Administración

Para acceder a la zona de administración de Django lo haremos con este enlace : http://127.0.0.1:8000/admin/

Para crear un usuario, antes hay que realizar la instrucción de las migraciones de aviso que nos daba al iniciar el servidor, si no, nos da error

python manage.py mígrate

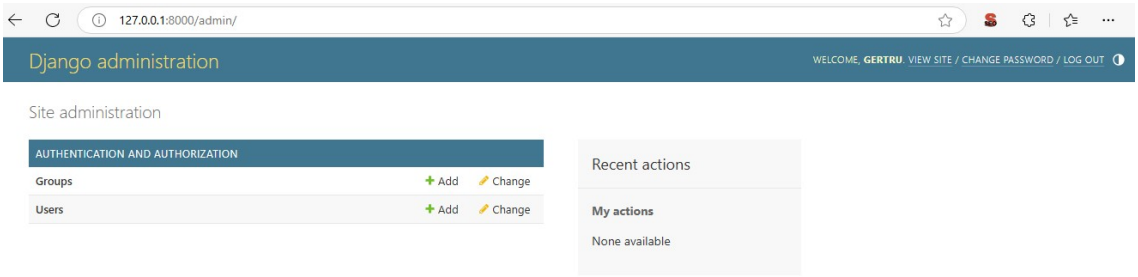
Hay que crear un super usuario.

Python manage.py createsuperuser

Luego hay que volver a iniciar el servidor

Python manage.py runserver

Accederemos al panel de esta manera.



Vemos una de las aplicaciones (en Django aplicaciones son abstracciones de funcionalidades) que vienen dentro de django, Authenticacion y Authorizacion. Recordar que Django está formado por distintas aplicaciones que son autónomas pero que también funcionan relacionadas entre sí. Este es un poco el potencial de este Framework.

Aparecen aquí los usuarios y grupos por defecto que tiene Django

Este sitio de administración de Django nos permitirá crear Cruds de forma automática (create, registrar, update, delete), en definitiva gestionar los registros de la BD

Modelos

Las tablas dentro de Django se van a llamar modelos, porque van a ser simbolizadas mediantes clases, que dentro del mundo de Django serán modelos.

El sitio de administración de Django nos va permitir la gestión de estas tablas o modelos

Las tablas que Django trae por defecto se almacenan en la db.Sqlite3 (se crea con ese nombre puesto que en el setting.py así se especifica):

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Podemos abrir esa db.Sqlite3 con el BD Browser for Sqlite, y poder así echar un vistazo

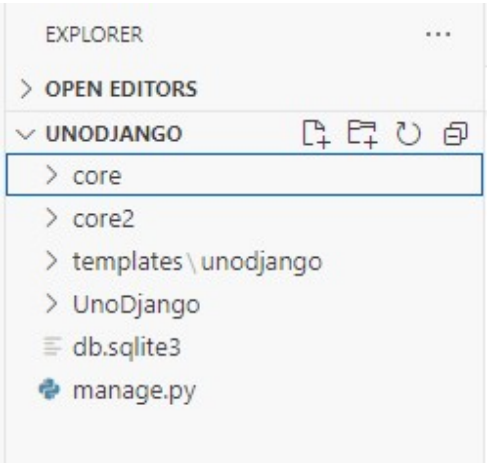
Cada aplicación que tengamos, tendrá su lógica de modelo de tablas. Si queremos crear una nueva aplicación para introducir otra lógica, como ya se ha expuesto anteriormente, tendremos que ejecutar el comando siguiente si tenemos acceso al manage.py

python manage.py startapp nombre_aplicación

Si no tenemos acceso al manage.py, la instrucción para crear una nueva aplicación sería similar a la que hemos utilizado para crear el proyecto.

django-admin startapp nombre_aplicación

La estructura quedaría de la siguiente manera, siendo los nombre de las aplicaciones creadas con las instrucciones anteriores core y core2:



Dentro de la carpeta templates, tendremos subcarpetas con el nombre de cada proyecto, en las que almacenaremos los .html o lo necesario para cada uno de ellos.

Cada aplicación dentro de Django tiene la misma estructura, creada con una finalidad, entre otras cosas con la finalidad de seguir un orden y un standard.

