

Introducción al desarrollo de aplicaciones web con Django



S.H.I.E.L.D.TM

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

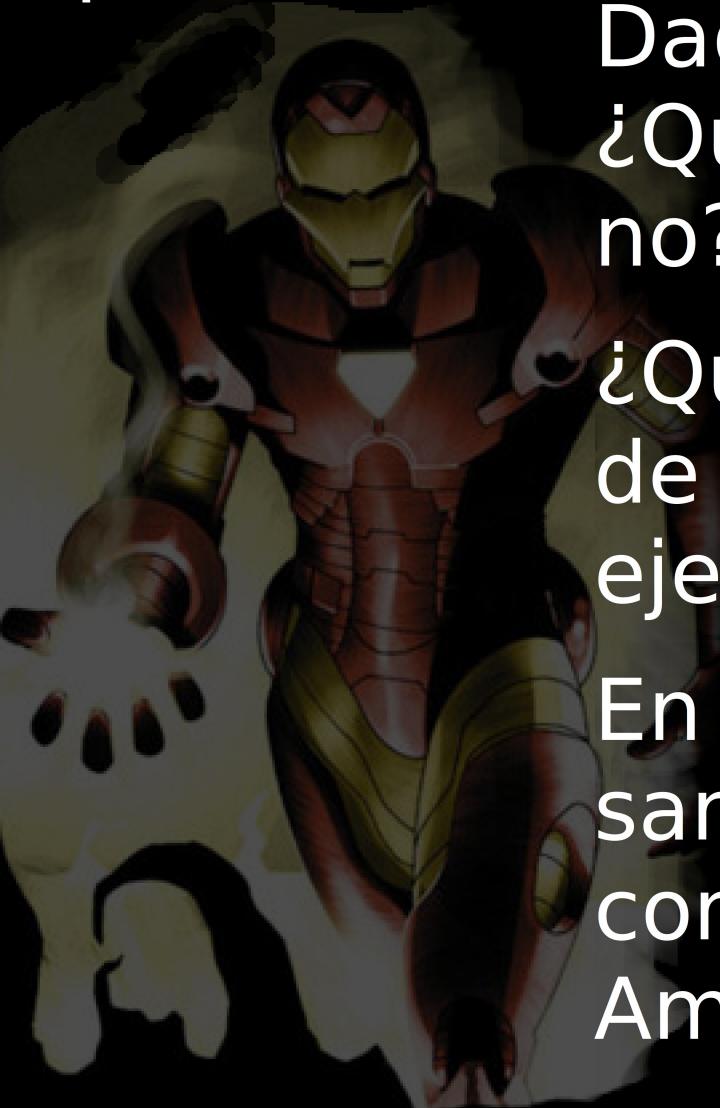


SHIELD™

A close-up, high-contrast portrait of a man's face, likely Nick Fury, wearing dark sunglasses. He has a serious, intense expression. The lighting is dramatic, with strong highlights and shadows. The background is dark and out of focus.

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

¿Qué poderes tiene un determinado héroe?
¿Podemos clasificarlos en distintos niveles de potencia?



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

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

En caso de apuro ¿El tipo sanguíneo 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 (Extensiones 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 diseño 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 múltiple, 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_y_resto(dividendo, divisor):  
    return dividendo // divisor, dividendo % divisor  
  
cociente, resto = division_y_resto(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: Tu 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 1.4.1, using settings 'hola.settings'  
Development server is running at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

¡Funciona!

The screenshot shows a Mozilla Firefox window with the title "Welcome to Django - Mozilla Firefox". The address bar displays "127.0.0.1:8000". The main content area shows the following text:

It worked!
Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Here's what to do next:

- If you plan to use a database, edit the `DATABASES` setting in `hola/settings.py`.
- Start your first app by running `python hola/manage.py startapp [appname]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

¿Para qué son estos 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:

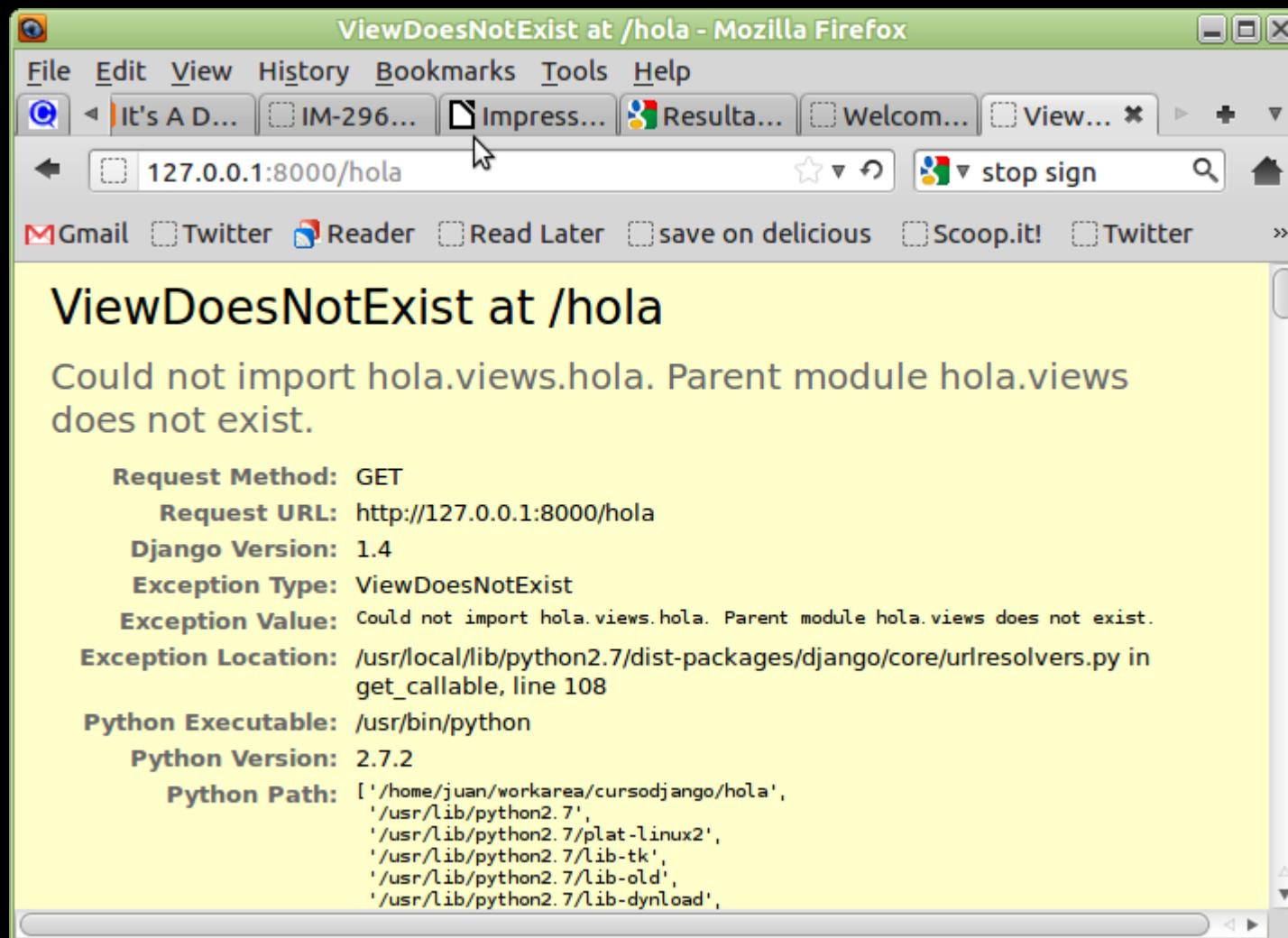
(expresion regular, función, <datos extra>)

