

Introducción al desarrollo de aplicaciones web con Django



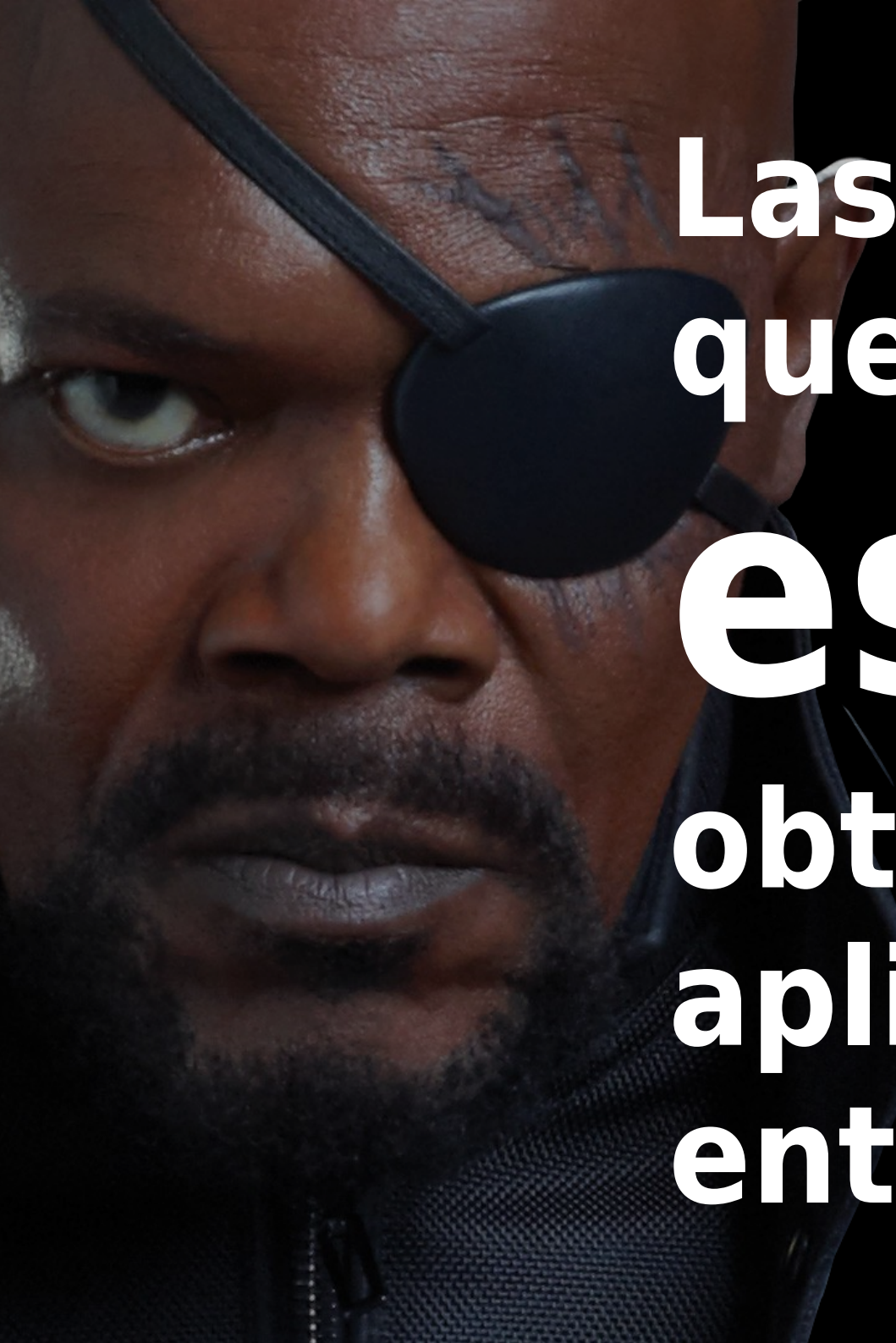
SHIELD™

Su misión, si decide aceptarla

Eres un agente recién incorporado a **S.H.I.E.L.D.** la agencia de inteligencia y antiterrorismo del Universo Marvel.

El mismísimo **Nick Furia**, mirándote fijamente con su ojo derecho, te ha encargado que desarrolles una aplicación para realizar el seguimiento de las actividades de los llamados "superheroes"





**Las respuestas
que Nick Furia
espera
obtener de tu
aplicación son,
entre otras:**

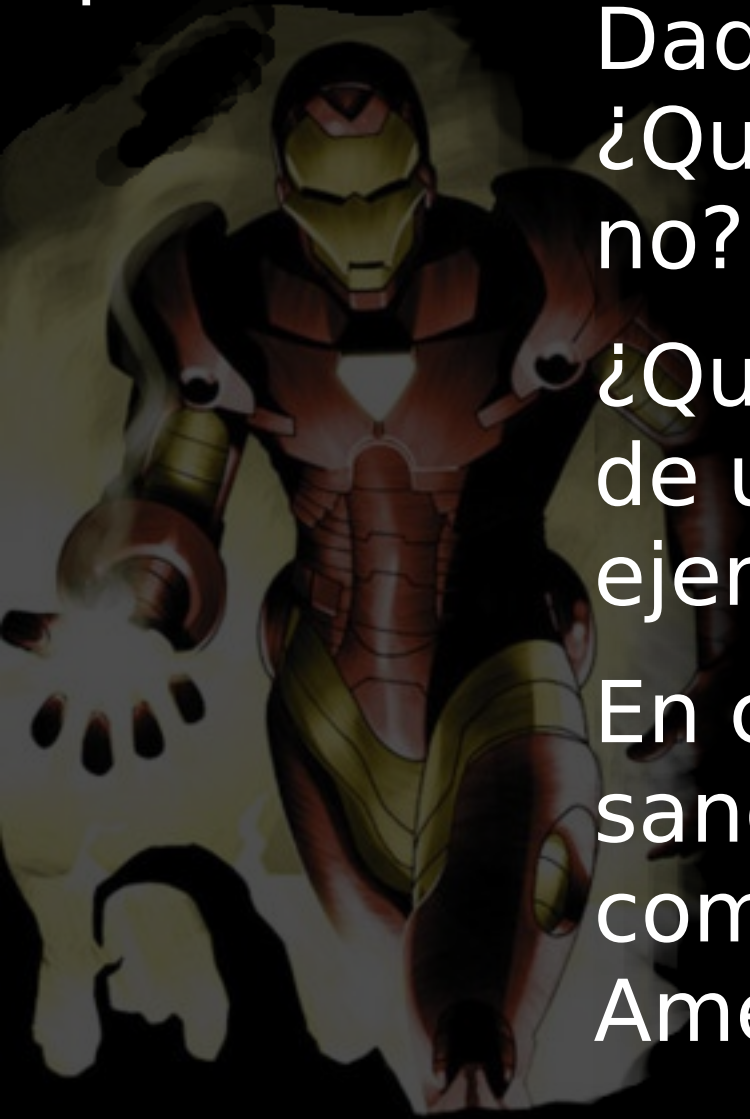
¿Qué poderes tiene un determinado heroe?

¿Podemos clasificarlos en distintos niveles de potencia?

Dado un poder determinado,
¿Qué heroes lo tienen? ¿Cuáles
no? ¿Cuáles pueden volar?

¿Qué superheroes forman parte
de un determinado grupo, por
ejemplo, los Cuatro Fantásticos?

En caso de apuro ¿El tipo
sanguineo de Spiderman es
compatible con el del Capitán
América?



¿Dónde fue visto Hulk por última vez?

¿Dónde suele verse con más frecuencia a Spiderman? ¿Podemos mostrar estos avistamientos en un mapa de GoogleMaps?