En toda aplicación vamos a tener una carpeta de migraciones en la que tendremos todos los archivos de migraciones generados automáticamente. Los cambios que vayamos haciendo en los modelos/tablas se irán creando en esa carpeta de migración.

El archivo _init_.py, el interprete de python es al primero que llama para tomar en cuenta en la ejecución todo lo que se indique en ese archivo. Tanto las carpetas de aplicaciones (core y core2) como la carpeta principal (UnoDjango) tienen este archivo.

El archivo admin.py, nos servirá para registrar nuestros modelos. Hay que generar CRUDs, para los modelos que nosotros elijamos.

El archivo apps.py, tiene un import de AppConfig. Es una clase con diferentes métodos para que internamente dentro de Django reconozca una carpeta como aplicación.

```
from django.apps import AppConfig
class CoreConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField' #tipo de dato de los índices será Big, para que pueda autoincrementarse hasta dónde se necesite.
    name = 'core'  #ruta donde se encuentra la carpeta core. Si esta carpeta estuviera dentro de una aplicación llamada app, el name='app/core'
```

El archivo models.py, en el se crearán los modelos previamente registrados en admin.py, que pertenezcan a la aplicación dónde se encuentre ese archivo.

El archivo tests.py, se utiliza para establecer las pruebas unitarias de nuestras vistas de la aplicación. Es una sugerencia de Django.

El archivo views.py, en él crearemos la lógica de la aplicación. Obviamente si el proyecto comienza a crecer toda esta lógica pude sustituirse por varios archivos que pueden estar dentro de una carpeta.

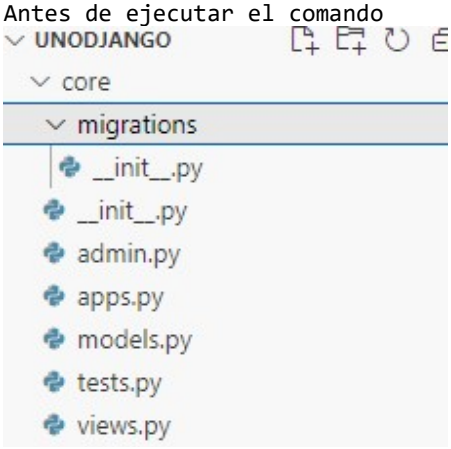
Centrándonos en el fichero models.py, éste importa models, que no es más que un conjunto de imports, clases, variables necesarias para crear nuestros modelos/tablas. Éstos estarán representados por clases. Django realiza esta abstracción para llevarlos a la BD a través de un traductor que toma la estructura de éstos modelos y los convierte a SQL, que se ejecutaran en la BD. Con lo cual en este fichero no tendremos código SQL, sino python. Todo irá a la BD mediante herramientas internas que tiene incorporado el Framework (intérpretes, ORM, etc).
En el fichero models.py, nos encontraremos con la importación de models. Las clases que creamos van a heredar de models, para que django sepa que son tablas de la BD. Django lo reconoce como un modelo, y en django cualquier modelo es una tabla.
Iremos construyendo el modelo de esta forma:

```
from django.db import models

# Create your models here.
# Al aplicar esta herencia, Django va a saber que Author es una tabla en la BD
class Author (models.Model):
    name=models.CharField (verbose_name='Nombre', # etiqueta dentro de la tabla
    max_length= 100,
    default=''
    )
    last_name=models.CharField(verbose_name='Apellido',
    max_length=150,
    default='')
    age=models.PositiveSmallIntegerField (verbose_name='Edad',
    )
```

Antes de poder llevar el modelo a la BD, hay que decirle a Django, que la nueva aplicación existe. En settings.py de la aplicación principal, hay que incluir core.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    #añadimos la nueva aplicación
    'core',
]
```

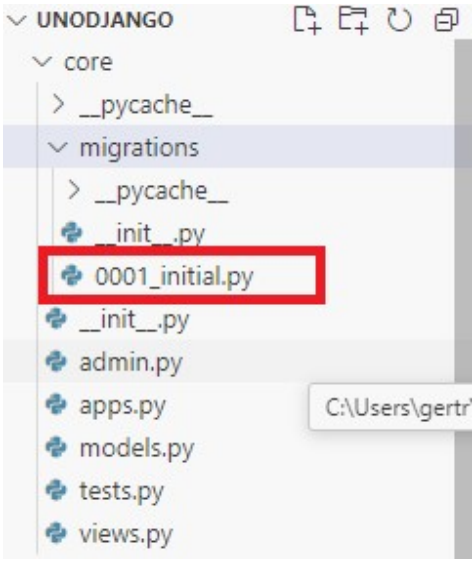
Para que todo esto se lleve a la BD, hay que realizar dos instrucciones, la primera para generar el scripts de los nuevos elementos o modificaciones de los que hubiera:
Antes de ejecutar el comando



python manage.py makemigrations

