



DESARROLLO CON PYTHON

Contenido de formación



Programación



Pyhton



Paradigma POO



Bases de datos SQL



Tkinter



Django



Flask

1. Primeros Instalación de SW
2. Variables y tipos de datos
3. Operadores (aritméticos, asignación)
4. Entrada y salida de datos
5. Estructuras de control
 - 5.1 Condicionales (operadores lógicos, operadores de comparación)
 - 5.2 Bucles (estructuras interactivas for, while)
6. Bloque de ejercicios de aplicación de conceptos
7. Funciones (parámetros, return, invocación, lambda)
 - 7.1 Variables locales y globales, funciones y métodos predefinidos
8. Lista y tuplas (creación, índices, recorrer y mostrar listas, listas multidimensionales)
9. Diccionarios y sets
10. Bloque de ejercicios aplicación de conceptos
11. Módulos y paquetes (creación y funcionalidad)
12. Sistemas de archivos y directorios
13. Manejo de errores (captura de excepciones, errores personalizados)
14. Programación orientada a objetos
15. Bases de datos SQLite
16. Bases de datos MySQL
17. Proyecto con Python
18. Interfaces graficas con Tkinter (aplicación de escritorio Python – Tkinter)
19. Desarrollo Web con Django
20. Interfaces graficas con Flask (aplicación de escritorio Python – Flask)



Modelos y bases de datos en Django

Pylint para Django, sirve para reconocer mejor el código de Django para evitar errores. Desde la consola digitar el siguiente código:

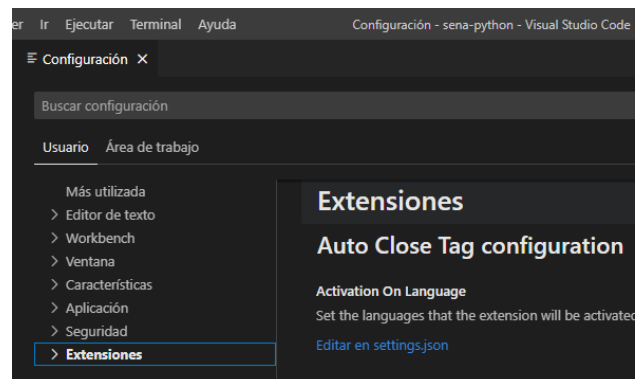
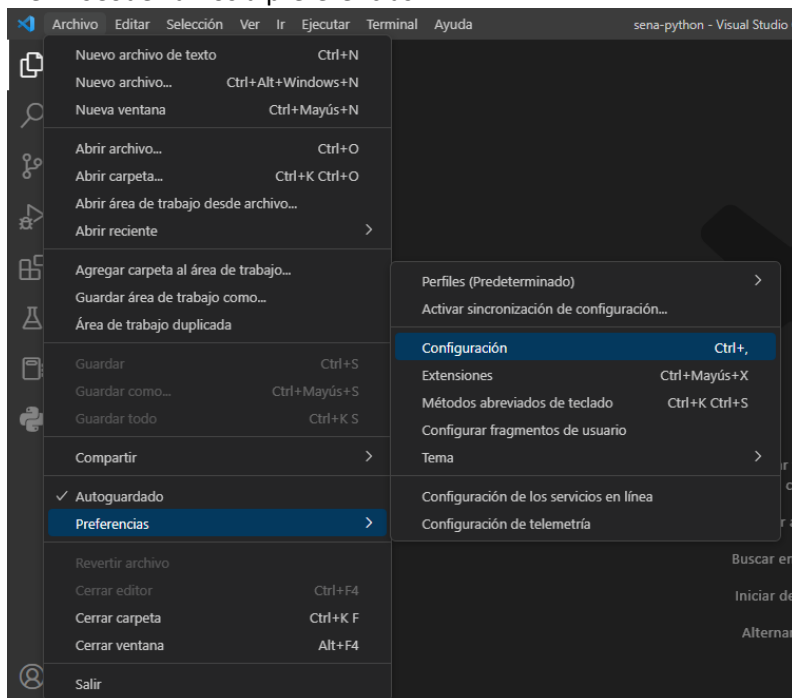
pip install pylint-django