Y sobre todo ¿**Quién puede más, La Cosa o La Masa?**



Desarrollo web (en el siglo XX)

Muy sencillo, solo tres tecnologías nuevas:

- HTML (Lenguaje de marcado)
- HTTP (Protocolo de comunicaciones)
- MIME (Extensión Multimedia)

Desarrollo web (en el siglo XXI)

Ya no tan sencillo. Además de lo anterior:

- CSS
- JavaScript
- Contenidos dinámicas
- Lenguajes de programación
- Fragmentación de navegadores
- Versiones por dispositivo (p.e. Móviles)
- Ajax
- Seguridad
- Caches en memoria para mejorar rendimiento
- Base de datos
- Distribución de carga
- ...

Django es un framework de desarrollo web

- Software libre
- Escrito 100% en Python, que también es SL
- Pensado para aplicaciones web de tamaño medio
- Ampliamente usado
- Sigue el Patrón MVC (Modelo/Vista/Controlador)
- Muchos módulos de terceras personas:
 - south
 - django-extensions
 - sentry
 - django-tagging
 - ...

Python es un lenguaje muy sencillo de leer (Se diseñó así)

- Orientado a objetos
- Estructuras de datos muy poderosas: tuplas, listas, mapas hash o diccionarios, arrays y conjuntos.
- Compilado a código intermedio
- ¡Divertido!
- Simple pero poderoso: Metaclases, herencia multiple, introspección...

Lo que puede sorprender a un novato en Python

- La indentación marca realmente el flujo del programa
- No hay métodos ni propiedades privadas
- Las funciones pueden devolver más de un resultado
- Las estructuras de datos y las estructuras de control están integradas
- Las funciones son objetos.

Las funciones pueden devolver más de un resultado

```
def division_yresto(dividendo, divisor):  
    return dividendo // divisor, dividendo % divisor  
  
cociente, resto = division_yresto(47, 9)  
print 'cociente:', cociente  
print 'resto:', resto
```

Las estructuras de datos y las de control están integradas

```
datos = {'uno':1, 'dos':2, 'tres':3, 'cuatro'}
```

```
for k in datos:
```

```
    print 'El %d se escribe %s' % (datos[k],k)
```

```
numeros = (4, 8, 15, 16, 23, 42)
```

```
for n in numeros:
```

```
    print n, n**n
```

Las funciones son objetos.

- Se pueden, por ejemplo, pasar como parámetros a otras funciones, guardar en variables, asignarles nuevos nombres...

```
def pordos(x):  
    return x*2  
  
def aplica(func, datos):  
    result = []  
    for v in datos:  
        result.append(func(v))  
    return result  
  
print aplica(pordos, [4,8,15,16,23,42])
```

¿Pero qué es un framework?

- Un *framework* es como una librería de código
- Ofrece soluciones para muchos problemas ya conocidos, y posibilidad de crearte tus propias soluciones en base a componentes más pequeños
- La diferencia con una librería es: Tú no te limitas a llamar al código del framework, el código del framework te llama a tí.

Al grano: Hola, Mundo

- Vamos a hacer la aplicación más pequeña posible en Django: Solo una página con el mensaje “Hola, Mundo”.
- Abran una terminal, shell, o línea de comando

```
django-admin.py startproject hola
```


¡OJO!

- No usar para el proyecto nombres de módulos python. Especialmente, no se debe usar “**django**” (entraría en conflicto con el mismo Django) ni “**test**” (entraría en conflicto con el sistema de tests unitarios)



Contenido de “hola, mundo”

- Si miramos el directorio recién creado, veremos una serie de ficheros:
 - `__init__.py`
 - `hola`
 - `manage.py`
 - `settings.py`
 - `urls.py`
 - `wsgi.py`

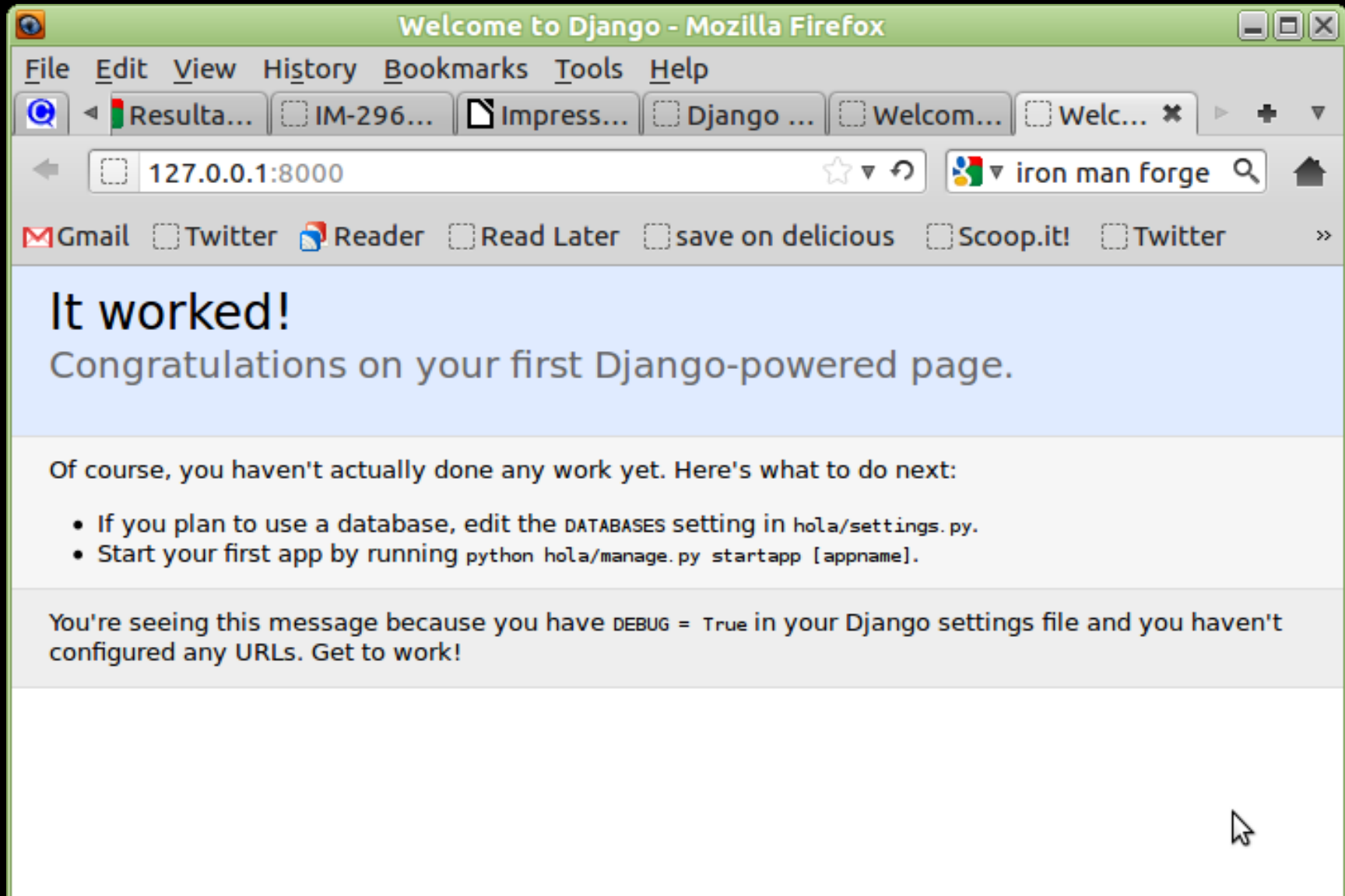
Pero... ¿funciona?

```
manage.py runserver
```

Debería aparecer algo así:

```
Django version 1.3.1, using settings 'hola.settings'  
Development server is running at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.  
[19/Jul/2012 12:42:44] "GET / HTTP/1.1" 200 2047
```

Funciona



¿Para qué son esos ficheros?