```
PS C:\Users\gertr\workspace\unodjango> python manage.py makemigrations
Migrations for 'core':
  core\migrations\0001_initial.py
    - Create model Author
PS C:\Users\gertr\workspace\unodjango>
```

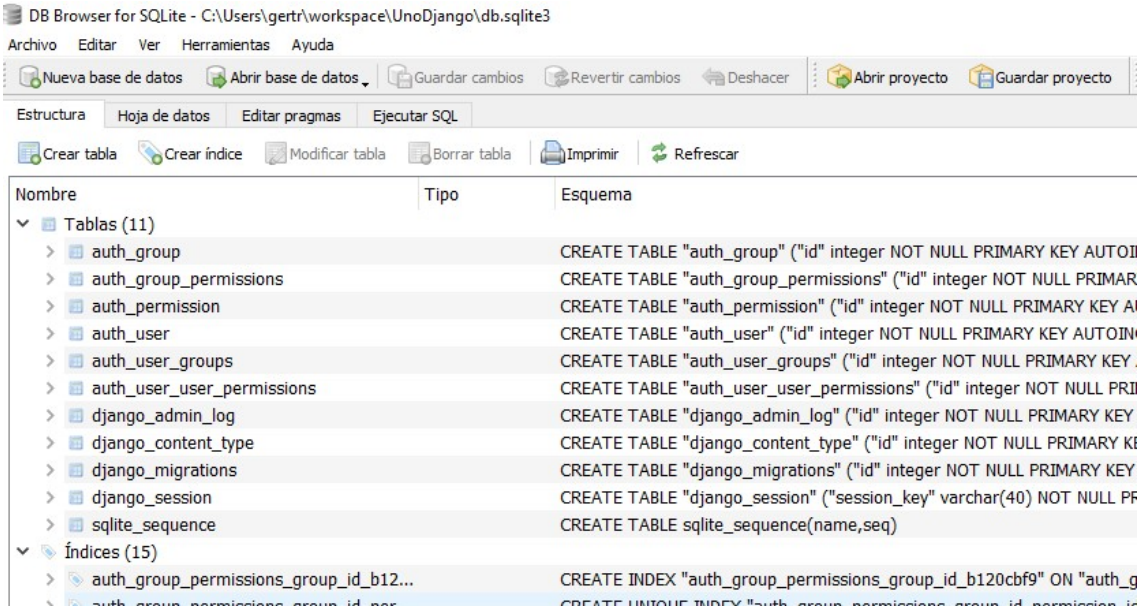
Después de ejecutar el comando:



Se genera un nuevo archivo, en el que aparece el código necesario para crear el modelo/tabla Author. El contenido de ese archivo 0001_initial.py sería el siguiente:

```
from django.db import migrations, models
class Migration(migrations.Migration):
    initial = True
    dependencies = [
    ]
    operations = [
        migrations.CreateModel(
            name='Author',
            fields=[ #como se observa, la id es creada automáticamente, pues nosotros
       en el models.py no habíamos especificado ninguna clave. Puede observarse que tiene
       auto_created a TRUE, indicación para aclarar que no se especificó  en el modelo
       original.
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
                ('name', models.CharField(default='', max_length=100,
verbose_name='Nombre')),
                ('last_name', models.CharField(default='', max_length=150,
verbose_name='Apellido')),
                ('age', models.PositiveSmallIntegerField(verbose_name='Edad')),
            ],
        ),
    ]
```

Si observamos en nuestra db.sqlite3 del proyecto, el modelo/tabla Author no está creada:



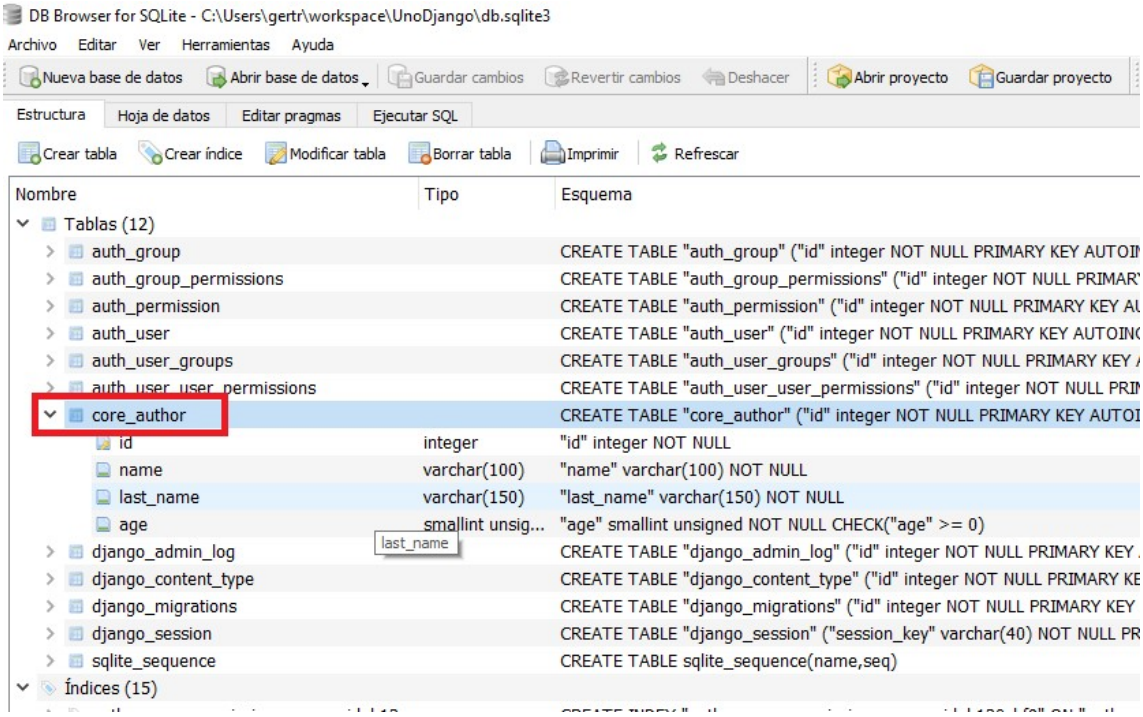
Se ha generado el script de creación de la tabla pero aún no se ha ejecutado, ese es el motivo por el que aún no está creada. Para que se ejecute, hay que realizar otra instrucción:

Python manage.py migrate

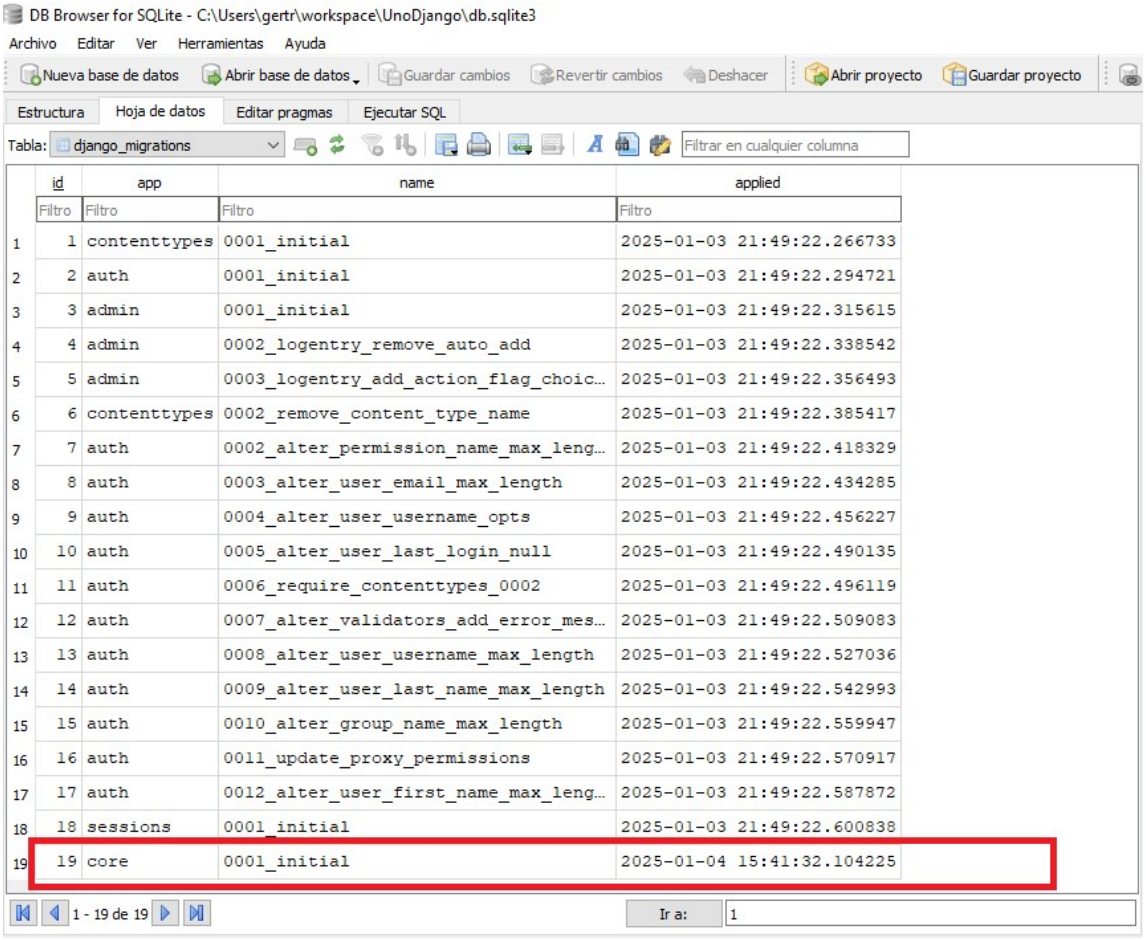
```
PS C:\Users\gertr\workspace\unodjango> Python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying core.0001_initial... OK
PS C:\Users\gertr\workspace\unodjango>
```

Puede observarse en el mensaje, que aplica todas las migraciones para las aplicaciones admin, auth, contenttypes, core y sessions, que son todas las que tenemos en el fichero settings.py dentro de la lista INSTALLED_APPS.