Añadimos la URL /hola

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

- ...
 - Marca de final
 - Literalmente, “hola”
 - Marca de inicio
- La función a llamar

Si intentamos acceder, nos dará un error



No te enfades; esto es bueno



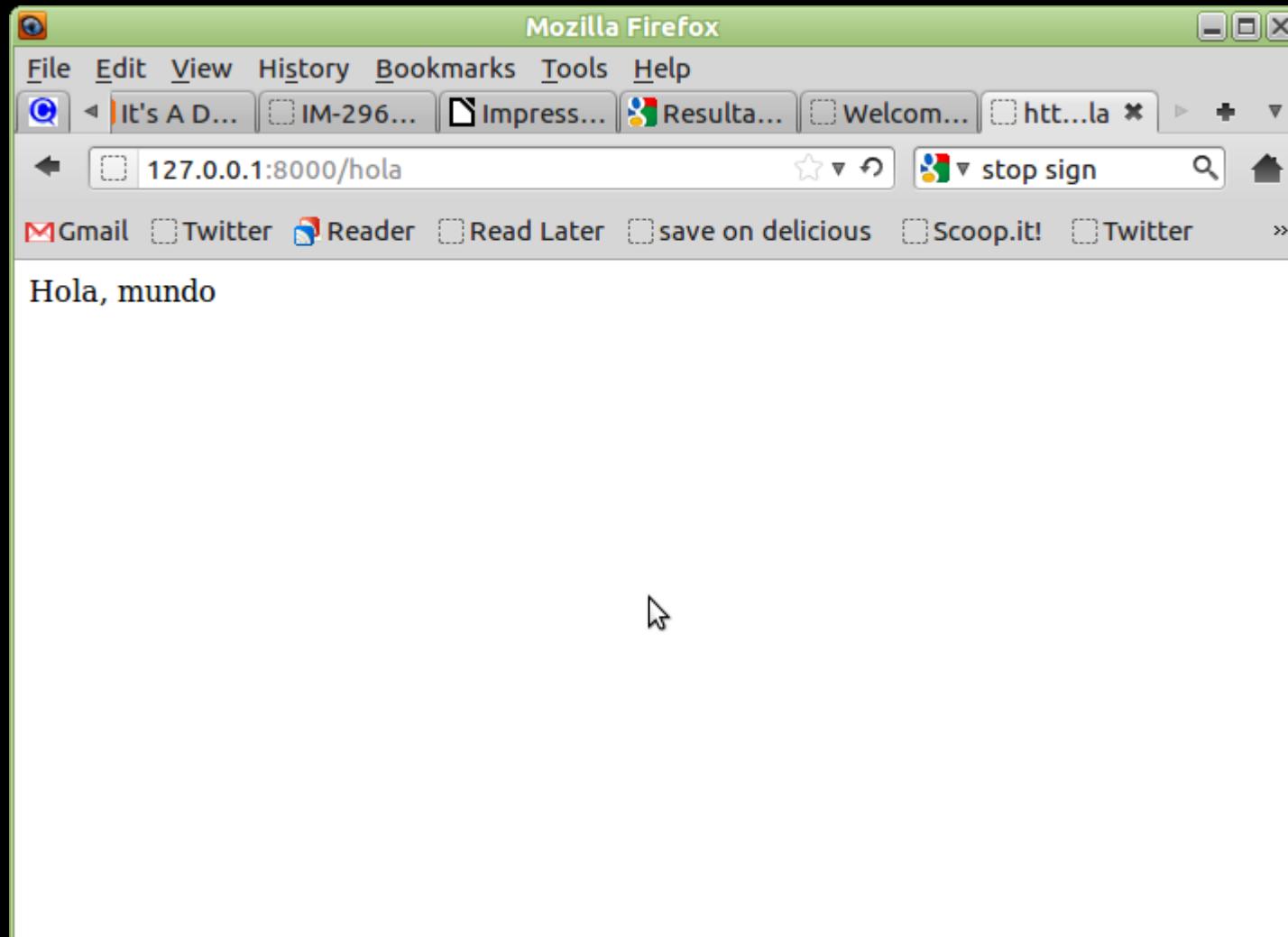
Porque la vista no existe: escribámosla

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

```
from django.http import HttpResponse

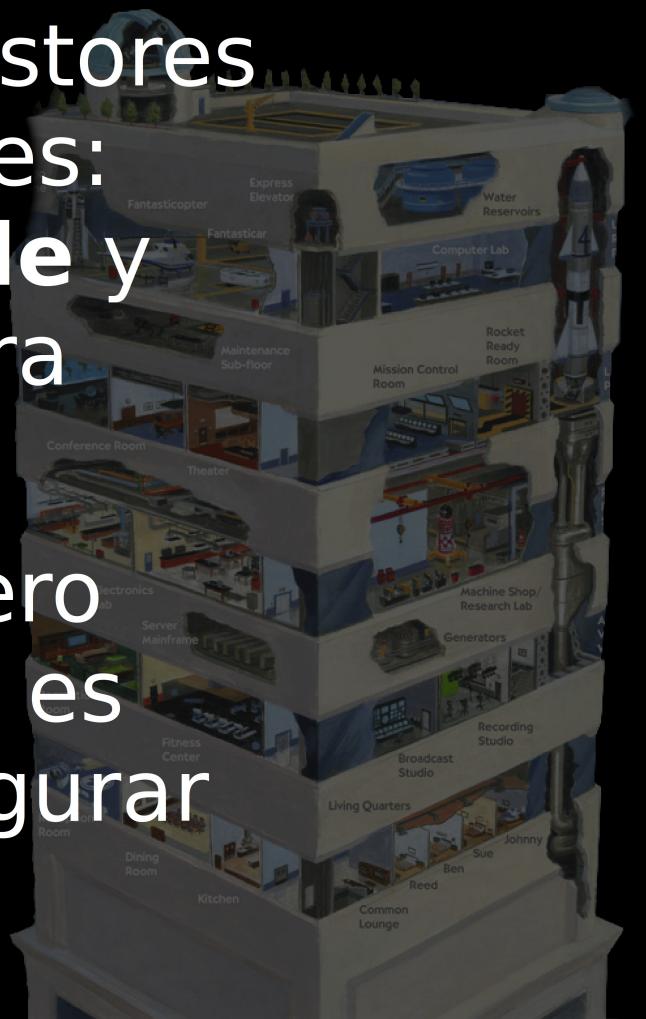
def hola(request):
    return HttpResponse("Hola, mundo")
```

iFunciona!



Bases de datos

- Para hacer algo más impresionante, necesitamos algún tipo de base de datos