- El directorio **hola** externo es solo un contenedor para el proyecto. A django no le interesa especialmente, se puede cambiar de nombre, por ejemplo.
- **manage.py**: Es una utilidad de línea de comando que nos permite interactuar de diferentes maneras con el proyecto.
- El directorio **hola** interno es donde irá nuestro código, ya que es el que importará Django.

¿Para qué son esos ficheros? (2)

- **hola/__init__.py**: Un fichero que indica que este directorio debe ser considerado un módulo (es decir, importable)
- **hola/settings.py**: El fichero de configuración para este proyecto
- **hola/urls.py**: El mapeo entre urls y nuestro código
- **hola/wsgi.py**: para poner nuestra aplicación en un servidor web que entienda el protocolo wsgi

Siguiente paso

- Bien, ya tenemos un servidor de prueba funcionando.
- Vamos a modificar la aplicación para poder presentar la página que diga “Hola, mundo”.
- Los pasos son:
 - Pensaremos la url que queremos
 - Mapearemos la url a una función (vista)
 - Escribimos la vista que genera la página

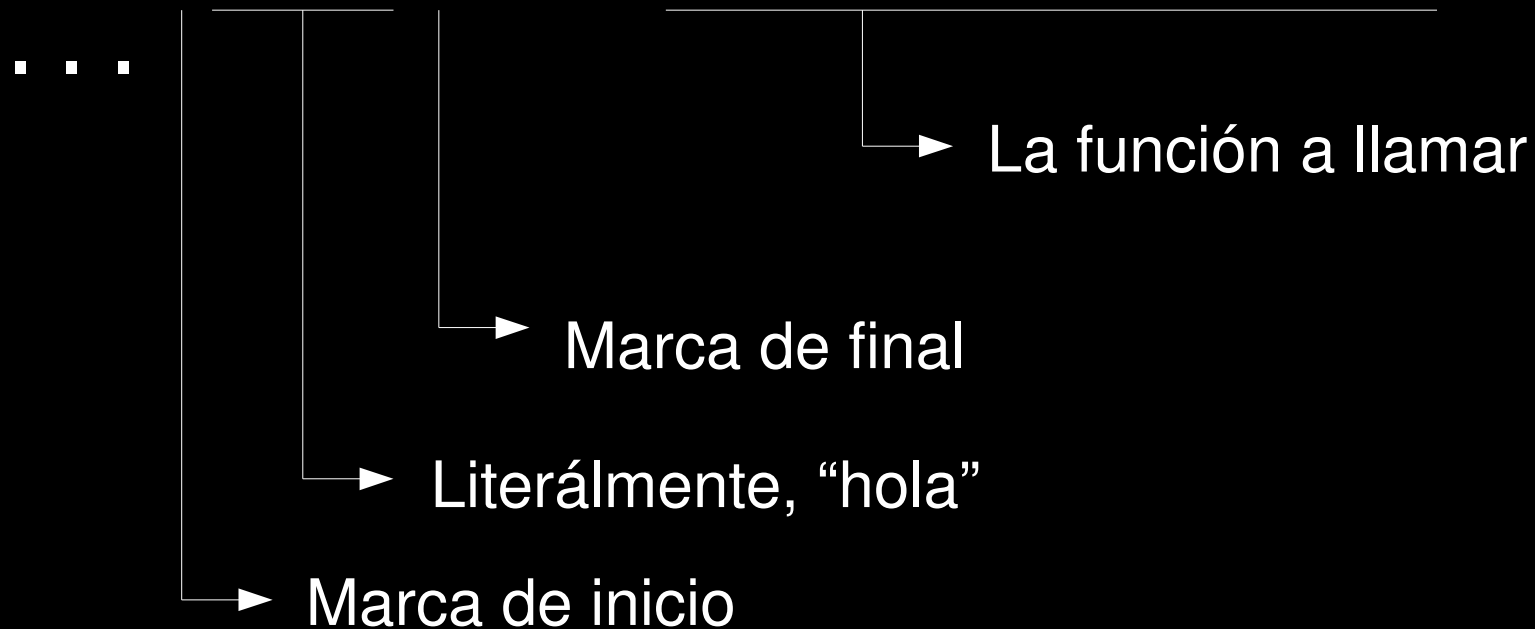
El fichero “urls.py”

- Vamos a mapear la url `/hola`.
- Dentro del fichero `hola/hola/urls.py` se define la variable llamada `urlpatterns`.
- Consiste en una serie de tuplas con el siguiente formato:

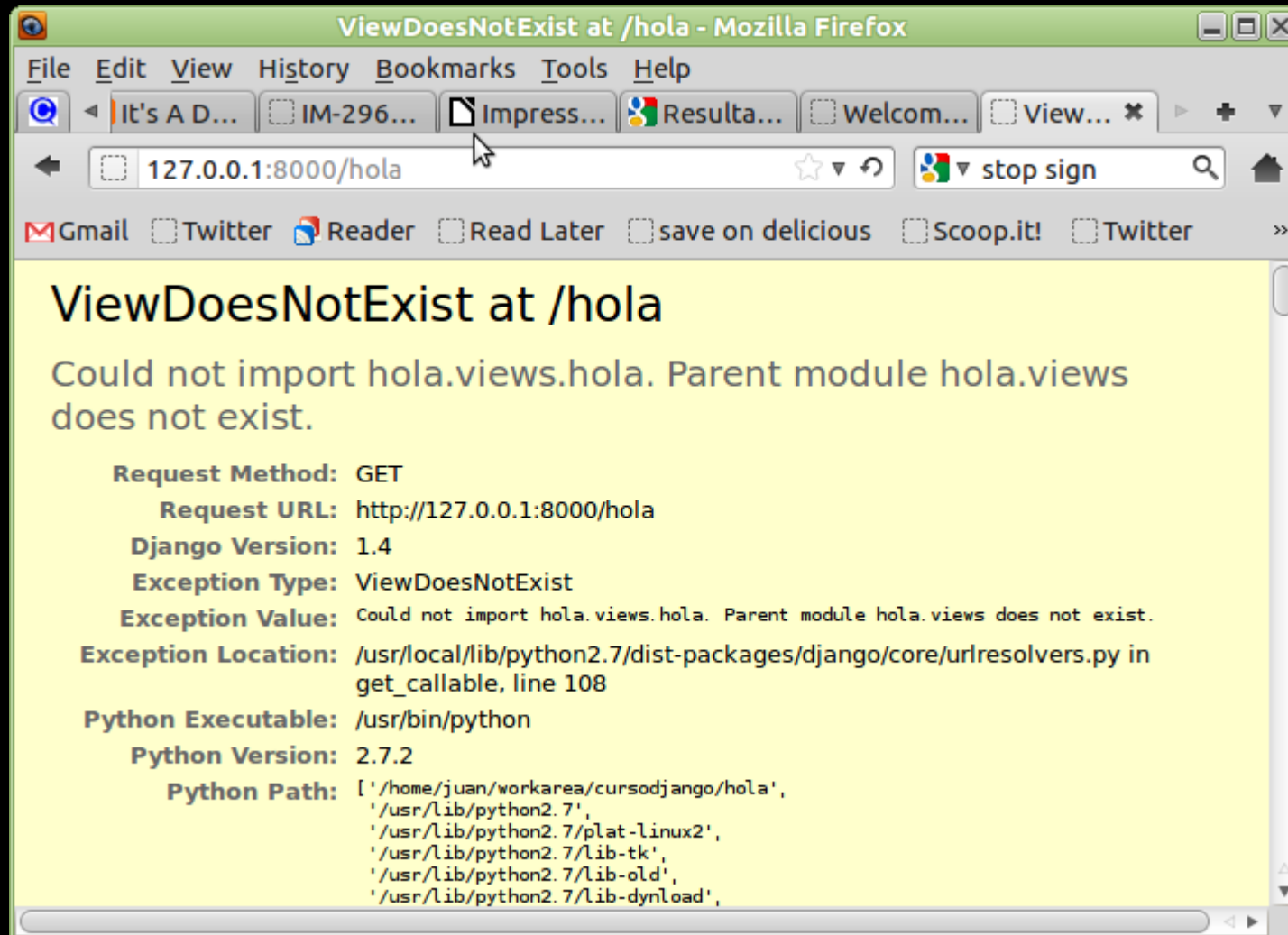
```
(expresion regular, function, <datos extra>)
```

