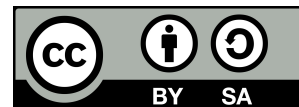


CURSO PYTHON

APLICACIONES WEB CON DJANGO



Autor: Jon Vadillo
www.jonvadillo.com



Contenidos

1. Introducción y fundamentos básicos
2. Crea tu primer proyecto en Django
3. Crea tu primera aplicación en django
4. **El modelo en Django, acceso a datos y la aplicación de administrador**
5. Vistas y plantillas en Django
6. Vistas basadas en clases (Class Based Views)
7. Formularios en Django

4. El modelo en Django, acceso a datos y la aplicación de administrador

Configuración básica de Django

- Base de datos

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        # Otras opciones: 'django.db.backends.postgresql', 'django.db.backends.mysql', ...  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'), # Nombre de la base de datos  
        # Aquí se incluyen otras propiedades como USER, PASSWORD, HOST, etc.  
    }  
}
```

- TIME_ZONE = 'Europe/Madrid'
- DEBUG = True

Base de datos

- Django usa un [Object-Relational-Mapper \(ORM\)](#) para **mapear las definiciones de Modelos** en el código Django con la **estructura de la base de datos**.
- Django sigue la pista a los cambios y puede crear scripts de migración de la base de datos (`/myproject/myapp/migrations/`) para **migrar automáticamente** la estructura de datos
- Comandos principales:
 - **`python manage.py makemigrations`**: crea (pero no aplica) las migraciones
 - **`python manage.py migrate`**: aplica realmente las migraciones a tu base de datos

Modelos

Model UserProfile:	
username	
first_name	
last_name	
email	

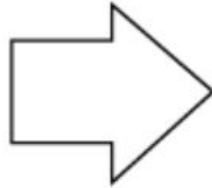


Table yourapp_userprofile	
ID	1001
username	big-nige
first_name	Nigel
last_name	George
email	nigel@example.com

Modelos

```
class Coche(models.Model):  
    matricula = models.CharField(max_length=10)  
    modelo = models.CharField(max_length=40)
```


Modelos

- Los modelos **definen la estructura de los datos** almacenados.
- La definición del modelo es independiente de la base de datos empleada.
- Los modelos están definidos, normalmente, en el archivo **models.py**
- Son implementados como [subclases de `django.db.models.Model`](#), y pueden incluir **campos, métodos y metadata**.

```
from django.db import models

class MyModelName(models.Model):

    # Campos
    my_field_name = models.CharField(max_length=20)
    ...

    # Metadata
    class Meta:
        ordering = ["-my_field_name"] # Indica cómo se ordenarán los datos en consultas

    # Métodos
    def get_absolute_url(self):
        return reverse('model-detail-view', args=[str(self.id)])
```

Tipos de atributos

- [CharField](#) se usa para definir **cadenas de longitud corta a media**. Debes especificar la `max_length` (longitud máxima) de los datos que se guardarán.
- [TextField](#) se usa para **cadenas de longitud grande** o arbitraria.
- [IntegerField](#) es un campo para almacenar valores de **números enteros**
- [DateField](#) y [DateTimeField](#) se usan para guardar/representar **fechas e información fecha/hora** (como en los objetos Python `datetime.date` y `datetime.datetime`, respectivamente).
- [EmailField](#) se usa para validar direcciones de **correo electrónico**.
- [FileField](#) e [ImageField](#) se usan para subir **ficheros e imágenes** respectivamente (el `ImageField` añade simplemente una validación adicional de que el fichero subido es una imagen). Éstos tienen parámetros para definir cómo y donde se guardan los ficheros subidos.

Tipos de atributos

- [AutoField](#) es un tipo especial de **IntegerField** **que se incrementa automáticamente**. Cuando no especificas una clave primaria para tu modelo, se añade -automáticamente- una de éste tipo.
- [ForeignKey](#) se usa para **especificar una relación uno a muchos con otro modelo** de la base de datos .
- [ManyToManyField](#) se usa para **especificar una relación muchos a muchos** (ej. un libro puede tener varios géneros, y cada género puede contener varios libros). Tienen el parámetro `on_delete` para definir que ocurre cuando un registro asociado se borra (ej. un valor de `models.SET_NULL` establecería simplemente el valor a NULL).

Creando mis primeros Modelos

```
from django.db import models

class Empresa(models.Model):
    nombre = models.CharField(max_length=50)
    cif = models.CharField(max_length=12)

class Trabajador(models.Model):
    # Campo para la relación
    empresa = models.ForeignKey(Empresa, on_delete=models.CASCADE)
    nombre = models.CharField(max_length=40)
    fecha_nacimiento = models.DateField()
    # Es posible indicar un valor por defecto mediante 'default'
    antiguedad = models.IntegerField(default=0)
```

Interactuar con los modelos

- Django contiene una utilidad para interactuar con nuestra aplicación desde una consola.