- Django trabaja con varias gestores de bases de datos relacionales:
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_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 abbreviar)
- Mostraremos las **Entidades** que vemos en la definición del problema, y las **Relaciones** que hay entre ellas

Propuesta de modelos

- Superhéroes
- Poderes
- Equipos
- Avistamientos

Las relaciones

- Define como están vinculadas 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

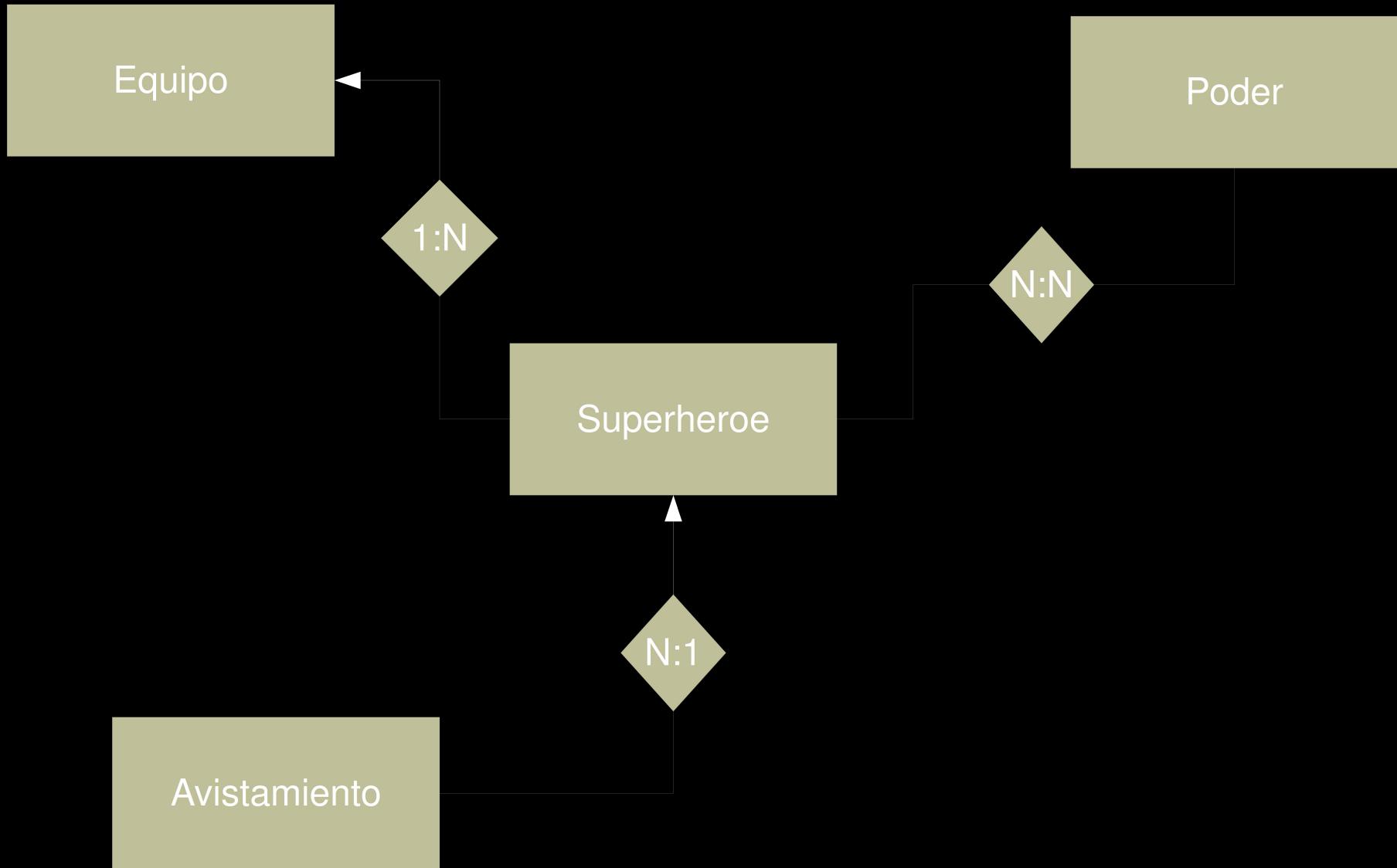
Por ejemplo, si el dominio del problema fuera el cine, con actores, películas y productoras:

- Entre productoras y películas tenemos una relación **1 a N**: Una productora produce varias películas, pero una película sólo tiene una productora
- La relación entre actores y películas, sin embargo, es **N a N**. Un actor puede trabajar en varias películas, y en una película normalmente hay varios actores

Propuesta de relaciones

- Héroes \leftarrow N a N \rightarrow Poderes
 - Un héroe tiene varios poderes
 - Un poder puede ser compartido por varios héroes
- Héroes - N a 1 \rightarrow Equipos
 - Un superhéroe sólo pertenece a un equipo
 - Un equipo está compuesto por varios héroes
- Héroes \leftarrow 1 a N - Avistamientos
 - Un héroe 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 superhéroes:
 - Nombre (texto)
 - Identidad (texto)
 - Fecha de nacimiento (fecha)
 - Nivel (Número del 1 al 100)
 - grupo_sanguíneo (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 = ( ('O', 'O (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()
    heroe = 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 tremadamente 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 aplicación 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()

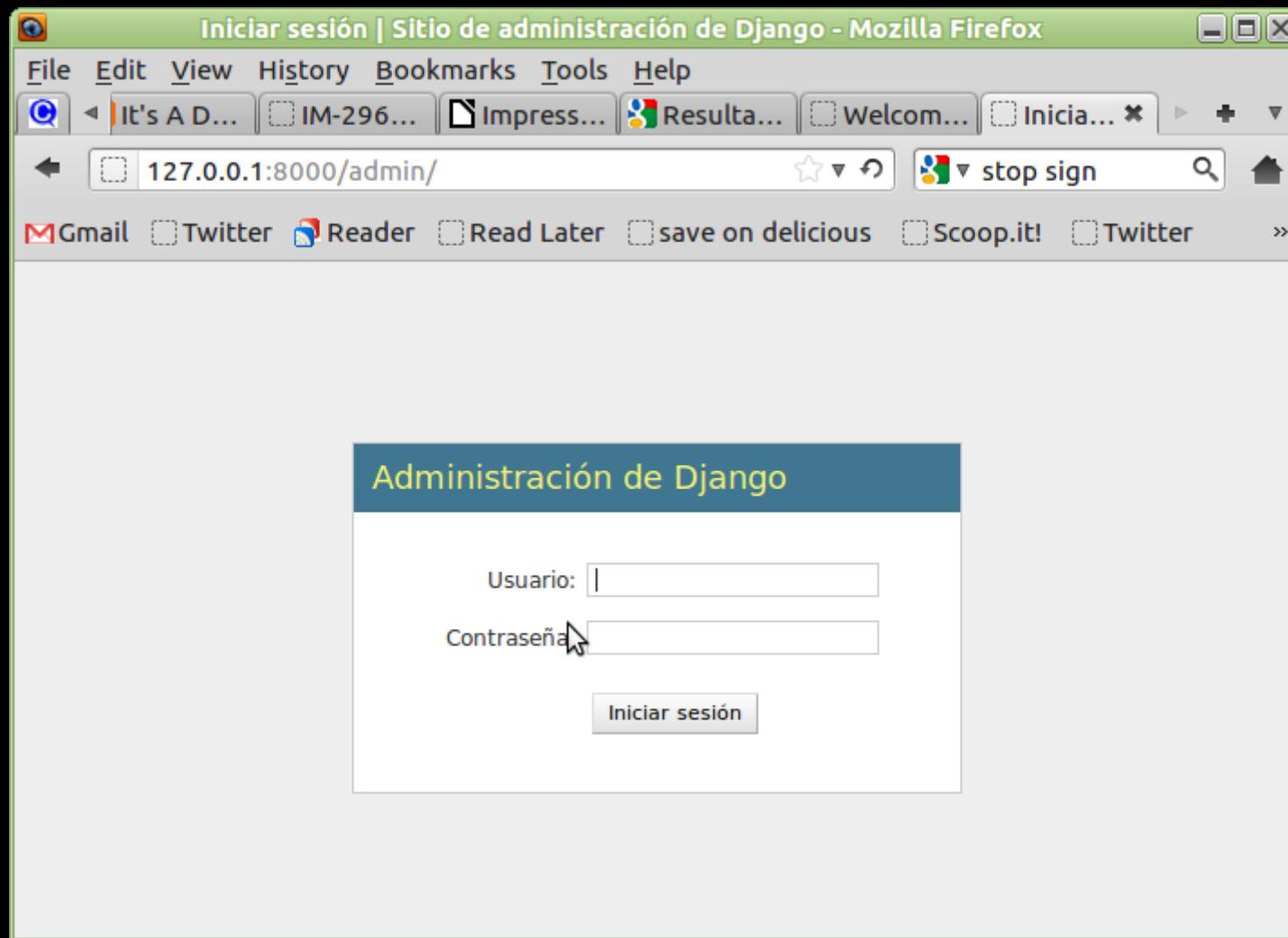
urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'shield.views.home', name='home'),
    # url(r'^shield/', include('shield.foo.urls')),