Ahora si nos vamos a BD, tendremos la nueva tabla:



Estos scripts, Django lleva un registro de cuales han sido aplicados. Se encuentra en la información de la tabla django_migrations



	id	app	name	applied
	Filtro	Filtro	Filtro	Filtro
1	1	contenttypes	0001_initial	2025-01-03 21:49:22.266733
2	2	auth	0001_initial	2025-01-03 21:49:22.294721
3	3	admin	0001_initial	2025-01-03 21:49:22.315615
4	4	admin	0002_logentry_remove_auto_add	2025-01-03 21:49:22.338542
5	5	admin	0003_logentry_add_action_flag_choic…	2025-01-03 21:49:22.356493
6	6	contenttypes	0002_remove_content_type_name	2025-01-03 21:49:22.385417
7	7	auth	0002_alter_permission_name_max_leng…	2025-01-03 21:49:22.418329
8	8	auth	0003_alter_user_email_max_length	2025-01-03 21:49:22.434285
9	9	auth	0004_alter_user_username_opts	2025-01-03 21:49:22.456227
10	10	auth	0005_alter_user_last_login_null	2025-01-03 21:49:22.490135
11	11	auth	0006_require_contenttypes_0002	2025-01-03 21:49:22.496119
12	12	auth	0007_alter_validators_add_error_mes…	2025-01-03 21:49:22.509083
13	13	auth	0008_alter_user_username_max_length	2025-01-03 21:49:22.527036
14	14	auth	0009_alter_user_last_name_max_length	2025-01-03 21:49:22.542993
15	15	auth	0010_alter_group_name_max_length	2025-01-03 21:49:22.559947
16	16	auth	0011_update_proxy_permissions	2025-01-03 21:49:22.570917
17	17	auth	0012_alter_user_first_name_max_leng…	2025-01-03 21:49:22.587872
18	18	sessions	0001_initial	2025-01-03 21:49:22.600838
19	19	core	0001_initial	2025-01-04 15:41:32.104225