

django

el curso

Día 1 - II

Tutorial

Un primer acercamiento a Django mediante la creación de una aplicación

Preparamos entorno de trabajo:

```
$ mkdir trabajo
$ cd trabajo
```

Creamos lo que en Django recibe el nombre de *proyecto*:

```
$ python /ruta/a/django-admin.py startproject mysite
$ cd mysite
$ ls
__init__.py
manage.py
settings.py
urls.py
```

`django-admin` puede mostrar ayuda:

```
$ python /ruta/a/django-admin.py help
$ python /ruta/a/django-admin.py help startproject # u otro cmd
$ python manage.py help
$ python manage.py help startproject # u otro cmd
```

Verificamos que podemos ejecutar el servidor Web de desarrollo:

```
$ python manage.py runserver
```

```
Validating models...
```

```
0 errors found
```

```
Django version 1.1 pre-alpha, using settings 'mysite.settings'
```

```
Development server is running at http://127.0.0.1:8000/
```

```
Quit the server with CONTROL-C.
```

Asignamos valores a los *settings* básicos:

```
$ edit settings.py
```

```
...  
DATABASE_ENGINE='sqlite3'  
DATABASE_NAME='mysite.db' # para sqlite3: nombre de archivo  
...  
# no críticos:  
TIME_ZONE='America/Cordoba'  
LANGUAGE_CODE='es-ar'
```

Revisamos la *lista de aplicaciones instaladas* en el *proyecto*:

```
$ view settings.py
```

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

Las aplicaciones activas por omisión en un proyecto recién creado son aplicaciones que no pertenecen al núcleo de Django (**contrib**).

Cada una de ellas hace uso de la base de datos (tablas, datos), la tarea de la sincronización entre:

- Lo expresado en el código de las mismas y
 - La (flamante en nuestro caso) BD
- es también una comando de **django-admin**: **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
```

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 'ramiro'):

E-mail address: **django@example.com**

Password: # tipeo 'a ciegas'

Password (again): # tipeo 'a ciegas'

Superuser created successfully.

Installing index for auth.Permission model

Installing index for auth.Message model

El comando **syncdb** recorre las aplicaciones listadas en `INSTALLED_APPS` y crea las tablas faltantes (todas en nuestro caso) en la BD.

Ahora podemos crear nuestra aplicación y *curiosear* un poco:

```
$ python manage.py startapp polls
$ cd polls
$ ls
__init__.py
models.py
views.py
```

`models.py` es especial, creamos una estructura de datos para nuestra app:

```
$ edit models.py
```

```
from django.db import models

# Create your models here.

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
```

Tenemos que avisarle a Django sobre nuestra aplicación, volvemos a trabajar sobre *settings*:

```
$ cd .. # al directorio padre 'mysite'
```

Agregamos la misma:

```
$ edit settings.py
```

```
...
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'polls', ◀
)
```

Han quedado no sincronizados

→ Lo expresado en los modelos de **models.py** y

→ La base de datos en si misma

en lo referente a lo que cada uno de ellos “piensa” debería ser el contenido de esta última.

Tenemos que re-sincronizar, pero antes exploremos un poco:

```
$ python manage.py sql polls
BEGIN;
CREATE TABLE "polls_poll" (
  "id" integer NOT NULL PRIMARY KEY,
  "question" varchar(200) NOT NULL,
  "pub_date" datetime NOT NULL
)
;
CREATE TABLE "polls_choice" (
  "id" integer NOT NULL PRIMARY KEY,
  "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
  "choice" varchar(200) NOT NULL,
  "votes" integer NOT NULL
)
;
COMMIT;
```

Dialecto SQL de SQLite 3 (recordemos `settings.DATABASE_ENGINE='sqlite3'`).

Otros *management commands* (ese es su nombre):

```
$ python manage.py validate # sqlindexes, sqlall, ..
0 errors found
```


Ahora si, **syncdb** *'amos*:

```
$ python manage.py syncdb
Creating table polls_poll
Creating table polls_choice
Installing index for polls.Choice model
```

En otro momento podríamos querer saber cómo borrar nuestra aplicación de la BD:

```
$ python manage.py sqlclear
Error: Enter at least one appname.

