Migraciones y modelos en Django

Primeros pasos

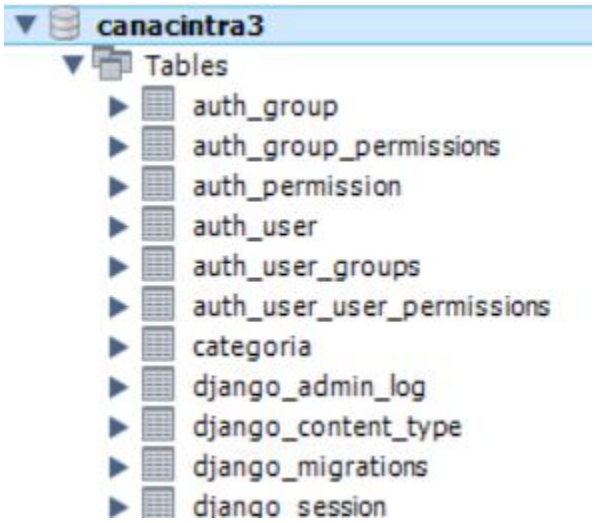
En este nuevo manual veremos como realizar modelos y migraciones para usar nuestra BD en Django, empezando por el apartado del backend, vamos a trabajar con una migración, hay que tener en cuenta que ya hemos configurado la BD en settings.py para usar mysql y no mysql lite.

Primero que nada, debemos tener una BD vacía y usar el siguiente comando en la terminal:

```
python manage.py migrate
```

```
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Esto migrara las tablas necesarias de Django a nuestra BD (categoría no incluida):



Ahora debes crearte un usuario para loguearte, ejecuta el siguiente comando en la terminal del proyecto, recuerda tener el entorno activo y estar dentro de la carpeta canacintra:

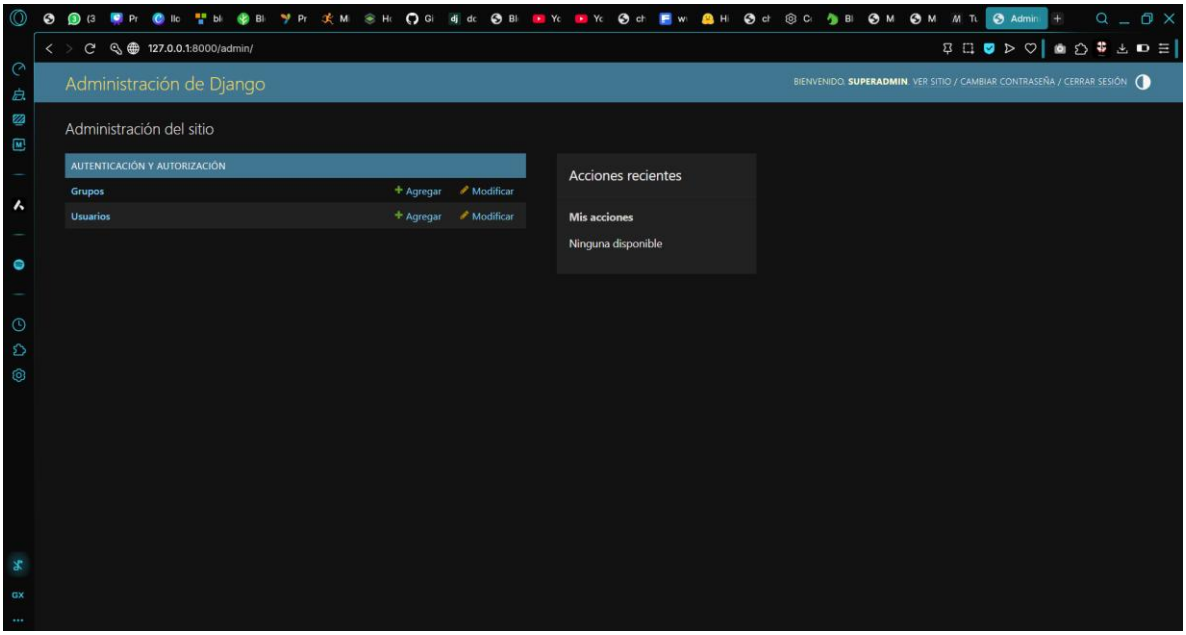
```
python manage.py createsuperuser
```

Aquí, vamos a agregar los datos que pide y una contraseña, esta no se verá al ingresarla así que habrá que tener cuidado, usamos "y" para aceptar

```
(canacintra_env) C:\proyectoWeb\canacintra>python manage.py createsuperuser
Nombre de usuario (leave blank to use 'thezt'): superadmin
Dirección de correo electrónico: superadmin@gmail.com
Password:
Password (again):
La contraseña es muy similar a  nombre de usuario.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