Añadimos la url /hola

```
...  
urlpatterns = patterns('',  
    (u'^hola$', 'hola.views.hola'),  
    ...  
)
```



Si intentamos acceder, nos dará un error

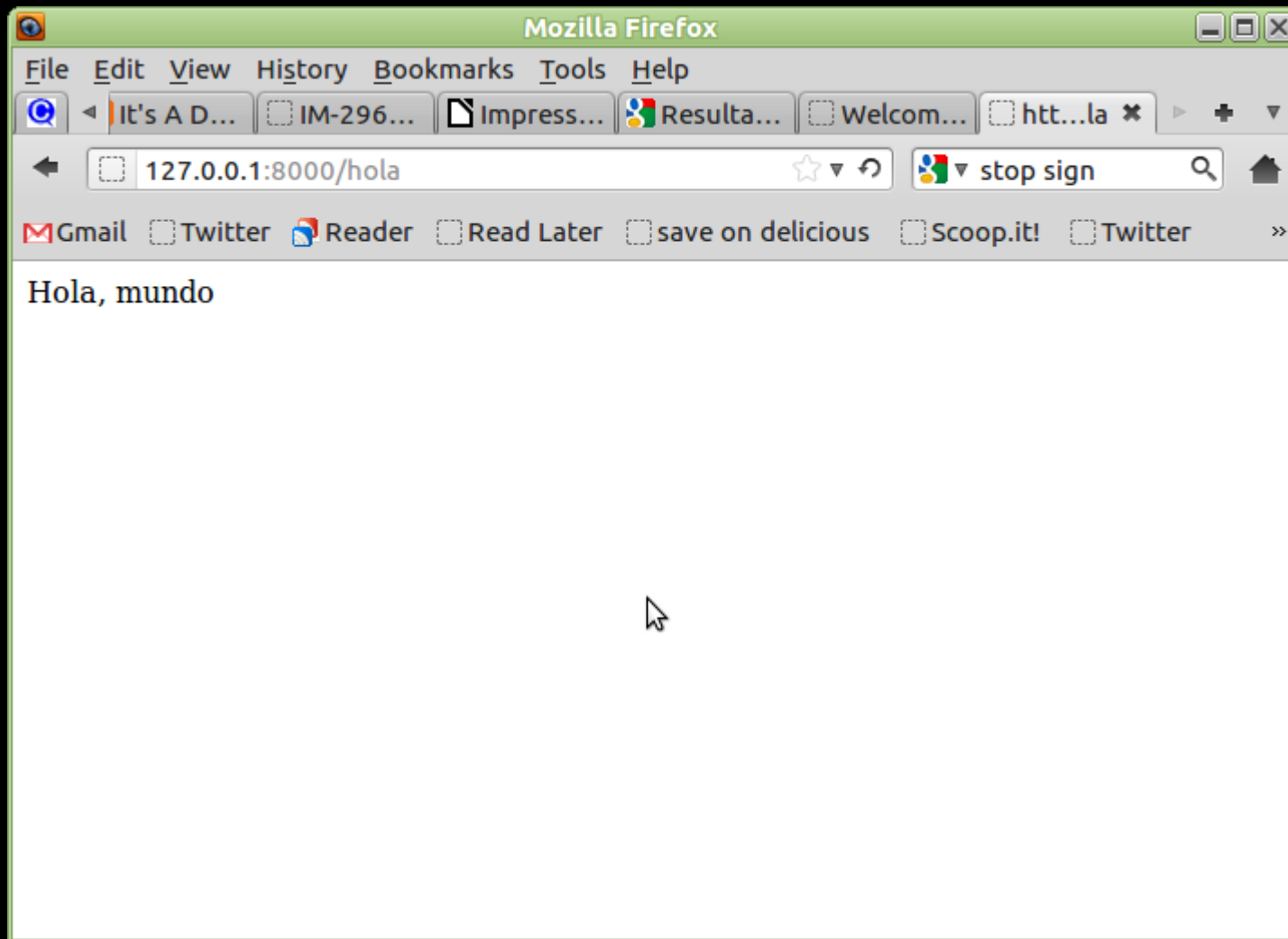


Porque la vista no existe: escribámosla

En el fichero `hola/hola/views.py`

```
from django.http import HttpResponse  
  
def hola(request):  
    return HttpResponse("Hola, mundo")
```

¡Funciona!



Bases de datos

- Para hacer algo más impresionante, necesitamos algún tipo de base de datos
- Django trabaja con varias bases de datos relaciones: postgresql, MySQL, Oracle y sqlite (Hay extensiones para otros SGBD)
- Mi favorita es postgresql, pero vamos a usar sqlite, porque es mucho más sencilla de configurar



Configurar bases de datos

- Creamos un nuevo proyecto con:

`django-admin startproject shield`

- Modificamos la configuración en settings.py

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': 'shield_db',  
        'USER': '',  
        'PASSWORD': '',  
        'HOST': '',  
        'PORT': '',  
    }  
}
```


Comprobar los cambios

- Con `manage.py validate` podemos comprobar si hay algún error en la configuración. Será tu nuevo mejor amigo.
- Con `manage.py syncdb` se pone al día la base de datos. Eso incluye crearla (en sqlite3) y alimentarla con los datos iniciales

