

urls.py

```
from django.conf.urls.defaults import *

# Uncomment the next two lines to enable the admin:
1 from django.contrib import admin
  admin.autodiscover()

urlpatterns = patterns('',
    ...
2 url(r'^admin/', include(admin.site.urls)),
    ...
)
```

Django admin: Instalación

settings.py

```
INSTALLED_APPS = (  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
    'django.contrib.admin', 1  
    ...  
)
```

Actualizando la BD

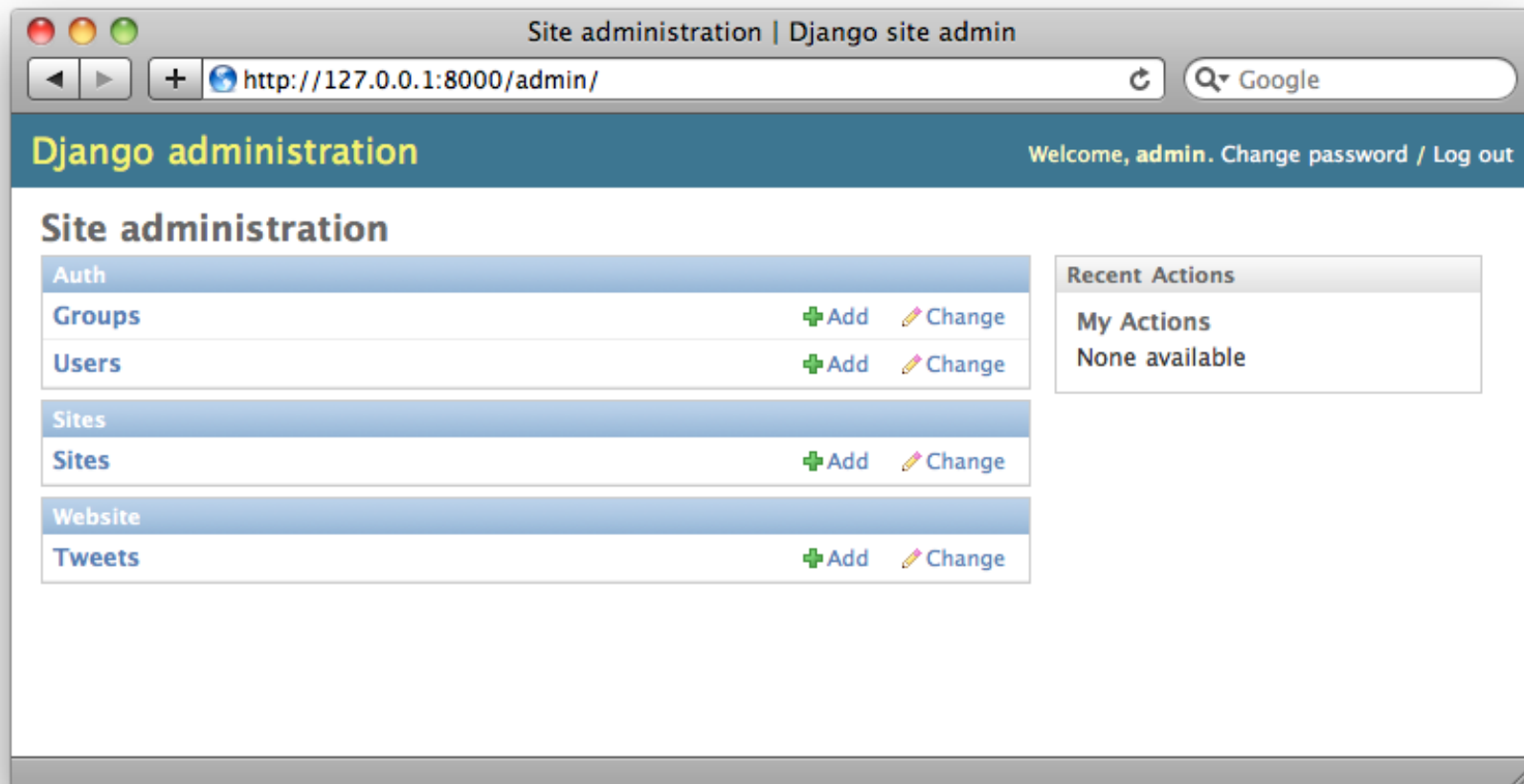
```
$ python manage.py syncdb
```

```
Creating table django_admin_log  
Installing index for admin.LogEntry model
```



Admin lista para usar

¿Y nuestra app?



admin.py

- Cada **app** debe de tener el suyo.
- Define qué modelos serán visibles desde el admin y permite personalizar su aspecto.

```
from django.contrib import admin
from website.models import Tweet

class TweetAdmin(admin.ModelAdmin):
    list_display = ('id', 'user', 'message', 'timestamp')

admin.site.register(Tweet, TweetAdmin)
```

Clases involucradas

Widget

Componente **visual** equivalente a HTML

TextInput

CheckboxInput



```
<input type='text'...>
```

```
<input type='checkbox'...>
```

