# **Modelos en Django**

Lo que queremos crear ahora es algo que almacene todas los posts de nuestro blog. Pero para poder hacerlo tenemos que hablar un poco sobre algo llamado objetos.

## **Objetos**

Hay un concepto en el mundo de la programación llamado programación orientada a objetos. La idea es que en lugar de escribir todo como una aburrida secuencia de instrucciones de programación podemos modelar cosas y definir cómo interactúan entre ellas.

Entonces, ¿qué es un objeto? Es un conjunto de propiedades y acciones. Suena raro, pero te daremos un ejemplo.

Si queremos modelar un gato crearemos un objeto Gato que tiene algunas propiedades, como por ejemplo color, edad, estado de ánimo (es decir, bueno, malo, con sueño ;)), dueño (que es un objeto Persona o, tal vez, en el caso de que el gato sea callejero, esta propiedad estará vacía).

Además el Gato tiene algunas acciones: ronronear, arañar o alimentarse (en la cual daremos al gato algo de ComidaDeGato, que podría ser un objeto independiente con propiedades, como por ejemplo, sabor).

Gato  
---------  
color  
edad  
humor  
dueño  
ronronear()  
rasguñar()  
alimentarse(comida\_de\_gato)

ComidaDeGato  
----------  
sabor

Básicamente se trata de describir cosas reales en el código con propiedades (llamadas propiedades del objeto) y las acciones (llamadas métodos).

Y ahora, ¿cómo modelamos los posts en el blog? Queremos construir un blog, ¿cierto?

Necesitamos responder a la pregunta: ¿Qué es un post de un blog? ¿Qué propiedades debería tener?

Bueno, seguro que nuestros posts necesitan un texto con su contenido y un título, ¿cierto? También sería bueno saber quién lo escribió, así que necesitamos un autor. Por último, queremos saber cuándo se creó y publicó la entrada.

Post  
--------  
title  
text  
author  
created\_date  
published\_date

¿Qué tipo de cosas podría hacerse con una entrada del blog? Sería bueno tener algún método que publique la entrada, ¿no?

Así que vamos a necesitar el método publish (publicar en Inglés).

Puesto que ya sabemos lo que queremos lograr, ¡podemos empezar a modelarlo en Django!

## **Modelos en Django**

Sabiendo qué es un objeto, podemos crear un modelo en Django para nuestros posts en el blog.

Un modelo en Django es un tipo especial de objeto que se guarda en la base de datos. Una base de datos es una colección de datos. Es un lugar en el cual almacenarás la información sobre usuarios, tus entradas de blog, etc. Utilizaremos una base de datos SQLite para almacenar nuestros datos. Este es el adaptador de base de datos predeterminado en Django -- será suficiente para nosotros por ahora.

Puedes pensar el modelo en la base de datos como una hoja de cálculo con columnas (campos) y filas (datos).

### **Crear una aplicación**

Para mantener todo en orden, crearemos una aplicación separada dentro de nuestro proyecto. Es muy bueno tener todo organizado desde el principio. Para crear una aplicación, necesitamos ejecutar el siguiente comando en la consola (dentro de la carpeta de djangogirls donde está el archivo manage.py):

(myvenv) ~/djangogirls$ python manage.py startapp blog

Vas a notar que se crea un nuevo directorio llamado blog que contiene una serie de archivos. Los directorios y archivos en nuestro proyecto deberían parecerse a esto:

djangogirls  
├── blog  
│ ├── \_\_init\_\_.py  
│ ├── admin.py  
│ ├── apps.py  
│ ├── migrations  
│ │ └── \_\_init\_\_.py  
│ ├── models.py  
│ ├── tests.py  
│ └── views.py  
├── db.sqlite3  
├── manage.py  
└── mysite  
 ├── \_\_init\_\_.py  
 ├── settings.py  
 ├── urls.py  
 └── wsgi.py

Después de crear una aplicación, también necesitamos decirle a Django que debe utilizarla. Lo hacemos en el archivo mysite/settings.py. Tenemos que encontrar INSTALLED\_APPS y agregar una línea que contiene 'blog', justo por encima de ]. El producto final debe tener este aspecto:

mysite/settings.py

INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'blog',  
]

### **Crear el modelo Post**

En el archivo blog/models.py definimos todos los objetos llamados Models. Este es un lugar en el cual definiremos nuestro post.