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

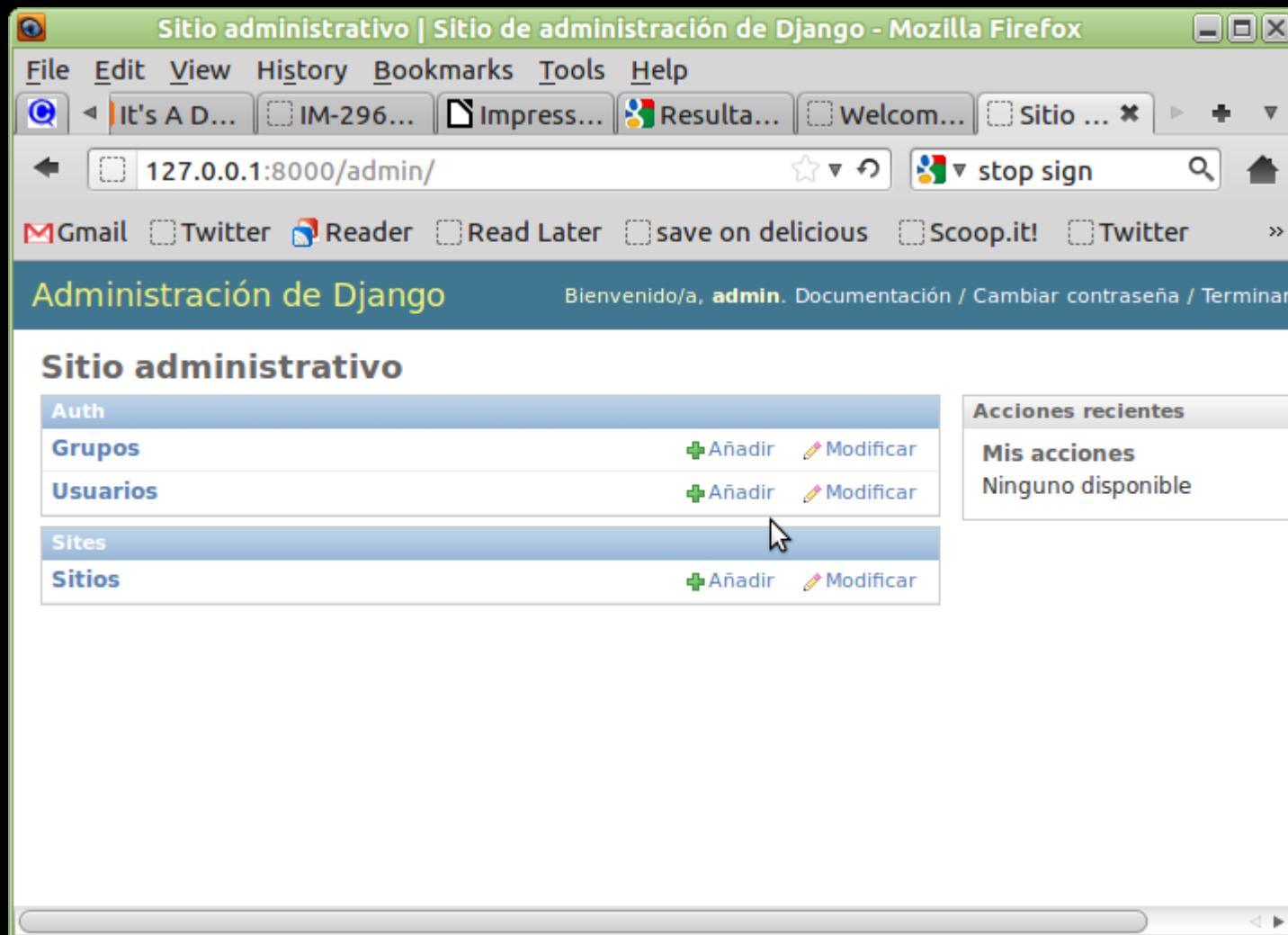
    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
)
```

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!

The screenshot shows the Django Admin interface running in Mozilla Firefox. The title bar reads "Sitio administrativo | Sitio de administración de Django - Mozilla Firefox". The address bar shows the URL "127.0.0.1:8000/admin/". The main content area displays the "Sitio administrativo" dashboard with three sections: "Auth", "Shield", and "Sites".

Auth

- Grupos [Añadir](#) [Modificar](#)
- Usuarios [Añadir](#) [Modificar](#)

Shield

- Avistamientos [Añadir](#) [Modificar](#)
- Equipos [Añadir](#) [Modificar](#)
- Heroes [Añadir](#) [Modificar](#)
- Poders [Añadir](#) [Modificar](#)

Sites

- Sitios [Añadir](#) [Modificar](#)

Acciones recientes

Mis acciones
Ninguno disponible

Añadimos unos cuantos héroes

Escoja heroe a modificar

Añadir heroe +

<input type="checkbox"/>	Nombre heroe	Identidad	Nivel	Lista poderes	Equipo
<input type="checkbox"/>	Spiderman	Peter Parker	30	Superagilidad, Superfuerza, Sentido arágnido, 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,  
        )
```