$ python manage.py sqlclear polls
BEGIN;
DROP TABLE "polls_choice";
DROP TABLE "polls_poll";
COMMIT;
```

sqlclear trabaja en forma similar a **syncdb** pero a nivel aplicación y no a nivel *proyecto*.

Django nunca modifica ni borra automáticamente tablas en la base de datos.

Con la estructura referente a la aplicación ya creada en BD, podemos explorar la API del ORM para agregar y jugar con datos. El comando `shell` nos ayuda en esto ya que prepara un entorno adecuado:

```
$ python manage.py shell
Python 2.5.2 (r252:60911, Sep 29 2008, 21:15:13)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
(InteractiveConsole)
>>> █
```

Importemos nuestros modelos:

```
>>> from polls.models import Poll, Choice

# Esto es un comentario -- No hay encuestas, todavía.
>>> Poll.objects.all()
[]

# Creamos una nueva encuesta.
>>> import datetime
>>> p = Poll(question=u"Cómo vá?", pub_date=datetime.datetime.now())

# Grabamos el objeto Poll en la BD
>>> p.save()
```

```
# Ahora ya tiene un identificador (ID)
>>> p.id
1

# Accedemos a columnas a través de atributos del objeto, con Python.
>>> p.question
u'Cómo vá?' # posiblemente codificado (acentos)
>>> p.pub_date
datetime.datetime(2008, 10, 28, 18, 40, 21)

# Cambiamos valores modificando dichos atributos y
# llamando a save().
>>> p.pub_date = datetime.datetime(2008, 10, 31, 0, 0)
>>> p.save()

# objects.all() muestra todas las encuestas que hay e la BD.
>>> Poll.objects.all()
[<Poll: Poll object>]
```

Notar la forma (estándar de Python) en la que aparece el objeto Poll cuando se pide una representado textual del mismo:

```
[<Poll: Poll object>]
```

Para facilitar nuestro trabajo en este shell interactivo y para depuración podemos personalizar dicha representación para que sea algo mas útil.

Para ello modificamos los modelos agregándoles métodos `__unicode__`, y ya que estamos, un método extra a Poll (salimos del shell):

```
$ edit polls/models.py
```

```
...
import datetime

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    def __unicode__(self):
        return self.question

    def was_published_today(self):
        return self.pub_date.date() == datetime.date.today()

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()

    def __unicode__(self):
        return self.choice
```

Volvemos al shell:

```
$ python manage.py shell
[...]
>>> from polls.models import Poll, Choice

# Buscamos el objeto que ya hemos agregado usando un atajo provisto
# idéntico a usar Poll.objects.get(id=1).
>>> Poll.objects.get(pk=1)
<Poll: Cómo vá?>
# (notar que la representación cambió)

# Probamos nuestro método was_published_today().
>>> p = Poll.objects.get(pk=1)
>>> p.was_published_today()
False # No estamos en Halloween

>>> Poll.objects.all()
[<Poll: Cómo vá?>]

# Django provee una API para consultas a la BD basada en argumentos
# 'keyword'.
>>> Poll.objects.filter(id=1)
[<Poll: Cómo vá?>]
>>> Poll.objects.filter(question__startswith=u'Cómo')
[<Poll: Cómo vá?>]
```

Mas consultas:

```
# Buscamos la encuesta del año 2008:
>>> Poll.objects.get(pub_date__year=2008)
<Poll: Cómo vá?>

>>> Poll.objects.get(id=2)
Traceback (most recent call last):
  ...
DoesNotExist: Poll matching query does not exist

# Creamos un par de objetos Choice relacionados a la encuesta:
>>> p = Poll.objects.get(pk=1)
>>> p.choice_set.create(choice='Todo bien', votes=0)
<Choice: Todo bien>
>>> p.choice_set.create(choice='Bastante mal', votes=0)
<Choice: Bastante mal>
>>> c = p.choice_set.create(choice='Preocupado', votes=0)
```

```
...  
  