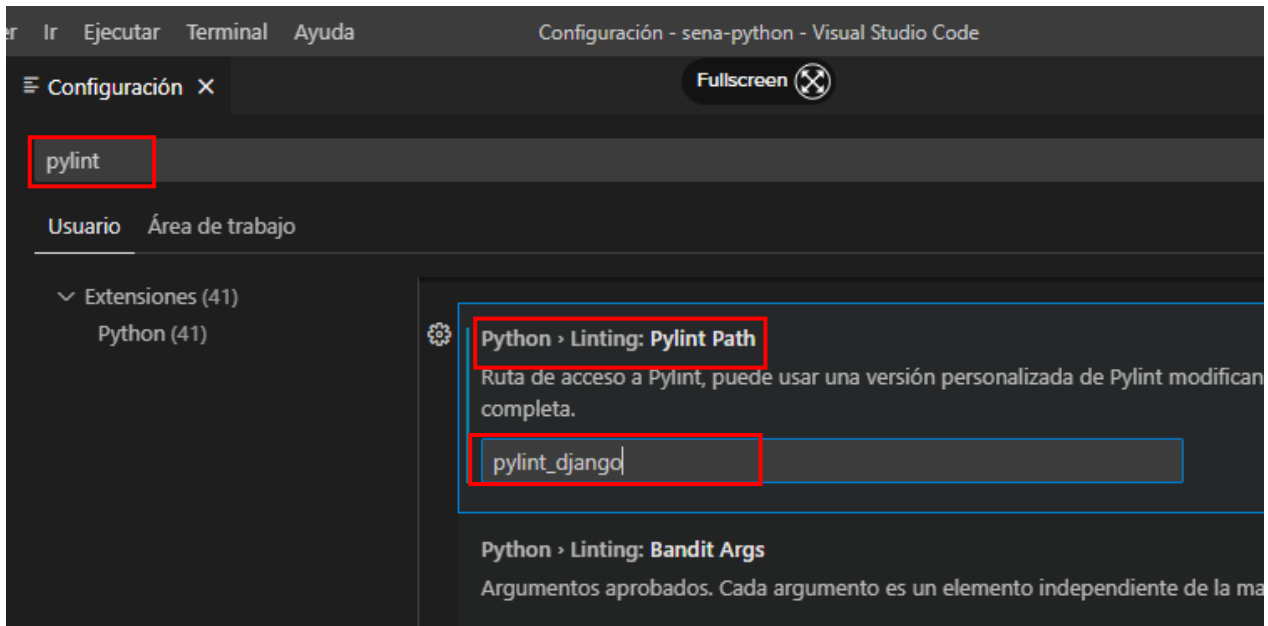
```
C:\Users\MGV>pip install pylint-django
Collecting pylint-django
  Downloading pylint_django-2.5.3-py3-none-any.whl (81 kB)
----- 81.2/81.2 kB 2.3 MB/s eta 0:00:00
Collecting pylint-plugin-utils>=0.7
  Downloading pylint_plugin_utils-0.7-py3-none-any.whl (10 kB)
Collecting pylint<3,>=2.0
  Downloading pylint-2.16.2-py3-none-any.whl (530 kB)
----- 530.7/530.7 kB 2.0 MB/s eta 0:00:00
Collecting platformdirs>=2.2.0
  Downloading platformdirs-3.0.0-py3-none-any.whl (14 kB)
Collecting astroid<=2.16.0-dev0,>=2.14.2
  Downloading astroid-2.14.2-py3-none-any.whl (273 kB)
----- 273.0/273.0 kB 2.4 MB/s eta 0:00:00
Collecting isort<6,>=4.2.5
  Downloading isort-5.12.0-py3-none-any.whl (91 kB)
----- 91.2/91.2 kB 2.6 MB/s eta 0:00:00
Collecting mccabe<0.8,>=0.6
  Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Collecting tomlkit>=0.10.1
  Downloading tomlkit-0.11.6-py3-none-any.whl (35 kB)
Collecting dill>=0.3.6
  Downloading dill-0.3.6-py3-none-any.whl (110 kB)
----- 110.5/110.5 kB 3.2 MB/s eta 0:00:00
Collecting colorama>=0.4.5
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting lazy-object-proxy>=1.4.0
  Downloading lazy_object_proxy-1.9.0-cp311-win_amd64.whl (22 kB)
Collecting wrapt<2,>=1.14
  Downloading wrapt-1.14.1.tar.gz (50 kB)
----- 50.9/50.9 kB 2.7 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
Installing collected packages: wrapt, tomlkit, platformdirs, mccabe, lazy-object-proxy, colorama, astroid, pylint, pylint-plugin-utils, pylint-django
DEPRECATION: wrapt is being installed using the legacy 'setup.py install' method, but will soon require a 'pyproject.toml' and the 'wheel' package is not installed. pip 23.1 will enforce this change. A possible replacement is to enable the '--use-pep517' option. Discussion: https://github.com/pypa/pip/issues/8559
Running setup.py install for wrapt ... done
Successfully installed astroid-2.14.2 colorama-0.4.6 dill-0.3.6 isort-5.12.0 lazy-object-proxy-1.9.0 mccabe-0.7.0 platformdirs-3.0.0 pylint-2.16.2 pylint-django-2.5.3 pylint-plugin-utils-0.7 wrapt-1.14.1

[notice] A new release of pip available: 22.3.1 -> 23.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip

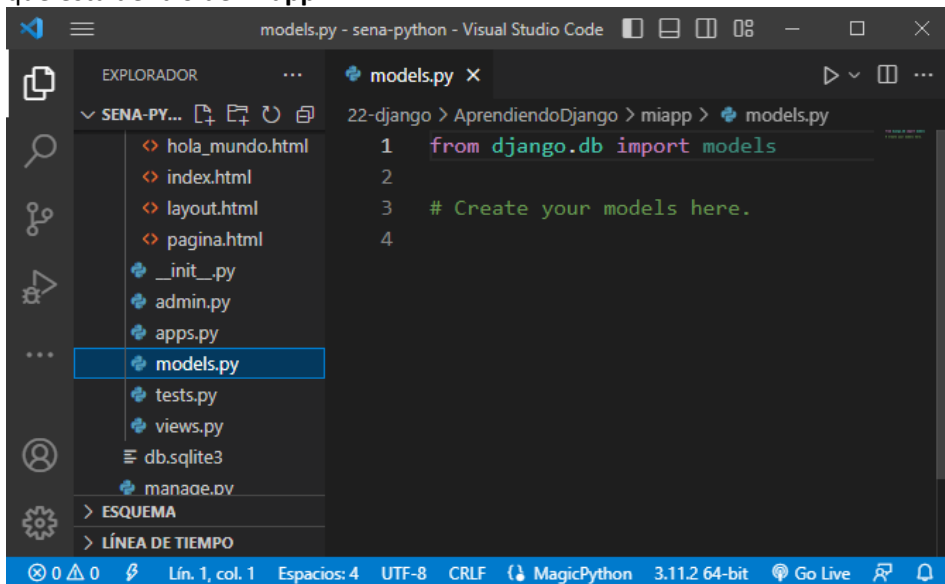
C:\Users\MGV>
```