Vamos abrir blog/models.py, borramos todo lo que haya por defecto y escribimos un código como este:

blog/models.py

from django.db import models  
from django.utils import timezone  
  
class Post(models.Model):  
 author = models.ForeignKey('auth.User')  
 title = models.CharField(max\_length=200)  
 text = models.TextField()  
 created\_date = models.DateTimeField(  
 default=timezone.now)  
 published\_date = models.DateTimeField(  
 blank=True, null=True)  
  
 def publish(self):  
 self.published\_date = timezone.now()  
 self.save()  
  
 def \_\_str\_\_(self):  
 return self.title

Comprueba que has usado dos guiones bajos (\_) en cada lado del str. Esta convención se usa en Python con mucha frecuencia y a veces también se llaman "dunder" (abreviatura de "double-underscore" o, en español, "doble guión bajo").

Da un poco de miedo, ¿no? Pero no te preocupes, ¡vamos a explicar qué significan estas líneas!

Todas las líneas que comienzan con from o import son líneas para agregar algo de otros archivos. Así que en vez de copiar y pegar las mismas cosas en cada archivo, podemos incluir algunas partes con from... import ....

class Post(models.Model):, esta línea define nuestro modelo (es un objeto).

* class es una palabra clave que indica que estamos definiendo un objeto.
* Post es el nombre de nuestro modelo. Podemos darle un nombre diferente (pero debemos evitar espacios en blanco y caracteres especiales). Empieza siempre el nombre de una clase con una letra mayúscula.
* models.Model significa que Post es un modelo de Django, así Django sabe que debe guardarlo en la base de datos.

Ahora definimos las propiedades de las que hablábamos: title, text, created\_date, published\_date y author. Para ello tenemos que definir un tipo de campo (¿es texto? ¿un número? ¿una fecha? ¿una relación con otro objeto, es decir, un usuario?).

* models.CharField, así es como defines un texto con un número limitado de caracteres.
* models.TextField, este es para texto largo sin límite. Suena perfecto para el contenido de la entrada del blog, ¿no?
* models.DateTimeField, este es una fecha y hora.
* modelos.ForeignKey, este es una relación con otro modelo.

No vamos a explicar aquí cada pedacito de código porque nos tomaría demasiado tiempo. Deberías echar un vistazo a la documentación de Django si quieres saber más sobre los campos de los Modelos y cómo definir cosas diferentes a las descritas anteriormente (<https://docs.djangoproject.com/en/1.11/ref/models/fields/#field-types>).

¿Y qué sobre def publish(self):? Es exactamente el método publish que mencionábamos antes. def significa que es una función/método y publish es el nombre del método. Puedes cambiar el nombre del método, si quieres. La regla de nomenclatura es utilizar minúsculas y guiones bajos en lugar de espacios. Por ejemplo, un método que calcule el precio medio se podría llamar calcular\_precio\_medio.

Los métodos suelen devolver (return, en inglés) algo. Hay un ejemplo de esto en el método \_\_str\_\_. En este escenario, cuando llamemos a \_\_str\_\_() obtendremos un texto (**string**) con un título de Post.

Nota también que def publish(self): y def \_\_str\_\_(self): están indentadas dentro de nuestra clase. Debido a que Python es sensible a los espacios en blanco, necesitamos indentar nuestros métodos dentro de las clases. De otra forma, el método no pertenecerá a la clase, y puedes obtener un comportamiento inesperado.

Si algo todavía no está claro sobre modelos, ¡no dudes en preguntar! Sabemos que es complicado, sobre todo cuando aprendes lo que son funciones y objetos al mismo tiempo. Pero con suerte, ¡todo tiene un poco más de sentido para ti ahora!

### **Crear tablas para los modelos en tu base de datos**

El último paso aquí es agregar nuestro nuevo modelo a la base de datos. Primero tenemos que hacer saber a Django que hemos hecho cambios en nuestro modelo (¡lo acabamos de crear!). Escribe python manage.py makemigrations blog. Se verá así:

(myvenv) ~/djangogirls$ python manage.py makemigrations blog  
Migrations for 'blog':  
 blog/migrations/0001\_initial.py:  
 - Create model Post

**Nota:** Recuerda guardar los archivos que edites. De lo contrario, tu computadora ejecutará la versión anterior lo cuál quizás te muestre mensajes inesperados.

Django preparó un archivo de migración que ahora tenemos que aplicar a nuestra base de datos. Escribe python manage.py migrate blog y el resultado debería ser:

(myvenv) ~/djangogirls$ python manage.py migrate blog  
Operations to perform:  
 Apply all migrations: blog  
Running migrations:  
 Rendering model states... DONE  
 Applying blog.0001\_initial... OK

¡Hurra! ¡Nuestro modelo Post ya está en nuestra base de datos! Estaría bien verlo, ¿no? ¡Salta al siguiente capítulo para ver qué aspecto tiene tu Post!