# Los objetos Choice tiene disponible una API para acceder a sus  
# objetos Poll relacionados.  
>>> c.poll  
<Poll: Cómo vá?>  
  
# y viceversa: objetos Poll pueden acceder a objetos Choice.  
>>> p.choice_set.all()  
[<Choice: Todo bien>, <Choice: Bastante mal>, <Choice: Preocupado>]  
>>> p.choice_set.count()  
3  
  
# La API sigue automáticamente las relaciones tanto como sea  
# necesario sin que existan limitaciones arbitrarias.  
# Usamos doble guión bajo para separar relaciones.  
# Encontrar todos los Choices para cualquier encuesta cuya fecha  
# de publicación sea durante el año 2008.  
>>> Choice.objects.filter(poll__pub_date__year=2008)  
[<Choice: Todo bien>, <Choice: Bastante mal>, <Choice: Preocupado>]  
  
# Por último veamos como borrar una de las opciones de la  
# encuesta con el método .delete().  
>>> c = p.choice_set.filter(choice__startswith='Preocupado')  
>>> c.delete()
```

La aplicación de administración (a.k.a. Admin)

Agregamos la misma:

```
$ edit settings.py
```

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

Hemos agregado una aplicación al *proyecto*, debemos ejecutar **syncdb**:

```
$ python manage syncdb
```


Modificamos la configuración de manejo de URLs:

```
$ edit urls.py
```

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

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

urlpatterns = patterns('',
    # Example:
    # (r'^mysite/', include('mysite.foo.urls')),

    # Uncomment the admin/doc line below and add 'django.contrib.admindocs'
    # to INSTALLED_APPS to enable admin documentation:
    # (r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    (r'^admin/(.*)', admin.site.root),
)
```

Ejecutamos el servidor Web de desarrollo:

```
$ python manage.py runserver
```

```
Validating models...
```

```
0 errors found
```


```
Django version 1.1 pre-alpha, using settings 'mysite.settings'
```

```
Development server is running at http://127.0.0.1:8000/
```

```
Quit the server with CONTROL-C.
```

Ya podemos acceder con un navegador Web a

<http://localhost:8000/admin/>



The image shows a screenshot of the Django Admin interface. At the top, there is a blue header with the text "Administración de Django" in yellow. Below the header, there is a white form area. The form contains two input fields: "Usuario:" followed by a text input box, and "Contraseña:" followed by a password input box. Below these fields is a button labeled "Identificarse".

Nos identificamos con los datos del usuario que ingresamos cuando ejecutamos **syncdb** por primera vez en el *proyecto*.



(Recordar que **auth** y **sites** son aplicaciones).

Notar que todavía no hay nada relacionado con nuestra aplicación **polls**. Para ello debemos registrar uno o mas de sus modelos en la aplicación Admin, creamos el archivo **admin.py** en el directorio de la aplicación:

```
$ edit polls/admin.py
```

```
from models import Poll
from django.contrib import admin

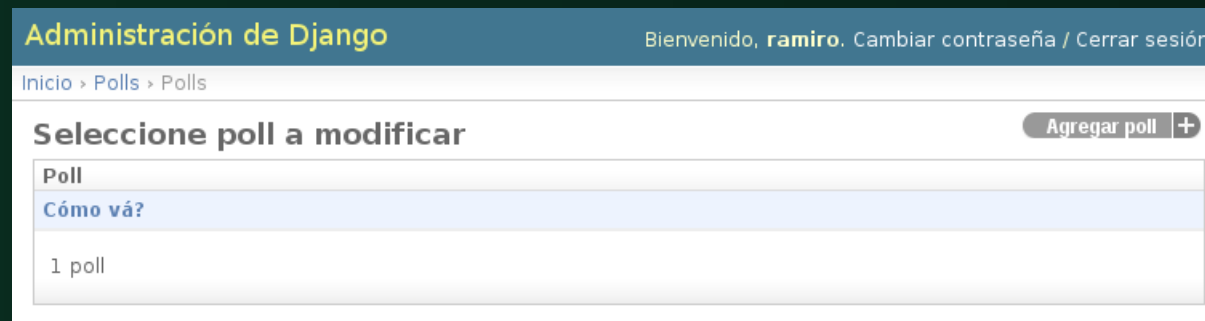
admin.site.register(Poll)
```

Luego de eso reiniciamos el servidor web de desarrollo.