```
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
```

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 'juan'): admin

E-mail address: euribates@gmail.com

Password:

Password (again):

Superuser created successfully.

Installing custom SQL ...

Installing indexes ...

Installed 0 object(s) from 0 fixture(s)

Definición de los modelos

- Vamos a definir los modelos con los que vamos a trabajar
- Para ello, vamos a realizar un diagrama llamado Diagrama de Entidad/Relación (o E/R para abreviar).
- Mostraremos las Entidades que vemos en la definición del problema, y las relaciones que hay entre ellas

Propuesta de modelos

- Superheroes
- Poderes
- Equipos
- Avistamientos

Las relaciones

- Define como estan vinculadas (o no) entre si las entidades
- Aquí es importante determinar la cardinalidad de la relación.
- Según la cardinalidad, sólo hay tres tipos posibles de relaciones:
 - 1 a 1
 - 1 a N (o N a 1)
 - N a N

Ejemplo de relaciones

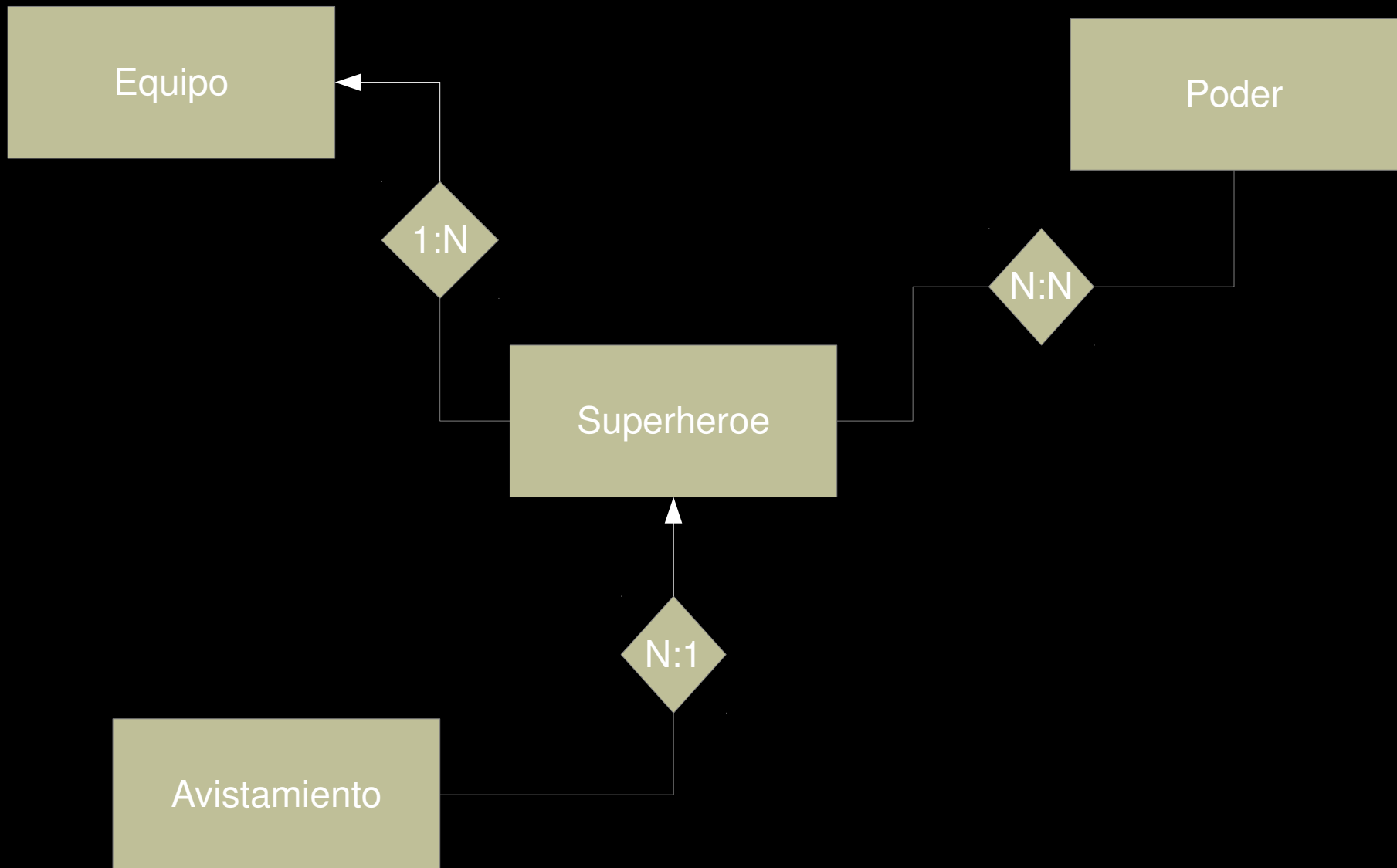
Por ejemplo, si el dominio del problema fuera el cine, con actores, películas y productoras:

- La relación entre película y productora es 1 a N:
Una productora produce varias películas, pero una película sólo tiene una productora
- La relación entre actores y películas es N a N.
Un actor puede aparecer en varias películas, y en una película pueden actuar varios actores.

Propuesta de relaciones

- Heroes \leftarrow N a N \rightarrow Poderes
 - Un heroe tiene varios poderes
 - Un poder puede ser compartido por varios heroes
- Heroes – N a 1 \rightarrow Equipos
 - Un superheroe sólo pertenece a un equipo
 - Un equipo está compuesto por varios heroes
- Heroes \leftarrow 1 a N – Avistamientos
 - Un heroe puede ser visto en varios sitios (en momentos diferentes)

Esquema E/R



Atributos de los modelos

- Para cada modelo, definimos una serie de atributos, que son características que nos interesa modelar de cada entidad.
- Los tipos de atributos que podemos modelar en Django son valores numéricos, textos, fechas, direcciones web y muchos más (Además, puedes definir tus propios tipos de atributos)

Propuesta de atributos(1)

- Para los superheroes:
 - Nombre (texto)
 - Identidad (texto)
 - Fecha de nacimiento (fecha)
 - Nivel (Numero del 1 al 100)
 - grupo_sanguineo (O, A, B, AB)
 - Foto (tipo imagen)
 - Correo electrónico

Propuesta de atributos(2)

- Para los equipos:
 - Nombre (texto)
 - Fecha de creación (fecha)
 - Cuartel General (texto)

Propuesta de atributos(3)

- Para los avistamientos:
 - fecha (fecha)
 - Latitud (Número en coma flotante)
 - Longitud (Número en coma flotante)
 - Observación (texto)

Propuesta de atributos(4)

- Para los poderes:
 - Nombre del poder (texto)
 - Descripción (texto)

A modelar en Django

- Crearemos un nuevo fichero `models.py`
- Definimos una clase por cada modelo, que debe derivar de:
`django.db.models.Model`
- Cada clase contiene los atributos propios del modelo.
- Podemos definir métodos también, si queremos.

shield/models.py (1)

```
from django.db import models

class Equipo(models.Model):
    nombre_equipo = models.CharField(max_length=60)
    f_creacion = models.DateField()
    Direccion = models.TextField()

class Poder(models.Model):
    nombre_poder = models.CharField(max_length=32)
    descripcion = models.CharField(max_length=500)
```


shield/models.py (2)

```
GRUPO_SANGUINEO = ( ('0', '0 (Universal)'),  
                    ('A', 'Grupo A'),  
                    ('B', 'Grupo B'),  
                    ('AB', 'Grupo AB') )  
  
class Heroe(models.Model):  
    nombre_heroe = models.CharField(max_length=120)  
    identidad = models.CharField(max_length=170)  
    f_nacimiento = models.DateField()  
    nivel = models.IntegerField()  
    grupo_sanguineo = models.CharField(  
        max_length=2,  
        choices=GRUPO_SANGUINEO,  
    )  
    foto = models.FileField(upload_to='fotos')  
    correo = models.EmailField()  
    equipo = models.ForeignKey(Equipo, blank=True)  
    poderes = models.ManyToManyField(Poder)
```

shield/models.py (3)

```
class Avistamiento(models.Model):  
    f_avistamiento = models.DateTimeField()  
    latitud = models.FloatField()  
    longitud = models.FloatField()  
    hero = models.ForeignKey(Heroe)
```

Añadir nuestra aplicación en el fichero settings

- Para que django reconozca los modelos que acabamos de crear, hay que indicarle que existe una nueva aplicación, shield.
- En `settings.py` hay una variable que determina que aplicaciones están activas, `INSTALLED_APPS`.
- Tenemos que añadirla nosotros, `manage.py startproject` no lo hace por nosotros.

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

Comprobaciones

- Con `manage.py validate` podemos detectar problemas
- Si todo ha ido bien, con `manage.py syncdb` veremos que se crean **cinco (5)** nuevas tablas
- Con `manage.py sql shield` podemos ver el código SQL generado para crear las tablas.
- Con `manage.py sql dbshell` podemos interactuar con el cliente de consola de la BD (si está instalado).

Activamos el admin



El admin solo es otra app más

- Pero es tremendamente útil
- Proporciona un entorno **CRUD** (*Create, Read, Update, Delete*) completo de nuestros modelos
- Configurable (hasta cierto punto)
- Si lo configurable no alcanza, puedes ir reemplazándolo poco a poco por tu propio código.

No viene activo por defecto

- Descomentar `django.contrib.admin` en la variable `INSTALLED_APPS`, en el fichero `settings.py`.
- Ejecutar `python manage.py syncdb`. Se crearán nuevas tablas, propias de la app `admin`.
- Editar `shield/urls.py` y descomentar las líneas que hacen referencia a `admin`. Son en total tres líneas.