Y en VsCode vamos a preferencias:





Crear modelos y entidades dentro del modelo vista controlador para interactuar con la base de datos vamos a **models.py** que esta dentro de **miapp**



El modelo define las tablas y campos con sus propiedades que estarán en la base de datos:

Tipos de datos en Django

Toda la información es extraída desde la **documentación oficial de Django 3**. [Model field reference](#)

models.CharField:

Utilizado para cadenas (String), puedes especificar cadenas pequeñas o grandes, mediante el parámetro max-length
Parámetros:

max-length: Espera un número entero, que indica el tamaño de la cadena.

blank: Espera un valor Booleano, que indica si el campo puede recibir valores vacíos o no.

choices: Espera un arreglo de String, espera un array de parejas, el cual será las únicas opciones para un valor en este campo.





```
class Autor(models.Model):
    full_name = models.CharField(max_length=60)
    adresse = models.CharField(max_length=80, blank=True)
    gender = models.CharField(max_length=80, choices=array_choices)
```

models.BooleanField:

Utilizado para especificar campos de tipo booleano (Verdadero o Falso)

Parámetros:

blank = Especifica si este campo puede o no ser vacío.

default = Especifica el valor por defecto si se intenta ingresar un valor vacío.

```
class Libro(models.Model):
    titulo = models.CharField(max_length=60)
    publicado = models.BooleanField(default=False)
```

models.DateField:

Utilizado para campos que requieran el registro de fechas (objetos de tipo fecha sin hora)

Parámetros:

null: se especifica valor booleano para indicar si puede o no ser vacío.

auto_now: valor booleano que indica que si la fecha se completa automáticamente basándose en el servidor, requiere configuración adicional.

```
class Libro(models.Model):
    titulo = models.CharField(max_length=60)
    publicado = models.BooleanField(default=False)
    fecha_publicacion = models.DateField(null=True)
```

models.DateTimeField:

A diferencia del DateField este aparte registrar la fecha, también registra la hora y trae procesos y métodos extra para manipular los datos de tipo hora.