Field

Lógica de un campo, asociado a un Widget

EmailField

IPAddressField



```
widget, initial, error, ...
```

Form

Conjunto de Fields de un formulario

```
ContactForm [nombre, email, telefono, mensaje, ...]
```

Fields

- BooleanField
- CharField
- ChoiceField
- TypedChoiceField
- DateField
- DateTimeField
- DecimalField
- EmailField
- FileField
- FilePathField
- FloatField
- ImageField
- IntegerField
- IPAddressField
- MultipleChoiceField
- NullBooleanField
- RegexField
- SlugField
- TimeField
- URLField
- ComboField
- MultiValuefield
- SplitDateTimeField
- ModelChoiceField
- ModelMultipleChoiceField

Creación de un Form

- **Paso 1/3:** Definición del formulario en **forms.py**

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100, label='Topic')
    email = forms.EmailField(required=False)
    message = forms.CharField(widget=forms.Textarea)
```

Creación de un Form

- **Paso 2/3:** Maquetación del formulario en su **template**

```
<html>
<body>
  <h1>Contact us</h1>

  {% if form.errors %}
    <p style="color: red;">
      Please correct the error{{ form.errors|pluralize }} below.
    </p>
  {% endif %}

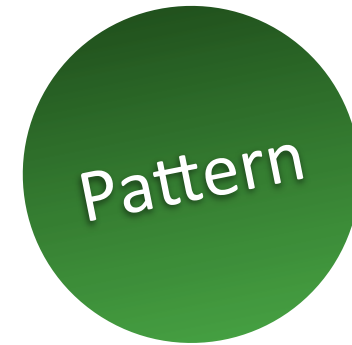
  <form action="" method="post">
    <table>
      {{ form.as_table }}
    </table>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Creación de un Form

- **Paso 3/3:** Programación de la vista en **views.py**

```
from django.shortcuts import render
from mysite.contact.forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            cd = form.cleaned_data
            send_mail(cd['subject'], cd['message'], ...)
            # ...
            return HttpResponseRedirect('/contact/thanks/')
    else:
        form = ContactForm()
    return render(request, 'contact_form.html', {'form': form})
```



Validación propia

Podemos programar validación extra asociada a cada Field del formulario escribiendo un método `clean_<fieldname>`:

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    email = forms.EmailField(required=False)
    message = forms.CharField(widget=forms.Textarea)

    def clean_message(self):
        message = self.cleaned_data['message']
        num_words = len(message.split())
        if num_words < 4:
            raise forms.ValidationError("Not enough words!")
        return message
```

Maquetación propia

Podemos personalizar la maquetación tanto como queramos, prescindiendo de las ayudas de `form.as_table`:

```
...  
<form action="" method="post">  
  <div class="field">  
    {{ form.subject.errors }}  
    <label for="id_subject">Subject:</label>  
    {{ form.subject }}  
  </div>  
  <div class="field">  
    {{ form.email.errors }}  
    <label for="id_email">E-mail:</label>  
    {{ form.email }}  
  </div>  
  ...  
  <input type="submit" value="Submit">  
</form>  
...
```

Para los diseñadores:

```
<style type="text/css">  
  ul.errorlist {  
    ...  
  }  
  .errorlist li {  
    ...  
  }  
</style>
```

Views avanzadas

El primer parámetro de **patterns()** sirve de algo