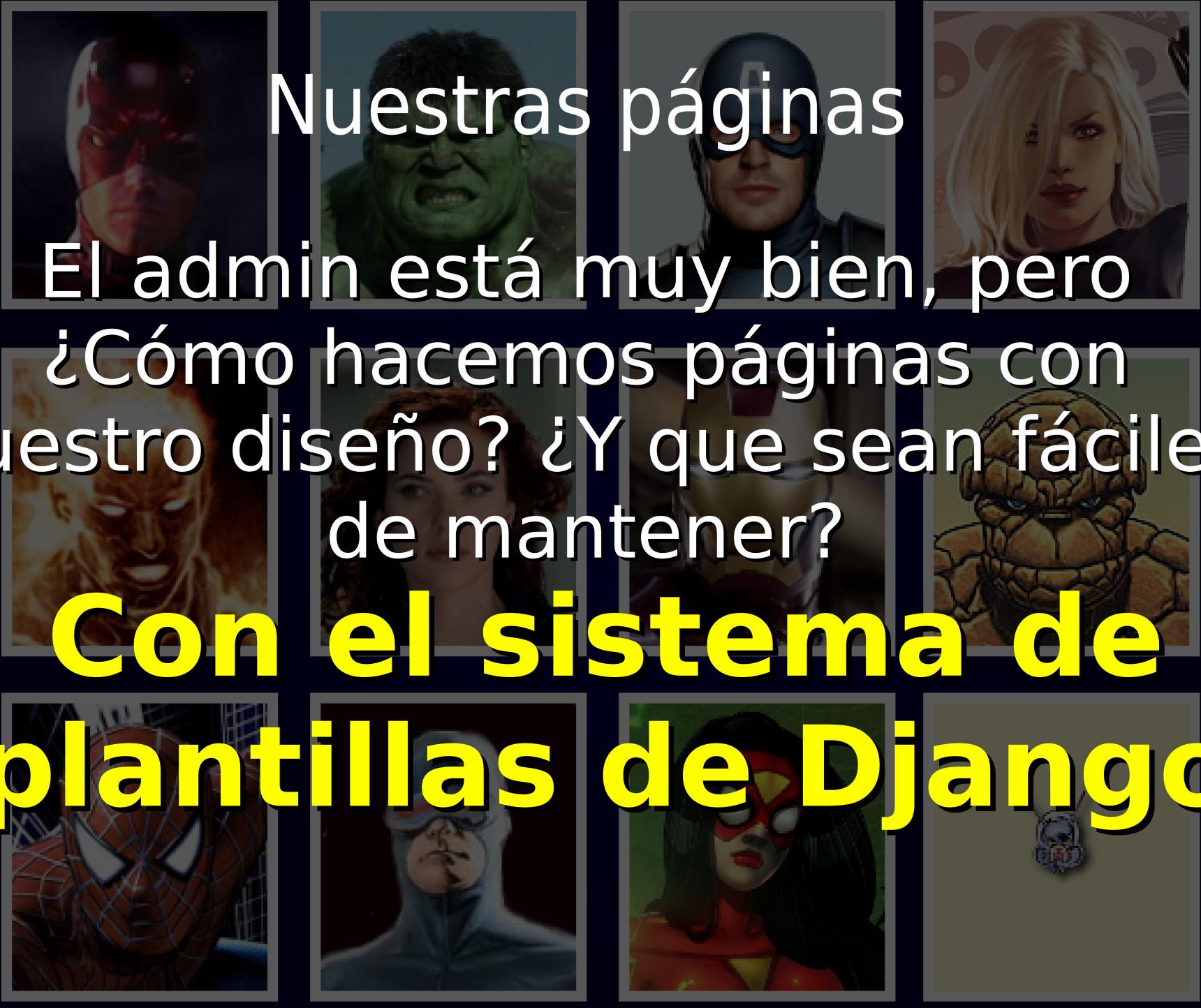
Afinando el admin (2)

- 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?



Nuestras páginas

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

**Con el sistema de
plantillas de Django**

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

WELCOME TO UNOFFICIAL CHANNELS

Jul 23 [Ante sed fringilla](#)

Jul 18 [Turpis dolor risus](#)

Jul 7 [Nunc venenatis iaculis](#)

Jul 2 [Lorem ipsum etiam](#)

Jun 28 [Sed phasellus dolor](#)

Jun 24 [Arcu phasellus](#)

This is **Unofficial Channels**, a free, fully standards-compliant CSS template by [Free CSS Templates](#). The images used in this template are from [Fotograph](#). 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 recomienda separar los contenidos estáticos de los dinámicos. Como primer paso, vamos a cambiar de sitio las hojas de estilo y las imágenes, que estaban en una ruta relativa a la página, para que están bajo una ruta absoluta: **/static**
- La hoja de estilo estará en **/static/css/**
- Las imágenes en **/static/images/**
- Debemos modificar el código de los ficheros **base.html** y **style.css** para reflejar estos cambios

La aplicación staticfiles

- Desde Django 1.3 hay una aplicación que se encarga de gestionar los contenidos estáticos, **staticfiles**
- Recopila, a partir de las aplicaciones instaladas, todos los recursos estáticos en una ubicación común, lista para ser servida en producción
- Dicha ubicación esta definido en la variable **STATIC_ROOT**, en el fichero **settings.py**

La aplicación staticfiles (2)

- 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 (3)

- La aplicación buscará en cada app instalada una carpeta **static**, y copiará su contenido a la carpeta **static** común
- En desarrollo, con el comando **runserver**, las diferentes carpetas **static** están siendo observadas; cualquier cambio en un **static** particular se copia automáticamente al **static** común

La aplicación staticfiles (4)

- En explotación, usaremos:
manage.py collectstatic --noinput
- Solo copiará ficheros nuevos o que se hayan modificado
- La opción **--noinput** es para que no nos pida confirmación de la operación

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 absoluto **/static**
- También cambiamos algunos textos en **base.html** y sustituimos los gráficos **banner.jpg** y **pic01.jpg** por otros más adecuados y/o molones

Probemos con una vista simple

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

```
urlpatterns = patterns('',
    ...
    (r'^/ejemplo/$', 'shield.views.ejemplo'),
    ...
)
```