(canacintra_env) C:\proyectoWeb\canacintra>
```

Ahora accedemos a http://127.0.0.1:8000/admin

Iniciamos sesión y aparecerá esto:



Esta ventana contiene todo lo indispensable para llevar a cabo un control de roles y permisos

Modelos en django:

Creando la tabla de "Categoria"

En Django, un modelo es una clase de Python que define la estructura de una tabla en la base de datos. Cada atributo de esa clase se convierte en una columna, y cada instancia en un registro (fila).

Estructura básica de un modelo

```
from django.db import models


class NombreDelModelo(models.Model):
    campo1 = models.TipoDeCampo(opciones)
    campo2 = models.TipoDeCampo(opciones)


    class Meta:
        db_table = 'nombre_tabla_en_bd'
        verbose_name_plural = 'Nombre legible en admin (opcional)'
```

Vamos a crear un modelo llamado Categoria. Supongamos que queremos registrar un nombre para la categoría y vincularla a un usuario del sistema.

Primero, importamos el modelo de usuario de Django con:

```
from django.contrib.auth.models import User
```

en caso de que no haya un id borra la consulta con

```
fk_user = models.ForeignKey(User, on_delete= models.SET_NULL, null
```

Luego, definimos el modelo Categoria:

Django por defecto tiene auto-increment en el área de usuarios, en esta ocasión lo vamos a modificar de esta forma:

```
id = models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')
```

Esto crea un campo de clave primaria con incremento automático. Se usa cuando se espera que la tabla crezca mucho y se necesitan más valores que los que proporciona un AutoField.

Para crear un campo se define:

```
campo1 = models.TipoDeCampo(opciones)
```

Por ejemplo:

```
nombre = models.CharField(max_length=60)
```

Al nivel de los datos de la tabla se agrega

```
class Meta:
        managed = True
        db_table = 'categoria'
        verbose_name_plural = 'Categoria'
```

para agregar el nombre de la tabla

El código completo seria:

```python
class Categoria(models.Model):
    id = models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')
    nombre = models.CharField(max_length=60)
    createdat = models.DateTimeField(auto_now_add=True)
    updatedat = models.DateTimeField(null=True, blank=True)
    fk_user = models.ForeignKey(User, on_delete= models.SET_NULL, null = True)

    class Meta:
        managed = True
        db_table = 'categoria'
        verbose_name_plural = 'Categoria'
```

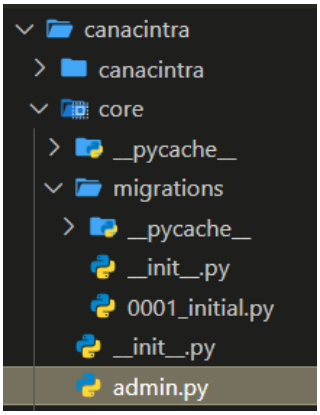
- **nombre**: Campo de texto con un máximo de 60 caracteres. Guarda el nombre de la categoría.
- **createdat**: *DateTimeField(auto_now_add=True)* guarda automáticamente la fecha y hora cuando se crea el registro.
- **updatedat**: DateTimeField(null=True, blank=True) permite almacenar la fecha de la última actualización. Este campo lo puedes actualizar manualmente cuando edites el objeto.

Opciones de configuración (clase Meta):

- **managed = True**: Indica que Django debe encargarse de crear, actualizar y eliminar esta tabla durante las migraciones.
- **db_table = 'categoria'**: Define el nombre exacto de la tabla en la base de datos.
- **verbose_name_plural = 'Categoria'**: Nombre que se mostrará en plural en el panel de administración de Django. (Aunque lo correcto en español sería 'Categorías').

Registro del modelo Categoria en el panel de administración de Django

Para que la tabla Categoría sea visible y administrable desde el Django Admin, necesitas registrarla en el archivo admin.py de tu aplicación.



Usamos el siguiente código para la tabla categoría:

```
from django.contrib import admin

from core.models import Categoria

# Register your models here.

class CategoriaAdmin(admin.ModelAdmin):

    list_display = ['id', 'nombre', 'createdat', 'updatedat',
'fk_user']

admin.site.register(Categoria, CategoriaAdmin)
```

Este código muestra los campos id, nombre, createdat, updatedat, y fk_user en la lista del admin.

Migración de la base de datos

Una vez definido el modelo y registrado en el admin, debes crear y aplicar la migración para que la tabla se cree en la base de datos con los siguientes comandos:

```
python manage.py makemigrations
```

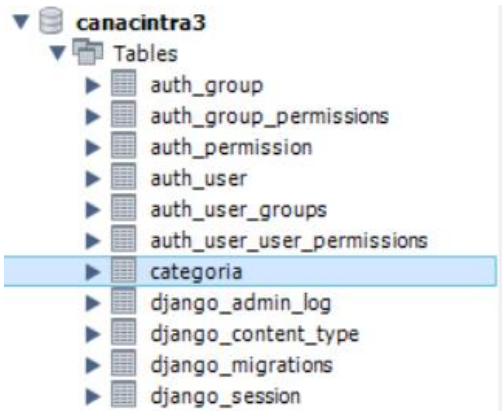
para cargar la tabla y

```
python .\manage.py  migrate
```

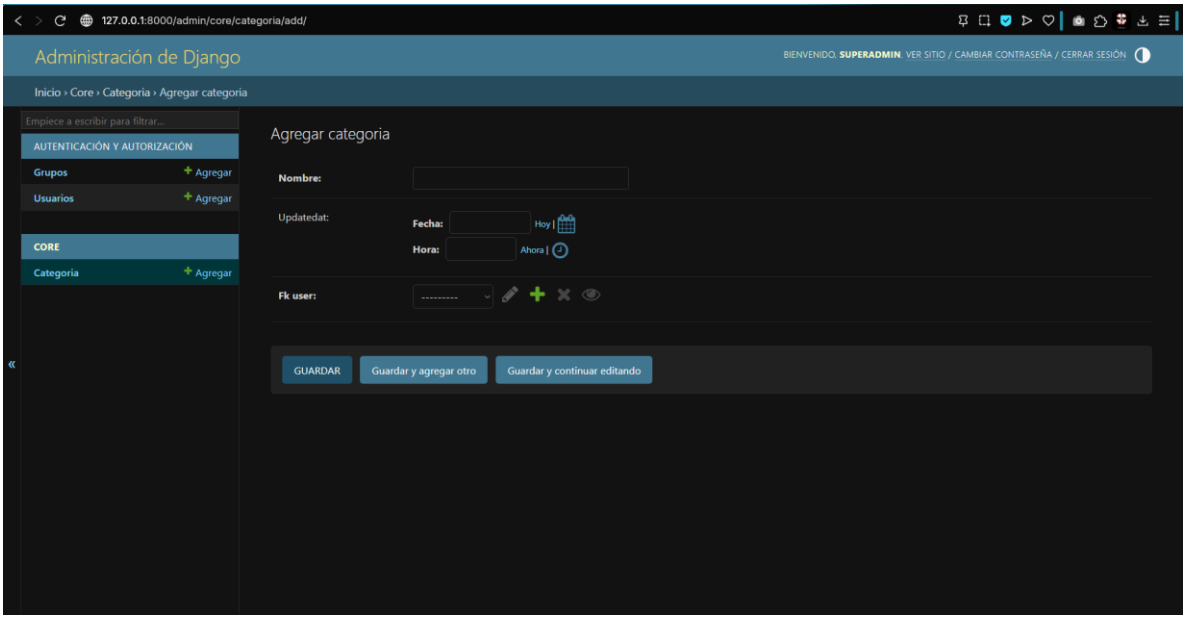
para ejecutar la migración

```
(canacintra_env) C:\proyectoWeb\canacintra>python manage.py makemigrations core
Migrations for 'core':
  core\migrations\0001_initial.py
    + Create model Categoria

(canacintra_env) C:\proyectoWeb\canacintra>python manage.py migrate
System check identified some issues:

WARNINGS:
?: (mysql.W002) MySQL Strict Mode is not set for database connection 'default'
        HINT: MySQL's Strict Mode fixes many data integrity problems in MySQL, suc
s strongly recommended you activate it. See: https://docs.djangoproject.com/en/5.2
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying core.0001_initial... OK
```

Y ya podrás ver tu tabla dentro de tu BD:



Así también dentro de la ventana del admin:



Ahora debemos realizar el resto de los modelos para las tablas de canacintra.

Generación de modelos

Teniendo una BD completa podemos generar los modelos mediante el uso del siguiente comando:

```
python .\manage.py inspect > modelejemplo.py
```

Este comando generara todos los modelos de la BD si ya tiene tablas creadas y las guardara en un archivo, se recomienda hacer esto en un proyecto a parte para evitar conflictos entre migraciones, nos generara un archivo de este tipo:

modelejemplo.py