```
from django.conf.urls.defaults import *

urlpatterns = patterns('',
    url(r'^hello/$', 'mysite.views.hello'),
    url(r'^time/$', 'mysite.views.current_datetime'),
    url(r'^time/plus/(d{1,2})/$', 'mysite.views.hours_ahead'),
)
```

Views avanzadas

En ficheros **urls.py** que gestionen **varias apps...**

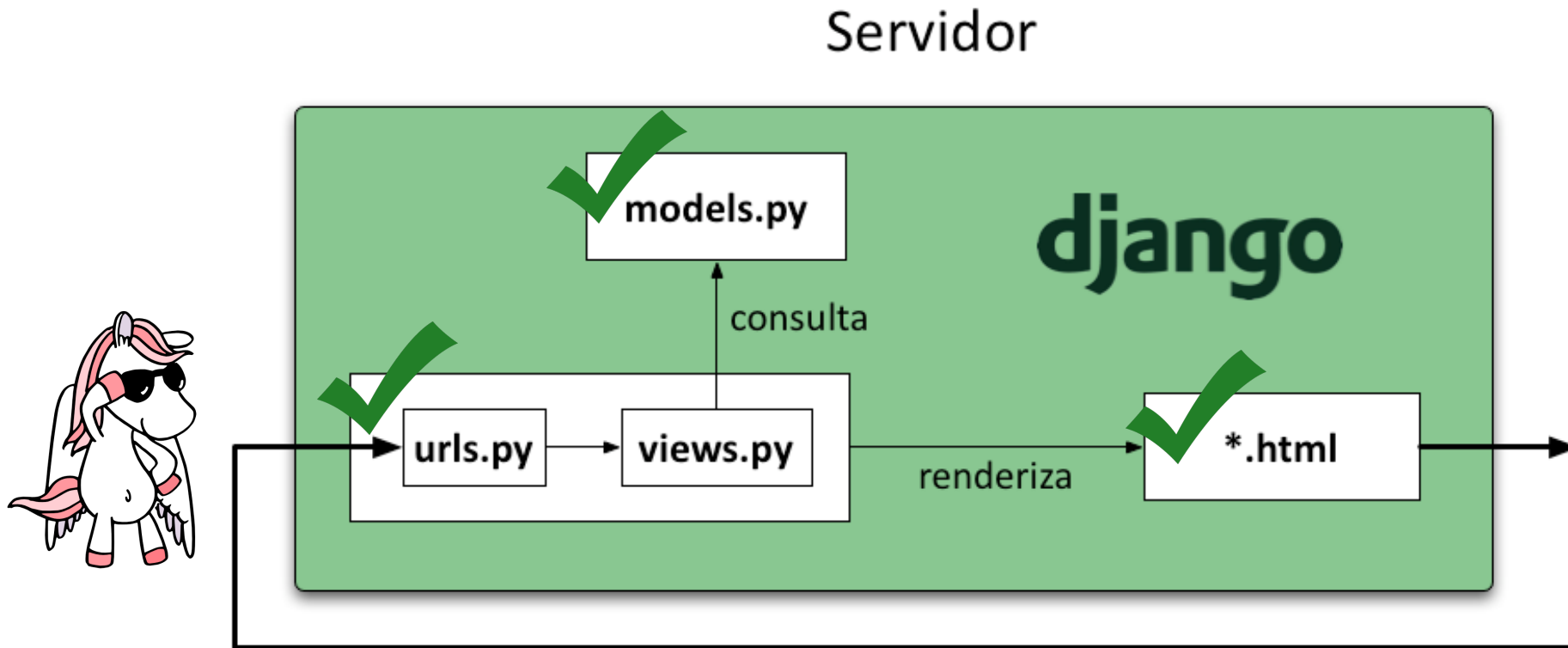
```
from django.conf.urls.defaults import *

urlpatterns = patterns('mysite.views',
    url(r'^hello/$', 'hello'),
    url(r'^time/$', 'current_datetime'),
    url(r'^time/plus/(\d{1,2})/$', 'hours_ahead'),
)

urlpatterns += patterns('weblog.views',
    url(r'^tag/(\w+)/$', 'tag'),
)
```

Views avanzadas

¿Dónde somos capaces de **reutilizar** código entre apps?



Views avanzadas

Generic Views

Vistas con funcionalidad genérica parametrizable mediante un diccionario de **Extra Options**



Las tuplas de **patterns()** pueden tener 3 elementos

```
from django.conf.urls.defaults import *
from mysite import views

urlpatterns = patterns('',
    url(r'^foo/$', views.foo_bar_view, {'template_name': 'template1.html'}),
    url(r'^bar/$', views.foo_bar_view, {'template_name': 'template2.html'}),
)
```

Views avanzadas

```
from django.views.generic.simple import direct_to_template
```

Renderiza directamente la template indicada



```
from django.conf.urls.defaults import *  
from django.views.generic.simple import direct_to_template  
  
urlpatterns = patterns('',  
    url(r'^about/$', direct_to_template, {'template': 'about.html'})  
)
```

Es la Generic View más simple y más utilizada

dajaxproject.com

Fork me on GitHub



- Set de librerías AJAX para django.
- django-dajaxice
 - Core de comunicación.
 - Enviar información asincrónicamente.
- django-dajax
 - manipular el DOM usando Python.

Objetivos dajaxice



Communication uniforme entre el cliente y el servidor .

- Agnostico a cualquier Framework JS.
- Prototype, jQuery, etc...
- Lógica de presentación fuera de las vistas.
- Crossbrowsing.

Ejemplo

```
from django.utils import simplejson
from dajaxice.decorators import dajaxice_register

@dajaxice_register
def my_example(request):
    return simplejson.dumps({'message': 'Hello World'})
```

↑
Si es una funcion
que retorna json.

Ejemplo

Fork me on GitHub

function name

```
function on_whatever() {  
    Dajaxice.example.my_example(my_js_callback);  
}
```

app

callback

Ejemplo

callback



```
function my_js_callback(data){  
    alert(data.message);  
}
```

your stuff



Objetivos dajax



- Manipular el DOM usando Python.
- Sin necesidad de conocer JS en profundidad.
- Soporta Frameworks como



Ejemplo

```
from dajax.core.Dajax import Dajax

def assign_test(request):
    dajax = Dajax()
    dajax.assign('#list li', 'innerHTML', 'Hello!')
    dajax.add_css_class('#list li', 'new')
    ...
    return dajax.json()
```

tus acciones



Ejemplo

```
function on_whatever() {  
    Dajaxice.app.assign_test(Dajax.process);  
}
```

Dajax
callback