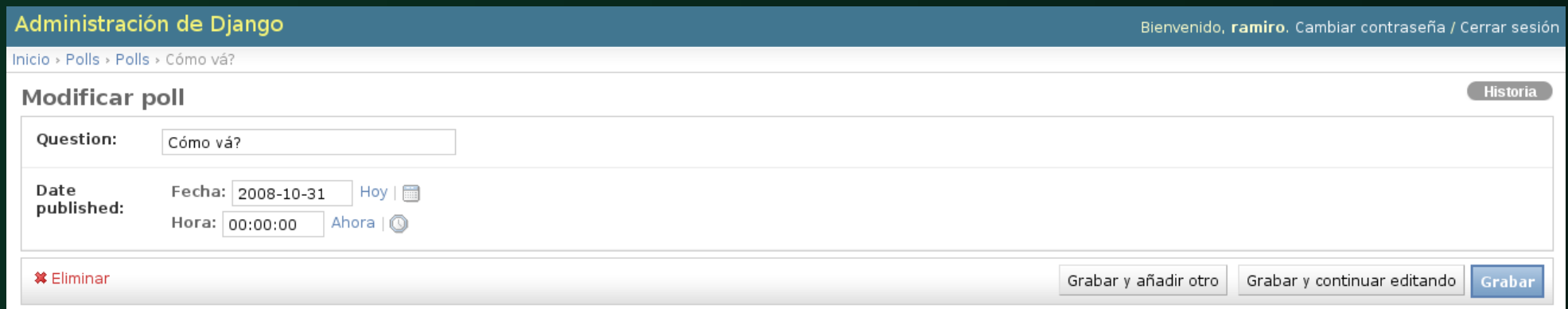
Una vez que ingresamos nuevamente a la aplicación de administración podemos ver que ahora si aparece Polls como modelo *administrable*:



Accedemos a la aplicación:



Y de allí, a la encuesta:



Podemos personalizar la aplicación de administración con muy poco código.

Por ejemplo, cambiar el orden de los campos en el formulario de edición de una encuesta. Para eso reemplazamos la línea `admin.site.register(Poll)` en el archivo `polls/admin.py`:

```
$ edit polls/admin.py
```

```
from models import Poll
from django.contrib import admin

admin.site.register(Poll)
class PollAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question']

admin.site.register(Poll, PollAdmin)
```

Resultado:



Administración de Django

Inicio > Polls > Polls > Cómo vá?

Modificar poll

Date published:	Fecha:	<input type="text" value="2008-10-31"/>	Hoy
	Hora:	<input type="text" value="00:00:00"/>	Ahora
Question:	<input type="text" value="Cómo vá?"/>		

Eliminar

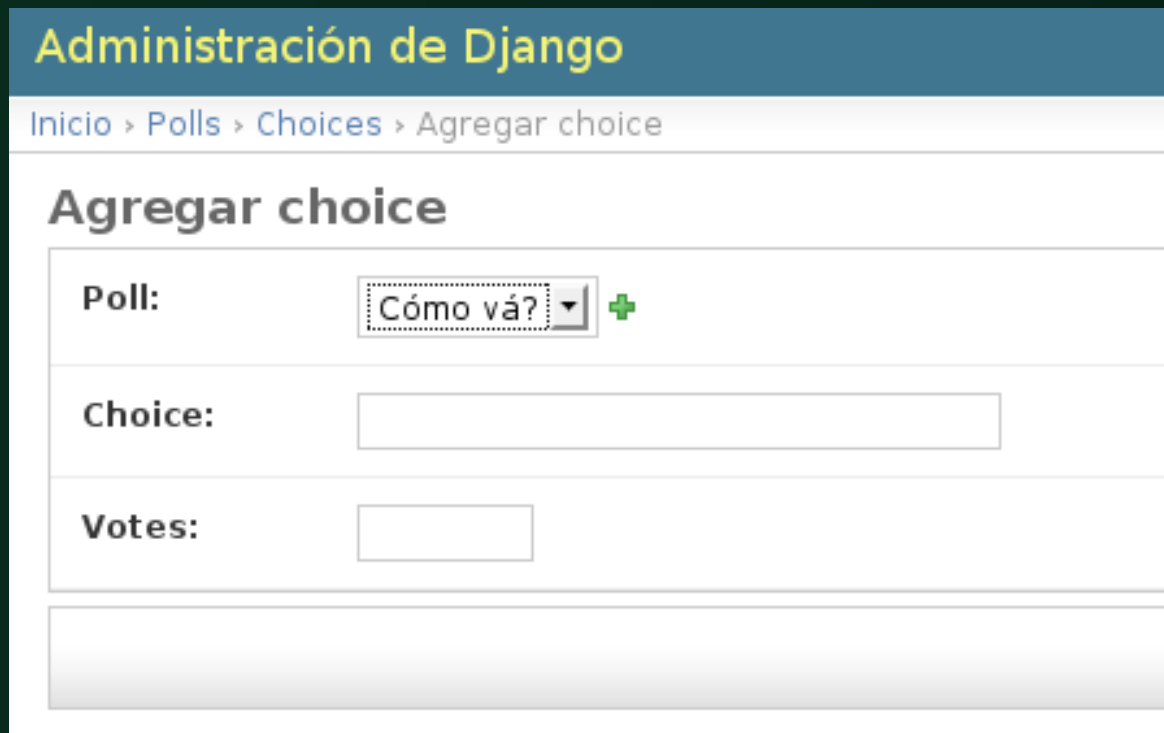
Hagamos también *administrables* los objetos **Choice**:

```
# admin.py
from mysite.polls.models import Poll, Choice
from django.contrib import admin

class PollAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question']

admin.site.register(Poll, PollAdmin)
admin.site.register(Choice)
```