```
C:\Users\Jon\Documents\dev\mysite> python manage.py shell
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
(InteractiveConsole)
>>>
```

QuerySet API

- **QuerySet API** facilita la interacción con los modelos.
- Los métodos que contiene pueden devolver un QuerySet iterable o no:
 - Devuelven QuerySet: `all()`, `filter()`, `order_by()`, `values()`, ...
 - Devuelven otra cosa: `create()`, `delete()`, `update()`, `get_or_create()`, `count()`, `first()`, ...

Interactuar con los modelos

```
from django.db import models

class Red(models.Model):
    ip = models.CharField(max_length=15)
    nombre = models.CharField(max_length=200)
    fecha_creacion = models.DateTimeField('Fecha de creacion')

class Equipo(models.Model):
    red = models.ForeignKey(Red, on_delete=models.CASCADE)
    tipo = models.CharField(max_length=30, default='')
    # Es posible indicar un valor por defecto mediante 'default'
    votes = models.IntegerField(default=0)
```


Interactuar con los modelos

```
# Importar los modelos recién creados.
>>> from myapp.models import Red, Equipo

# Consultar las redes (no hay ninguna todavía).
>>> Red.objects.all()
<QuerySet []>

# Crear una Red.
# Importamos timezone para utilizar su método now()
>>> from django.utils import timezone
>>> red = Red(ip="10.0.0.1", nombre="eduroam",
fecha_creacion=timezone.now())

# Para almacenar el objeto simplemente invocar el método save()
>>> red.save()
```

Interactuar con los modelos

```
# Django le asigna un id.
>>> q.id
1

# Acceder a sus atributos
>>> red.nombre
"eduroam"
>>> red.fecha_creacion
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)

# Actualizar los valores de los atributos y luego invocar save().
>>> red.nombre = "egibide"
>>> red.save()

# objects.all() muestra todas las redes de la base de datos.
>>> Question.objects.all()
<QuerySet [Question: Question object (1)]>

>>> redes = Red.objects.filter(ip__contains='10.0.0')
>>> redes = Red.objects.filter(nombre__contains='eduroam').count()
```

Interactuar con los modelos

- API de búsquedas

<https://docs.djangoproject.com/en/2.2/ref/models/querysets/>

- Todo sobre consultas

<https://docs.djangoproject.com/es/2.2/topics/db/queries/>

La aplicación de Administración

- Permite **crear, consultar, actualizar y borrar registros** a partir de nuestros modelos definidos.
- Muy útil para **comprobar rápidamente** todos nuestros modelos.
- Centrado en ver todos los datos sobre el modelo, **no es una interfaz pensada para usuarios finales**.
- La configuración viene hecha, solo hay que **registrar los modelos y crear un usuario administrador**.

La aplicación de Administración

1. Editar el fichero **admin.py** para registrar las clases de nuestro modelo y que se puedan ver en la aplicación de administración de Django:

```
from django.contrib import admin
from .models import Empleado, Departamento, Habilidad

# Register your models here.
admin.site.register(Empleado)
admin.site.register(Departamento)
admin.site.register(Habilidad)
```

La aplicación de Administración

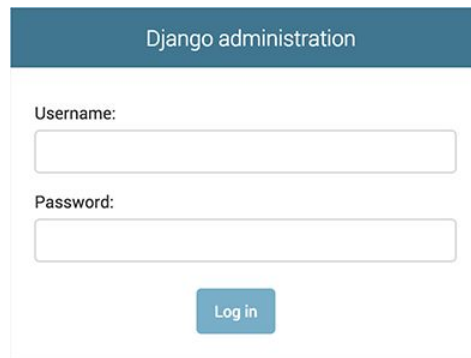
2. Crea un usuario para la aplicación de administración