El código de la vista

- Mas simple que el mecanismo de un chupete

```
from django.shortcuts import render_to_response

def ejemplo(request):
    return render_to_response(
        'shield/base.html',
        locals()
    )
```

Ajustamos la plantilla

The screenshot shows a website template for the Strategic Homeland Intervention, Enforcement and Logistics Division (S.H.I.E.L.D.). The header features a dark teal bar with navigation links: HEROES, ENLACES, SERVICIOS, ACERCA DE, DESCARGAS, BLOG, ALERTAS, and CONTACTO. Below this is a search bar with a magnifying glass icon. The main title "S.H.I.E.L.D." is displayed in large white letters. A large banner image of a S.H.I.E.L.D. Helicarrier flying through the sky is the background for the header and main content area. A black overlay bar across the banner contains the text "STRATEGIC HOMELAND INTERVENTION, ENFORCEMENT AND LOGISTICS DIVISION". The main content area has a light gray background. On the left, there's a sidebar with the heading "AUGUE ORCI" and a list of posts with dates and titles: Jul 23 - [Ante sed fringilla](#), Jul 18 - [Turpis dolor risus](#), Jul 7 - [Nunc venenatis iaculis](#), Jul 2 - [Lorem ipsum etiam](#), Jun 28 - [Sed phasleus dolor](#), and Jun 24 - [Arcu phasellus](#). On the right, the heading "WELCOME TO SHIELD" is followed by the S.H.I.E.L.D. logo (an eagle with wings spread, perched on a shield with the word "SHIELD" below it). To the right of the logo is a paragraph of text about the template: "This is **Unofficial Channels**, a free, fully standards-compliant CSS template by [Free CSS Templates](#). The images used in this template are from [Fotogrph](#). This free template is released under a [Creative Commons Attribution 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 :)".

Hagamos otra página

- Furia nos pide una página con un listado de los héroes que tenemos registrados, junto con sus poderes correspondientes
- Los pasos son siempre los mismos:
 - 1) Definimos la URL
 - 2) Mapeamos la URL deseada con la vista
 - 3) Escribimos la vista
 - 4) Si la vista usa una plantilla específica, escribir la plantilla

El URL de la página “héroes”

- La URL bien puede ser **/heroes/**
- En el fichero **urls.py**, realizamos el mapeo:

```
urlpatterns = patterns('',
    ...
    (r'^/heroes/$', 'shield.views.heroes'),
    ...
)
```

El código de la vista “Héroes”

```
def heroes(request):  
    heroes = models.Heroe.objects.all()  
    return render_to_response(  
        'shield/heroes.html',  
        locals()  
    )
```

Nota: `locals()` es una función propia de Python que nos devuelve un diccionario con todas las variables que haya accesibles en ese momento en el ámbito local; en este caso, las variables `heroes` (definida en la primera línea) y `request` (pasada como parámetro)

La plantilla usada por la vista “Héroes”

```
{% extends "shield/base.html" %}

{% block content %}

<div class="box">

<h2>Superheroes</h2>

<ul>{% for sh in heroes %}

    <li>{{ sh }}  

( {{ sh.lista_poderes }} )</li>

{% endfor %}</ul>

</div>

{% endblock %}
```

¡Funciona!

The screenshot shows a website with a dark teal header. The header contains a navigation menu with links: HEROES, ENLACES, SERVICIOS, ACERCA DE, DESCARGAS, BLOG, ALERTAS, and CONTACTO. Below the menu is a search bar with a magnifying glass icon. The main content area features a large, blurred image of the S.H.I.E.L.D. Helicarrier flying through a cloudy sky. Overlaid on this image is a dark horizontal bar containing the text "STRATEGIC HOMELAND INTERVENTION, ENFORCEMENT AND LOGISTICS DIVISION". The page is divided into two columns. The left column is titled "AUGUE ORCI" and lists several blog posts with dates and titles. The right column is titled "SUPERHEROES" and lists superhero abilities. At the bottom of the page, there is a footer note.

HEROES ENLACES SERVICIOS ACERCA DE DESCARGAS BLOG ALERTAS CONTACTO

S.H.I.E.L.D.

STRATEGIC HOMELAND INTERVENTION, ENFORCEMENT AND LOGISTICS DIVISION

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 phasleus dolor](#)

Jun 24 [Arcu phasellus](#)

MORBI TORTOR EGET

Lectus venenatis pharetra mauris

SUPERHEROES

Spiderman (Superagilidad, Superfuerza, Sentido arágnido, Adhesividad corporal)

Daredevil (Superagilidad, Radar)

La Cosa (Superfuerza, Superresistencia, Blindaje)

Iron Man (Volar, Superfuerza, Blindaje, Rayos repulsores, Contramedidas, Cibernética, Supersentidos, Extremis)

Antorcha Humana (Volar, Rayos de energía, Supernova)

Capitán América (Superagilidad, Superresistencia)

Cíclope (Rayos ópticos)

Mujer Invisible (Invisibilidad, Campos de fuerza)

Spiderwoman (Superagilidad, Superfuerza, Adhesividad corporal, Rayos de energía)

Explicando la plantilla

- Las plantillas Django son texto (normalmente html, pero también podría ser texto plano, csv, json, etc...) con dos tipos de marcas especiales: **variables** y **etiquetas**
- Los **variables** serán reemplazadas por los valores que contienen cuando la plantilla se evalúa
- Las **etiquetas** sirven para controlar la lógica de la plantilla

Las variables

- Las variables se escriben de la siguiente forma:
`{{ nombre_variable }}`
- El nombre de la variable puede contener caracteres alfanuméricos, el carácter “_” y se puede usar el carácter punto “.” para acceder a atributos, métodos, etc... Pero no puede contener espacios