Resultado:



The screenshot shows the Django Admin interface for adding a choice to a poll. The page title is "Administración de Django" and the breadcrumb trail is "Inicio > Polls > Choices > Agregar choice". The main heading is "Agregar choice". The form contains three fields: "Poll:" with a dropdown menu showing "Cómo vá?" and a green plus sign, "Choice:" with an empty text input field, and "Votes:" with an empty text input field.

`Choice` y `Poll` están relacionados, hagamos esto visible en la aplicación de administración:

```
# admin.py
from mysite.polls.models import Poll, Choice
from django.contrib import admin

class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3

class PollAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question']
    fieldsets = [
        (None, {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes':
['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Poll, PollAdmin)
admin.site.register(Choice)
```

Resultado:

Administración de Django Bienvenido, **ramiro**. [Cambiar contraseña](#) / [Cerrar sesión](#)

[Inicio](#) > [Polls](#) > [Polls](#) > [Cómo vá?](#)

Modificar poll Historia

Question:

Date information ([Mostrar](#))

Choices

Choice: #1

Choice:

Votes:

Choice: #2

Choice:

Votes:

Choice: #3

Choice:

Votes:

[✖ Eliminar](#)

(notar que también hemos comenzado a usar la opción/atributo *fieldsets* en la clase `PollAdmin`).

(notar que `ChoiceInline` descende de `StackedInline`, de allí el *layout* del *formset*).

Si cambiamos la clase *padre* por `TabularInline`:

```
# admin.py
from mysite.polls.models import Poll, Choice
from django.contrib import admin

class ChoiceInline(admin.TabularInline):
    model = Choice
    extra = 3

class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes':
['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Poll, PollAdmin)
```

Resultado:

Administración de Django Bienvenido, ramiro. Cambiar contraseña / Cerrar sesión

Inicio > Polls > Polls > Cómo vá?

Modificar poll Historia

Question:

Date information ([Mostrar](#))

Choice	Votes	Eliminar?
<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	

✖ Eliminar

Personalizando la página de lista de modelos de la aplicación

Podemos controlar qué campos y en que orden se muestran para cada modelo:

```
# admin.py
from mysite.polls.models import Poll, Choice
from django.contrib import admin

class ChoiceInline(admin.TabularInline):
    model = Choice
    extra = 3

class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes':
['collapse']}),
    ]
    inlines = [ChoiceInline]
    list_display = ('question', 'pub_date', 'was_published_today')

admin.site.register(Poll, PollAdmin)
```

(notar que en adición a los campos `question` y `pub_date` del modelo `Poll` usamos también el método `was_published_today`).

Resultado:

Administración de Django

Bienvenido, **ramiro**. [Cambiar contraseña](#) / [Cerrar sesión](#)

[Inicio](#) > [Polls](#) > [Polls](#)

Seleccione poll a modificar

Agregar poll **+**

Question	Date published	Was published today
Cómo vá?	31 Oct. 2008 medianoche	False

1 poll

This work is licensed under the Creative Commons Attribution-Noncommercial 2.5 Argentina License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/ar/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.