```
python manage.py createsuperuser
```

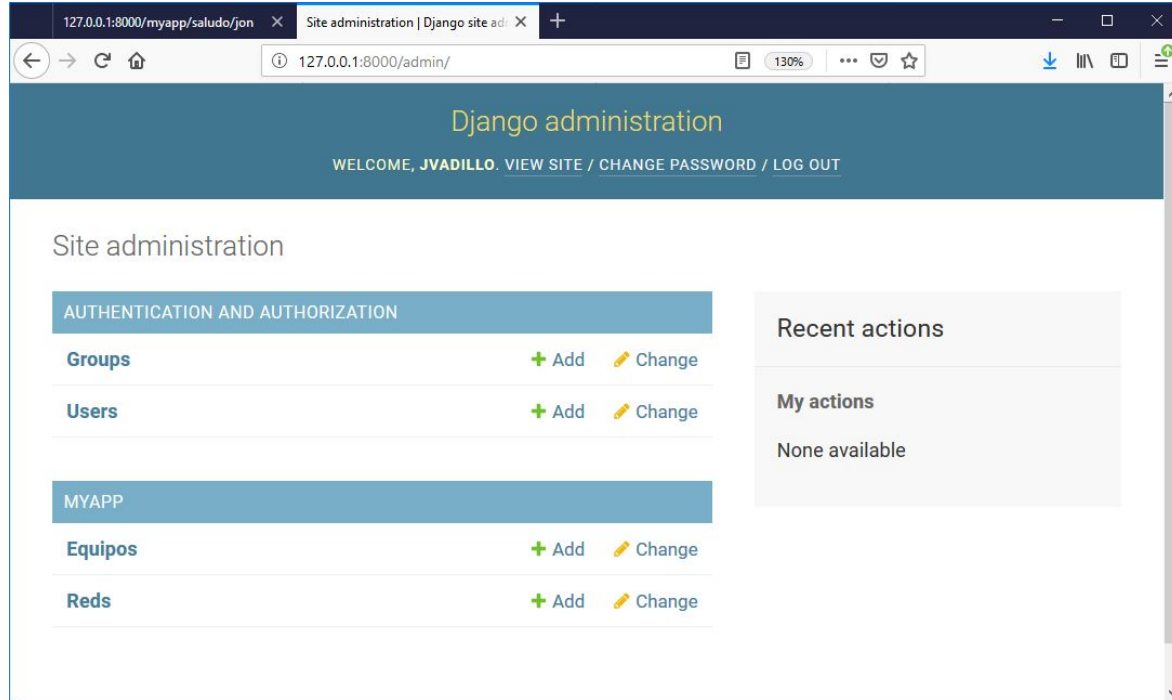
3. Iniciar el servidor y entrar

```
python manage.py runserver
```

4. Entrar en <http://127.0.0.1:8000/admin>



The image shows the Django administration login interface. It features a dark blue header with the text 'Django administration'. Below the header, there are two input fields: 'Username:' and 'Password:'. A blue 'Log in' button is positioned below the password field. The entire form is set against a light gray background.



Mejorar la usabilidad

- Añade el método `__str__(self)` a tus clases para que al consultarlas puedas identificarlas mejor:

```
class Red(models.Model):  
    # El campo en la base de datos tendrá el nombre de la variable  
    ip = models.CharField(max_length=15)  
    nombre = models.CharField(max_length=200)  
    fecha_creacion = models.DateTimeField('Fecha de creacion')  
  
    def __str__(self):  
        return self.nombre
```

Todo sobre la aplicación de administración en:

<https://docs.djangoproject.com/es/2.2/ref/contrib/admin/>

Hands on!

1. Crear el proyecto y la aplicación
2. Crear los modelos (**models.py**) Empresa y Trabajador de la diapositiva número 13.
3. Inserta, actualiza y consulta datos desde la consola
4. Utiliza la aplicación de administración para insertar datos.



<https://github.com/jvadillo/curso-django-paso-a-paso>



Hands on!



<https://github.com/jvadillo/curso-django-paso-a-paso>



Sources

- Documentación oficial: <https://www.djangoproject.com/>
- Mozilla MDN Web Docs: <https://developer.mozilla.org/>