```python
# This is an auto-generated Django model module.
# You'll have to do the following manually to clean this up:
#   * Rearrange models' order
#   * Make sure each model has one field with primary_key=True
#   * Make sure each ForeignKey and OneToOneField has `on_delete`
#   * Remove `managed = False` lines if you wish to allow Django t
# Feel free to rename the models, but don't rename db_table values
from django.db import models


class Archivo(models.Model):
    nombre = models.CharField(max_length=255)
    nombre_temporal = models.CharField(max_length=255)
    ruta = models.CharField(max_length=255)
    tipo = models.CharField(max_length=100)
    tamano = models.IntegerField(blank=True, null=True)
    descripcion_corta = models.CharField(max_length=150, blank=Tru
    descripcion_larga = models.TextField(blank=True, null=True)
    descargas = models.IntegerField(blank=True, null=True)
    created = models.DateTimeField()
    updated = models.DateTimeField(blank=True, null=True)
    fk_user = models.ForeignKey('User', models.DO_NOTHING, db_colu

    class Meta:
        managed = False
        db_table = 'archivo'
```

Errores al migrar

En caso de tener problemas y que las tablas no se muestren en la base de datos a pesar de haberlas migrado varias veces y veas este error

```
(canacintra_env) C:\proyectoWeb\canacintra>python manage.py migrate
System check identified some issues:

WARNINGS:
?: (mysql.W002) MySQL Strict Mode is not set for database connection 'default'
        HINT: MySQL's Strict Mode fixes many data integrity problems in MySQL, s
s strongly recommended you activate it. See: https://docs.djangoproject.com/en/5
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  No migrations to apply.
```

Puedes hacer lo siguiente, cabe aclarar que se perderan todos los datos de las tablas que hayas migrado anteriormente:

El aviso sobre MySQL Strict Mode es solo una advertencia (warning) y no impide que las migraciones se apliquen.

Sin embargo, el problema clave es:

No migrations to apply.

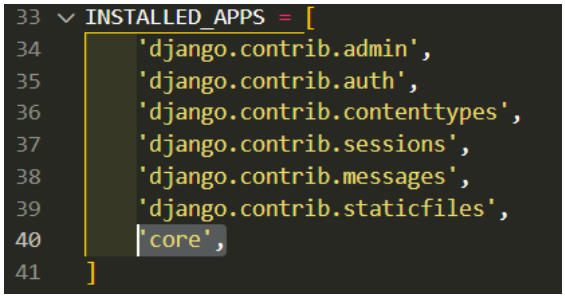
Eso significa que Django no detecta ningún cambio pendiente en tu app core, a pesar de que tienes el modelo Categoria.

Verificación rápida

1. ¿Tu archivo core/migrations/__init__.py existe?

Debe existir ese archivo, aunque esté vacío. Si no está, Django no reconocerá la carpeta como válida para migraciones.

2. ¿La app core está registrada en INSTALLED_APPS dentro del archivo settings.py?



3. Verifica que tu modelo no haya sido ya creado manualmente en la base de datos (pero sin registrar en Django)

Si tú creaste la tabla categoria manualmente en MySQL, y nunca hiciste una migración desde Django, entonces:

- Django no tiene registro en su historial de migraciones.
- No la recreará ni la detectará como nueva.

Solución completa: Borrar historial y migrar desde cero (modo fuerte)

1. Elimina migraciones viejas del sistema

Borra todo lo que empiece con 00 o similares en core/migrations/:

En una terminal tipo cmd dentro del proyecto accedemos a canacintra y ejecutamos el siguiente comando

```
del core\migrations\0*.py
```

```
(canacintra_env) C:\proyectoWeb\canacintra>del core\migrations\0*.py
```

2. Borra el historial de migraciones en MySQL (opcional pero recomendado)

Entra a tu base de datos MySQL (usa phpMyAdmin o consola) y ejecuta:

```
DELETE FROM django_migrations WHERE app = 'core';
```

Si en la BD el query suelta esto:

Error Code: 1175. You are using safe update mode and you tried to…

Debes desactivar el modo Seguro de mysql, hazlo con este comando:

```
SET SQL_SAFE_UPDATES = 0;
```

3. Ahora crea la migración de nuevo con:

```
python manage.py makemigrations core
```

4. Y luego aplícala:

```
python manage.py migrate
```

Esto debería de ser suficiente para que la migración sea aplicada:

```
(canacintra_env) C:\proyectoWeb\canacintra>python manage.py makemigrations core
Migrations for 'core':
  core\migrations\0001_initial.py
    + Create model Categoria

(canacintra_env) C:\proyectoWeb\canacintra>python manage.py migrate
System check identified some issues:

WARNINGS:
?: (mysql.W002) MySQL Strict Mode is not set for database connection 'default'
        HINT: MySQL's Strict Mode fixes many data integrity problems in MySQL, suc
s strongly recommended you activate it. See: https://docs.djangoproject.com/en/5.2
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying core.0001_initial... OK
```