```
from django.conf.urls import patterns, include, url

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

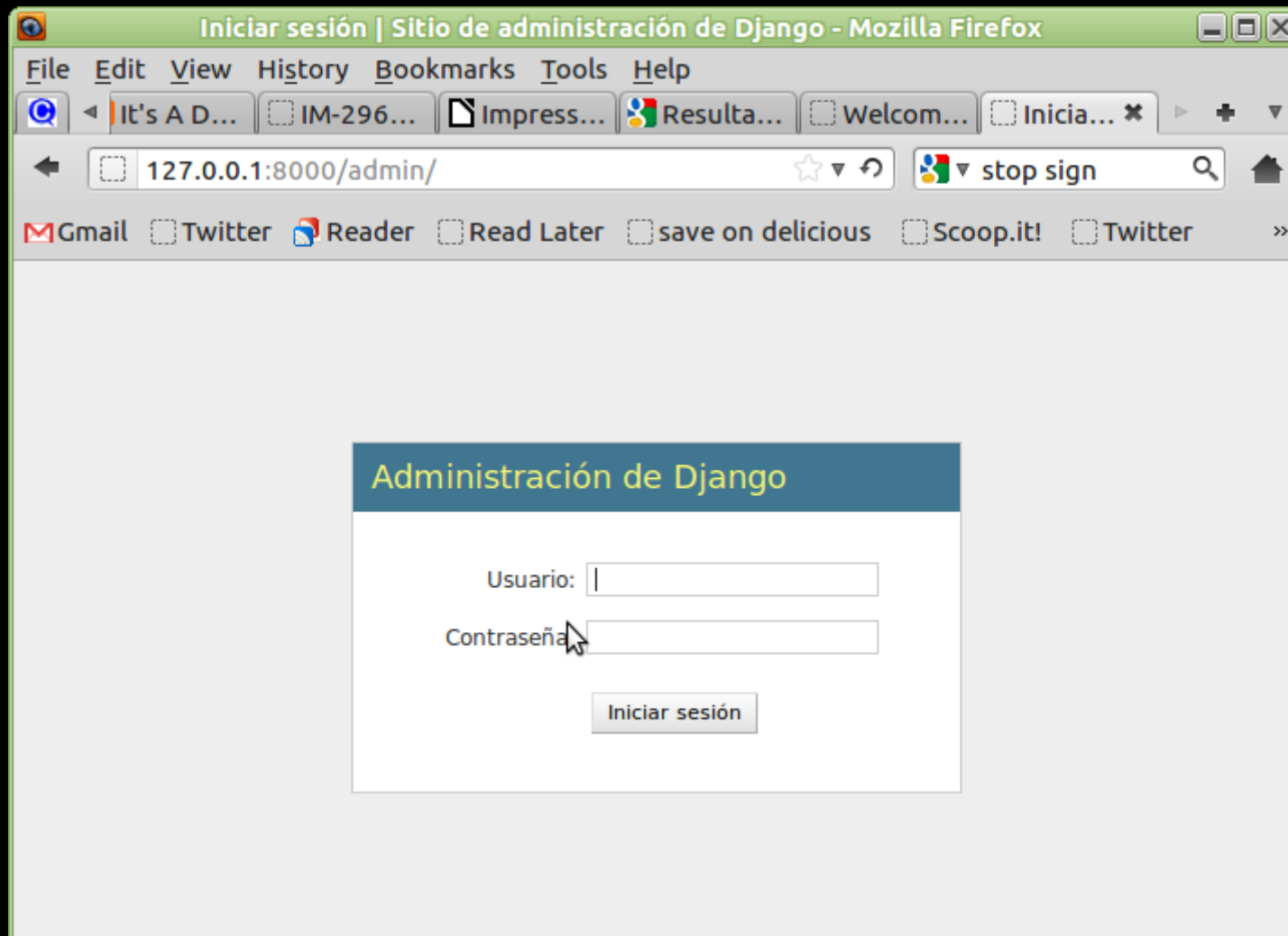
urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'shield.views.home', name='home'),
    # url(r'^shield/', include('shield.foo.urls')),

    # Uncomment the admin/doc line below to enable ...
    # url(r'^admin/doc/',
    #     include('django.contrib.admindocs.urls')),