El punto para acceder a atributos

- Cuando el sistema de plantillas encuentra un punto, intenta localizar algo que encaje, siguiendo este orden:
 1. Realizando una búsqueda como si fuera un diccionario
 2. Buscando un atributo
 3. Buscando un método
 4. Accediendo como si fuera un índice

Si no existe el atributo

- Si se intenta usar un atributo o una variable que no existe, el sistema de plantillas insertará el valor que haya definido en la variable **TEMPLATE_STRING_IF_INVALID** en el fichero `settings.py`
- Por defecto esta variable se inicializa a una cadena de texto vacía, es decir, que este tipo de errores se ocultan. Puede ser interesante, si estamos en desarrollo, poner algún mensaje de advertencia

Los filtros

- Las variables se pueden modificar mediante filtros: funciones que aceptan el valor de la variable (y, opcionalmente, un parámetro) y devuelven un resultado, que será el que se represente finalmente
- Así, `{{ nombre|upper }}` se sustituirá por el contenido de `nombre` en mayúsculas
- Los filtros se pueden encadenar: el resultado de uno se convierte en la entrada del siguiente
- Puedes escribir tus propios filtros

Las etiquetas

- Las etiquetas tienen la forma:

```
{% nombre_etiqueta %}
```

Lo más habitual es que las etiquetas delimiten un área, con una etiqueta de inicio y otra de final:

```
{% nombre_etiqueta %}
```

...

```
{% endnombre_etiqueta %}
```

Para que sirven las etiquetas

- Tienen diversas utilidades
 - Crear texto en la salida
 - Control de flujo
 - Cargar datos o recursos
- Django viene con unas 20 etiquetas, las más usadas son: **extends**, **include**, **block**, **for** e **if**.

Etiquetas de control de flujo: for

- La etiqueta **for** nos permite realizar un bucle sobre cualquier iterable (listas, tuplas, etc...). Por ejemplo, si **heroes** es un *array* de instancias del modelo **Heroe**, podemos listarlos así:

```
{% for h in heroes %}  
    <li>{{ h.nombre }}</li>  
{% endfor %}
```

Etiquetas de control de flujo: if/else

- La etiqueta **if** nos permite incluir o no un bloque de texto según el valor de una variable. Por ejemplo, si la variable **heroes** anterior fuera un array vacío (que se evalua a **False**) solo se incluiría en la salida el segundo bloque:

```
{% if heroes %}  
  <h3>Hay {{ heroes|length }} heroes</h3>  
{% else %}  
  <h3>i No hay heroes!</h2>  
{% endfor %}
```

Herencia de plantillas

- La mayor potencia de las plantillas de Django viene de la herencia
- La idea es definir una plantilla general (llamada tradicionalmente **base.html**) y luego ir definiendo versiones particularizadas
- La etiqueta **block** permite definir las áreas de la plantilla que se pueden cambiar.
- Veámoslo con un ejemplo

```
<!DOCTYPE html>
<html lang="en"><head>
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}My amazing site{% endblock %}</title></head>
<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
        {% endblock %}
    </div>
    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en"><head>
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}My amazing site{% endblock %}</title></head>
<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
        {% endblock %}
    </div>
    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

Título

Sidebar

Contenido

base.html

Personalización mediante herencia

- Ahora haremos una plantilla especializada para la página de heroes.
- Casi todo el **base.html** está bien, el bloque **sidebar** es correcto, pero queremos cambiar el título e incluir contenido en el bloque **content**
- Creamos una nueva plantilla, que deriva de **base.html** (con la etiqueta **extends**) y reescribimos los bloques que queramos.
- Siguiendo con el ejemplo:

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

{% block title %}Listado de heroes{% endblock %}

{% block content %}

<h1>Listado de heroes registrados en  
nuestra base de datos</h1>

<ul>

    {% for h in heroes %}

        <li>{{ h.nombre }} {{ h.lista_poderes }}</li>

    {% endfor %}

</ul>

{% endblock %}
```

Herencia de plantillas

- En resumen, la idea se definir un fichero **base.html** que nos sirva para todo el *site*, e ir haciendo versiones más especializadas a medida que nos hagan falta.
- Se puede usar el tag **include** para añadir partes comunes. Por ejemplo, el contenido del *sidebar* se podría poner en un fichero aparte, e incluirlo en el base.

Cosas que se han quedado en el tintero

- Todos las etiquetas y filtros incluidas, además de las que puedes escribir tu mismo
 - El sistema de consultas a la base de datos
 - Todos las aplicaciones que vienen por defecto
 - Las posibilidades del admin
 - Atajos (*shortcuts*) y decoradores
 - Trabajar con archivos
 - Vistas genéricas
 - El despliegue en producción
 - Añadir ordenes personalizadas al `manage.py`
 - Middleware
 - Sistemas de cache
 - Relaciones genéricas
 - Internacionalización
 - Vistas como clases
 - Formularios
 - Módulos (apps) de terceros
- ... Un mundo de posibilidades

El código de la aplicación

El código de la aplicación se puede descargar desde **bitbucket**:

```
hg clone ssh://hg@bitbucket.org/euribates/shield
```

O consultar vía web:

<https://bitbucket.org/euribates/shield>

Recursos en la web (1)

- Sobre Python

<http://www.python.org/>

<http://www.learnpython.org/>

- Sobre Django

<https://www.djangoproject.com/>

<https://code.djangoproject.com/wiki/DjangoResources>

Recursos en la web (2)

- Para conseguir trabajo como desarrollador Django

<http://www.djangoprojectjobs.org/>

<http://djangogigs.com/>

<http://www.twago.es/>

- Aplicaciones y desarrollos hechos con Django

<http://www.djangoproject-apps.com/>

<http://www.djangosites.org/>



S.H.I.E.L.D.TM



Juan Ignacio Rodríguez de León

email ~ euribates@gmail.com

twitter ~ @jileon

blog ~ www.elornitorrincoenmascarado.com

Tenerife
Lan Party 2k12 TL