Práctica 6: Modelado con Django

Lo que queremos crear ahora es algo que va a almacenar todos los posts en nuestro blog. Pero para poder hacerlo tenemos que hablar un poco acerca de algo llamado `objects`.

Objetos

Hay un concepto en el mundo de la programación llamado programación orientada a objetos [`object-oriented programming`]. La idea es que en lugar de escribir todo como una secuencia de instrucciones de programación podemos modelar cosas y definir cómo interactúan con las demás.

Entonces ¿Qué es un objeto? Es un conjunto de propiedades y acciones. Como por ejemplo. Si queremos modelar un gato crearemos un objeto `Cat` que tiene algunas propiedades, como son por ejemplo `color`, `age`, `mood` (es decir, bueno, malo, sueño), `owner` (que es un objeto `Person` o, tal vez, en el caso de que el gato sea callejero, esta propiedad estará vacía).

Y luego el `Cat` tiene algunas acciones: `purr`, `scratch`, o `feed` (en la cual daremos al gato algunos `CatFood`, que podría ser un objeto independiente con propiedades, como por ejemplo, `taste`).

```
Cat
--------
color
age
mood
owner
purr()
scratch()
feed(cat_food)

CatFood
----------
taste
```

Básicamente se trata de describir cosas reales en el código con propiedades (llamadas `object properties`) y las acciones (llamadas `methods`).

Queremos construir un blog, por lo que tenemos primero que responder algunas preguntas: ¿Qué es un post de un blog? ¿Qué características debe tener?

Seguro que nuestros posts necesitan un texto con su contenido y un título. También sería bueno saber quién lo escribió, así que necesitamos un autor. Por último, queremos saber cuándo el post fue creado y publicado.

```
Post
--------
title
text
author
created_date
published_date
```

¿Qué tipo de cosas podría hacerse con una entrada del blog? Sería bueno tener algún `method` que publique la entrada.

Así que vamos a necesitar el método `publish`. Puesto que ya sabemos lo que queremos lograr, podemos empezar a modelar 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 `database`. Una `database` es una colección de datos. Allí es el lugar en el cual almacenamos la información sobre usuarios, posts del blog, etc. Utilizaremos una base de datos SQLite para almacenar nuestros datos. Este es el adaptador de base de datos predeterminada en Django – será suficiente para nosotros por ahora.

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

Creando 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 `django-daw` donde está el archivo `manage.py`):

```
(myvenv) ~/django-daw$ python manage.py startapp blog
```

Vas a notar que se crea un nuevo directorio llamado `blog` y contiene una serie de archivos. Nuestros directorios y archivos en nuestro proyecto deberían verse así `tree . -L 2`:

```
django-daw
├── 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 añadir una línea que contenga `'blog'`, justo por encima del último `]`. El producto final debe tener este aspecto:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
]
```

Creando el modelo blog post

En el archivo `blog/models.py` definimos todos los objetos llamados `Models` - este es un lugar en el cual definiremos nuestro blog post. Vamos abrir `blog/models.py`, quitamos todo y escribimos un código como este:

```
from django.db import models
from django.utils import timezone


class Post(models.Model):
    author = models.ForeignKey('auth.User', on_delete=models.CASCADE)
    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
```

> Vuelve a verificar que usaste dos guiones bajos (`_`) en cada lado del `str`. Esta convención se utilizan con frecuencia en Python y a veces también los llamamos "dunder" (diminutivo en inglés de "double-underscore").

Puede parecer un poco intimidante ese código, pero veamos qué significan estas líneas.

Todas las líneas que comienzan con `from` o `import` son líneas para añadir 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 `object`).

- `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). Una clase siempre comienza con su primera letra en 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 que hablábamos: `title`, `text`, `created_date`, `published_date` y `author`. Para hacer eso tenemos que definir un tipo de

campo (¿es texto? ¿un número? ¿una fecha? ¿una relación con otro objeto, como un usuario?).

- `models.CharField` - así es como defines un texto con un número limitado de caracteres.
- `models.TextField` - esto es para textos largos sin un límite. Será ideal para el contenido de un post, ¿verdad?
- `models.DateTimeField` - esto es fecha y hora.
- `modelos.ForeignKey` - este es un vínculo con otro modelo.

No vamos a explicar cada pedacito de código, ya que nos tomaría demasiado tiempo. Debes 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.

¿Y qué sobre `def publish(self):`? Es exactamente nuestro método `publish` que mencionamos anteriormente. `def` significa que se trata de una función/método y `publish` es el nombre del método. Puedes cambiarlo, si quieres. La regla es que usamos minúsculas y guiones bajos en lugar de espacios (es decir, si quieres tener un método que calcule el precio medio, este podría llamarse `calculate_average_price`).

Los métodos muy a menudo devuelven algo. Hay un ejemplo de esto en el método `__str__`. En este escenario, cuando llamamos a `__str__()` obtendremos un texto (**string**) con un título de Post.

Si algo todavía no está claro sobre modelos, ¡no dudes en preguntar!. Puede ser muy complicado, sobre todo cuando estás entendiendo qué son funciones y objetos mientras sigues este documento.

Crear tablas para los modelos en tu `database`

El último paso es añadir nuestro nuevo modelo a nuestra base de datos. Primero tenemos que hacer que Django sepa que tenemos algunos cambios en nuestro modelo (acabamos de crearlo), escribe `python manage.py makemigrations blog`. Se verá así:

```
(myvenv) ~/django-daw$ python manage.py makemigrations blog
Migrations for 'blog':
  0001_initial.py:
  - Create model Post
```

> Nota: Recuerda guardar los cambio a los archivos que modificaste. De otra manera, la computadora ejecutara la version anterior lo cual podría dar errores inesperados.

Django preparará un archivo de migración que tenemos que aplicar ahora a nuestra base de datos escribiendo `python manage.py migrate blog`. El resultado debe ser:

```
(myvenv) ~/django-daw$ python manage.py migrate blog
Operations to perform:
  Apply all migrations: blog
Running migrations:
  Applying blog.0001_initial... OK
```

Nuestro modelo de Post está ahora en nuestra base de datos. En la siguiente práctica veremos cómo luce tu modelo Post.