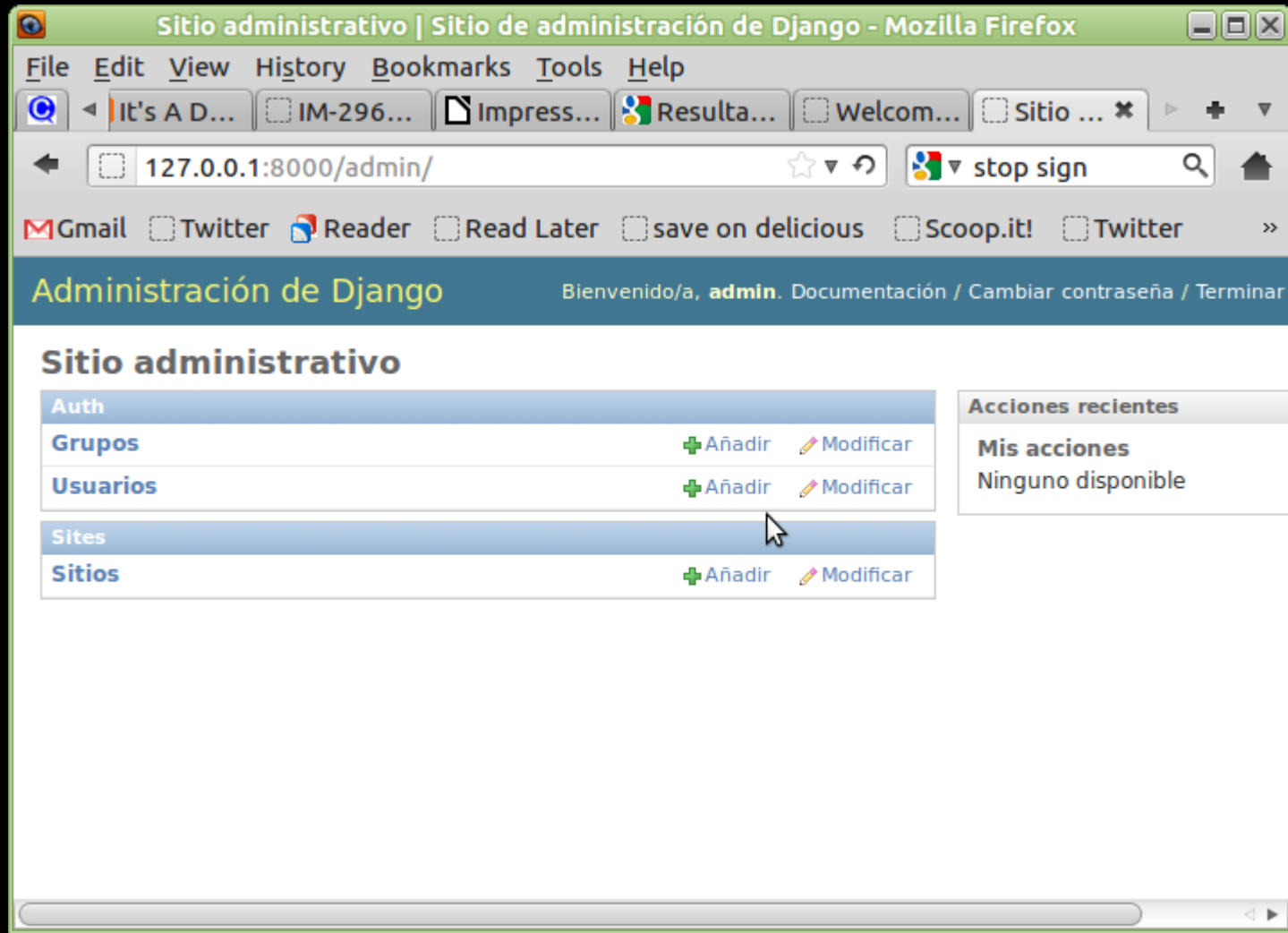
    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
)
```

Acceder al admin

`http://127.0.0.1:8000/admin/`



¿Dónde están nuestros modelos?



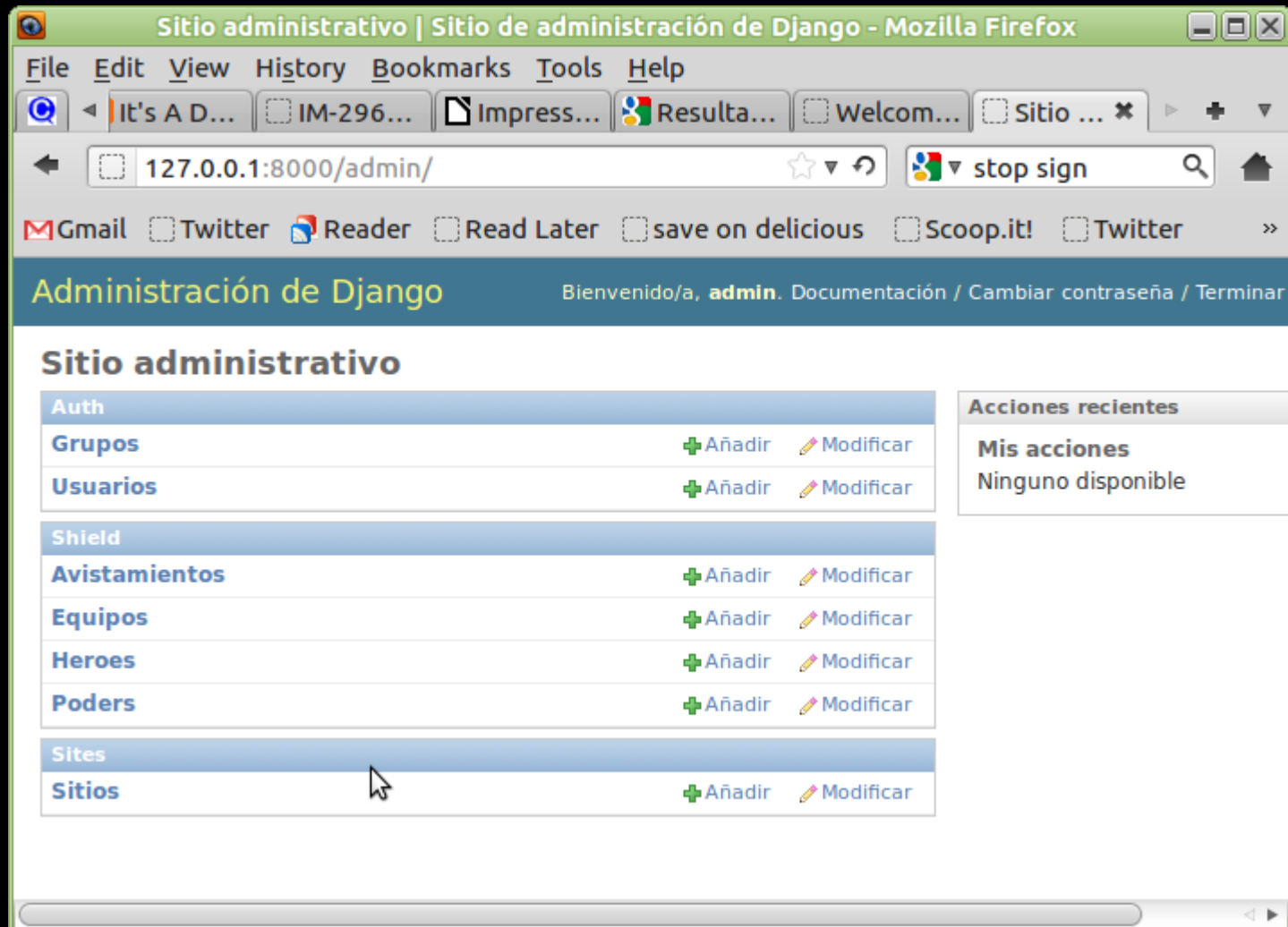
Falta solo un paso más

- Añadir un fichero `shield/admin.py`, para determinar que modelos, y de que manera, se administran con el admin.

```
from django.contrib import admin
from shield.models import Heroe, Poder
from shield.models import Equipo, Avistamiento

admin.site.register(Heroe)
admin.site.register(Poder)
admin.site.register(Equipo)
admin.site.register(Avistamiento)
```

!Funciona!



Añadimos unos cuantos heroes

Escoja heroe a modificar

Añadir heroe +

Acción: Ir seleccionados 0 de 11

<input type="checkbox"/>	Nombre heroe	Identidad	Nivel	Lista poderes	Equipo
<input type="checkbox"/>	Spiderman	Peter Parker	30	Superagilidad, Superfuerza, Sentido arácnido, Adhesividad corporal	Vengadores
<input type="checkbox"/>	Daredevil	Matthew (Matt) Michael Murdock	25	Superagilidad, Radar	Sin equipo
<input type="checkbox"/>	La Cosa	Benjamin (Ben) Grimm	85	Superfuerza, Superresistencia, Blindaje	Los Cuatro Fantásticos
<input type="checkbox"/>	Iron Man	Anthony (Tony) Stark	88	Volar, Superfuerza, Blindaje, Rayos repulsores, Contramedidas, Cibernética, Supersentidos, Extremis	Vengadores
<input type="checkbox"/>	Antorcha Humana	Jonathan (Johnny) Storm	45	Volar, Rayos de energía, Supernova	Los Cuatro Fantásticos
<input type="checkbox"/>	Capitán América	Steven (Steve) Rogers	25	Superagilidad, Superresistencia	Vengadores
<input type="checkbox"/>	Cíclope	Scott Summers	42	Rayos ópticos	Patrulla X
<input type="checkbox"/>	Mujer Invisible	Susan Storm	90	Invisibilidad, Campos de fuerza	Los Cuatro Fantásticos
<input type="checkbox"/>	Spiderwoman	Jessica Drew	32	Superagilidad, Superfuerza, Adhesividad corporal, Rayos de energía	Vengadores
<input type="checkbox"/>	La Masa (Hulk)	Robert Bruce Banner	99	Superfuerza, Superresistencia, Blindaje	Sin equipo
<input type="checkbox"/>	Viuda Negra	Natalia (Natasha) Alianovna Romanova	22	Espionaje	Vengadores

11 heroes

Afinando el admin

- Podemos añadir un método `__unicode__` a cada modelo, para representar en texto cada instancia. La función debe devolver, claro esta, una string en unicode.

```
class Heroe(models.Model):  
    ...  
    def __unicode__(self):  
        return u'%s (Nivel %d)' % (  
            self.nombre_heroe,  
            self.nivel,  
        )
```

Afinando el admin (2)

- Añadiendo subclases de la clase `ModelAdmin` podemos añadir columnas, campos de búsqueda y filtros, entre otras cosas:

```
class HeroeAdmin(admin.ModelAdmin):  
    search_fields = ('nombre_heroe', 'identidad',  
                    'poderes__nombre_poder')  
    list_display = ('nombre_heroe', 'identidad',  
                   'lista_poderes', 'equipo')  
    list_filter = ('equipo',)  
  
admin.site.register(Heroe, HeroeAdmin)
```

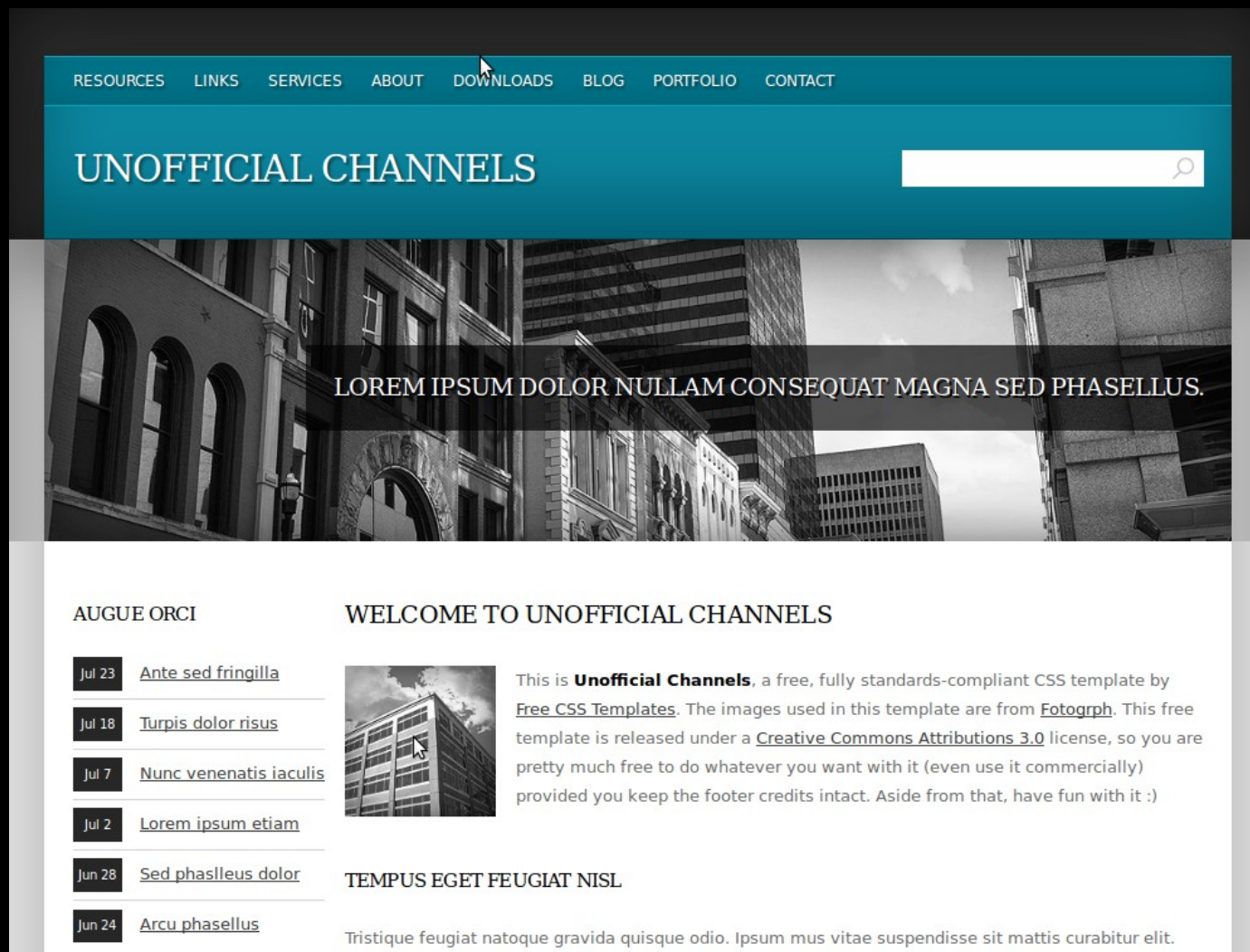

Nuestras páginas

El admin está muy bien, pero
¿Cómo hacemos páginas con
nuestro diseño? ¿Y que sean fáciles
de mantener?

Diseñamos una plantilla

- Como no tenemos tanto tiempo, vamos a adaptar una plantilla gratuita
- Hemos optado por **Unofficial Channels**,
Diseñada por Free CSS Templates
- <http://www.freecsstemplates.org>
- Licencia Creative Commons Attribution
3.0 License

Unofficial Channels



RESOURCES LINKS SERVICES ABOUT DOWNLOADS BLOG PORTFOLIO CONTACT

UNOFFICIAL CHANNELS

LOREM IPSUM DOLOR NULLAM CONSEQUAT MAGNA SED PHASELLUS.

AUGUE ORCI

Jul 23 [Ante sed fringilla](#)

Jul 18 [Turpis dolor risus](#)

Jul 7 [Nunc venenatis iaculis](#)

Jul 2 [Lorem ipsum etiam](#)

Jun 28 [Sed phaslleus dolor](#)

Jun 24 [Arcu phasellus](#)

WELCOME TO UNOFFICIAL CHANNELS



This is **Unofficial Channels**, a free, fully standards-compliant CSS template by [Free CSS Templates](#). The images used in this template are from [Fotogroph](#). This free template is released under a [Creative Commons Attributions 3.0](#) license, so you are pretty much free to do whatever you want with it (even use it commercially) provided you keep the footer credits intact. Aside from that, have fun with it :)

TEMPUS EGET FEUGIAT NISL

Tristique feugiat natoque gravida quisque odio. Ipsum mus vitae suspendisse sit mattis curabitur elit.

Vamos a añadir esta plantilla

- Dentro del directorio shield (interno) creamos un directorio `templates`, y dentro de este, otro `shield`.
- La cosa queda así:
`shield/templates/shield`
- Copiamos aquí el fichero `index.html`, pero con el nombre `base.html` (será nuestro html “básico”)

Hojas de estilo e imágenes

- Se suele recomendar separar los contenidos estáticos (Hojas de estilo, imágenes, etc) de los dinámicos
- Desde Django 1.3 hay una aplicación que se encarga de esto, `staticfiles`.
- Recopila, a partir de las aplicaciones instaladas, todos los recursos estáticos en una ubicación común, lista para ser servida en producción.

La aplicación staticfiles

- Crearemos, dentro del directorio `shield` (interno) una carpeta `static`, y dentro de esta, las carpetas `css` e `images`.
- Copiaremos el fichero `style.css` de la plantilla **Unofficial Channels** al directorio `shield/static/css`.
- Copiamos las imágenes (son tres, `banner.jpg`, `pic01.jpg` y `search.png`) al directorio `shield/static/images`.

La aplicación staticfiles (2)

- La aplicación buscará en cada aplicación instalada una carpeta `static`, y copiara su contenido a la carpeta `static` común.
- En desarrollo, si usamos el comando `runserver`, las diferentes carpetas `static` están siendo observadas; cualquier cambio en un `static` local se copia automáticamente al `static` común.

Vamos a modificarla un poco

- En resumen, las hojas de estilo y las imágenes, que estaban en un directorio relativo a la página, van a estar bajo un directorio absolute `/static`.
- La hoja de estilo estará en `/static/css/`
- Las imágenes en `/static/images/`
- Esto implica modificar el código de los ficheros `base.html` y `style.css` para reflejar estos cambios

Probemos con una vista simple

- Mapeamos la url `/test` a la vista `shield.views.test`.