Parámetros:

null: se especifica valor booleano para indicar si puede o no ser vacío.

auto_now: valor booleano que indica que si la fecha se completa automáticamente basándose en el servidor, requiere configuración adicional.

```
class Libro(models.Model):
    titulo = models.CharField(max_length=60)
    publicado = models.BooleanField(default=False)
    fecha_publicacion = models.DateTimeField(null=True, blank=True)
```

models.EmailField:

Como ya da a suponer el nombre, es un campo exclusivo para cadenas de tipo email, incluye validaciones extra para que solo acepte cadenas con un correcto formato de email. No necesita especificar el tamaño máximo de la cadena.

Parámetros:

blank: se pone en True si deseas que este campo no sea obligatorio

```
class Autor(models.Model):
    full_name = models.CharField(max_length=60)
    correo = models.EmailField(blank=True)
```





models.URLField:

Muy similar al EmailField, recibe una cadena que tenga el formato correcto de una url válida. No es necesario especificar el tamaño máximo de la cadena.

Parámetros:

null: Internamente esto se transforma en un objeto, por ello para que no sea obligatorio se deberá poner la propiedad null=True

```
class Autor(models.Model):
    full_name = models.CharField(max_length=60)
    correo = models.EmailField(blank=True)
    perfil = models.URLField(null=True)
```

models.ImageField:

Se utiliza cuando se desea subir y almacenar imágenes en un servidor, al procesar este atributo se creará un objeto que luego puedes manipular con facilidad, accediendo a la url de la imagen, el tamaño, ruta y otros datos más.

Parámetros:

null: recibe Booleano

blank: recibe Booleano

upload: recibe cadena, ahí puedes especificar el nombre de la carpeta donde deseas que se guarden las imágenes.

NOTA: puedes aplicar parámetros y funciones especial al guardar para procesar imágenes, un ejemplo de ello sería: si deseas comprimir el peso de una imagen para no sobrecargar tu servidor.

```
class Autor(models.Model):
    full_name = models.CharField(max_length=60)
    correo = models.EmailField(blank=True)
    perfil = models.URLField(null=True)
    foto = models.ImageField(null=True, upload='fotos')
```

models.PositiveIntegerField:

Parte del tipo de campo IntegerField, con la diferencia que este solo permitirá almacenar valores positivos enteros. Se puede usar por ejemplo en campos donde almacenes edad.

Parámetros:

default: puedes asignar un valor por defecto o nulo si no deseas que sea obligatorio.

```
class Autor(models.Model):
    full_name = models.CharField(max_length=60)
    correo = models.EmailField(blank=True)
    perfil = models.URLField(null=True)
    foto = models.ImageField(null=True, upload='fotos')
    edad = models.PositiveIntegerField(default=0)
```

Aquí se encuentran todos los que se pueden configurar en Django:

<https://docs.djangoproject.com/en/4.1/ref/models/fields/>



Configuración de los modelos que son los objetos – entidades que representan una tabla en la base de datos y sus propiedades que representan cada una de las columnas de BD:

```
1 from django.db import models
2
3 # Create your models here.
4 #colocar los nombres de las entidades en singular
5
6 class Article(models.Model): #Entidad
7
8     #propiedades *** Documentar cada tipo de dato
9     title = models.CharField(max_length=100)
10    content = models.TextField()
11    public = models.BooleanField()
12    created_at = models.DateTimeField(auto_now_add=True)
13    updated_at = models.DateTimeField(auto_now=True)
14
15 class Category(models.Model): #Entidad
16    name = models.CharField(max_length=100)
17    description = models.CharField(max_length=250)
18    created_at = models.DateField()
```

Crear tablas basadas en modelos:

1. Abrir los settings.py y confirmar que app este dentro de INSTALLED_APPS

```
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'miapp',
41 ]
42
43 MIDDLEWARE = [
44     'django.middleware.security.SecurityMiddleware',
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
48     'django.contrib.auth.middleware.AuthenticationMiddleware',
49     'django.contrib.messages.middleware.MessageMiddleware',
50     'django.middleware.clickjacking.XFrameOptionsMiddleware',
51 ]
```

2. Crear una **migración**, para que cuando se realice algún cambio en el modelo ese cambio se vea reflejado en las tablas, para lo cual abrimos la consola CMD, entramos a la ruta del proyecto que estamos realizando y digitamos el código:

python manage.py makemigrations y luego verificamos la creación en **migrations**

```
c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>python manage.py makemigrations
Migrations for 'miapp':
  miapp\migrations\0001_initial.py
    - Create model Article
    - Create model Category

c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>
```



```
1 Generated by Django 4.1.6 on 2023-02-22 19:26
2
3 from django.db import migrations, models
4
5
6 class Migration(migrations.Migration):
7
8     initial = True
9
10    dependencies = [
11    ]
12
13    operations = [
14        migrations.CreateModel(
15            name='Article',
16            fields=[
17                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
18                ('title', models.CharField(max_length=100)),
19                ('content', models.TextField()),
20                ('public', models.BooleanField()),
21                ('created_at', models.DateTimeField(auto_now_add=True)),
22                ('updated_at', models.DateTimeField(auto_now=True)),
23            ],
24        ),
25        migrations.CreateModel(
26            name='Category',
27            fields=[
28                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
```

3. Creación del SQL que se ejecutara en nuestro SGBD

python manage.py sqlmigrate miapp(Nombre de la app) 0001(numero de la migración)

```
c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>python manage.py sqlmigrate miapp 0001
BEGIN;
--
-- Create model Article
--
CREATE TABLE "miapp_article" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(
100) NOT NULL, "content" text NOT NULL, "public" bool NOT NULL, "created_at" datetime NOT NULL,
"updated_at" datetime NOT NULL);
--
-- Create model Category
--
CREATE TABLE "miapp_category" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(
100) NOT NULL, "description" varchar(250) NOT NULL, "created_at" date NOT NULL);
COMMIT;

c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>
```

3.1 migramos el sql **python manage.py migrate**

```
c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, miapp, sessions
Running migrations:
  Applying miapp.0001_initial... OK

c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>
```



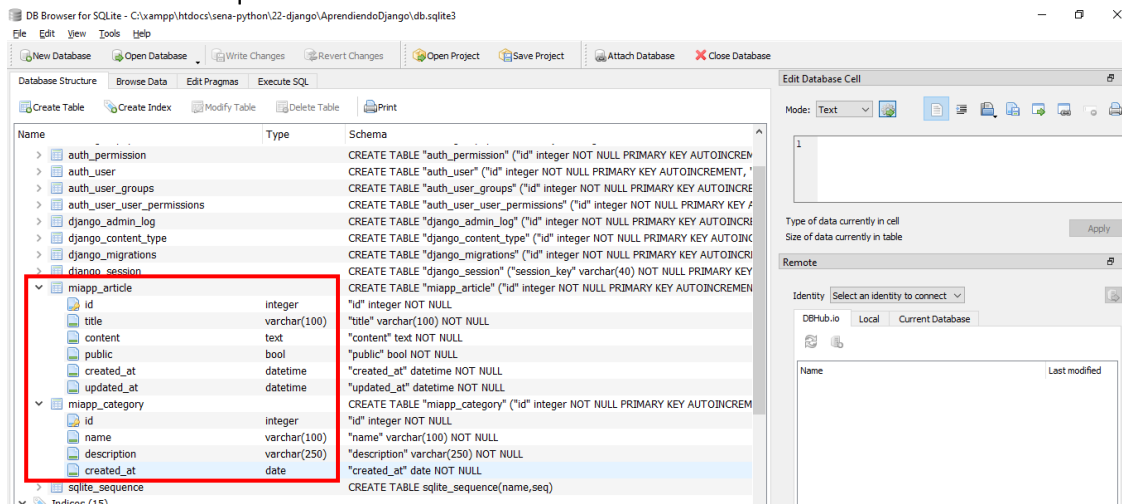


Verificación si fue creada la BD

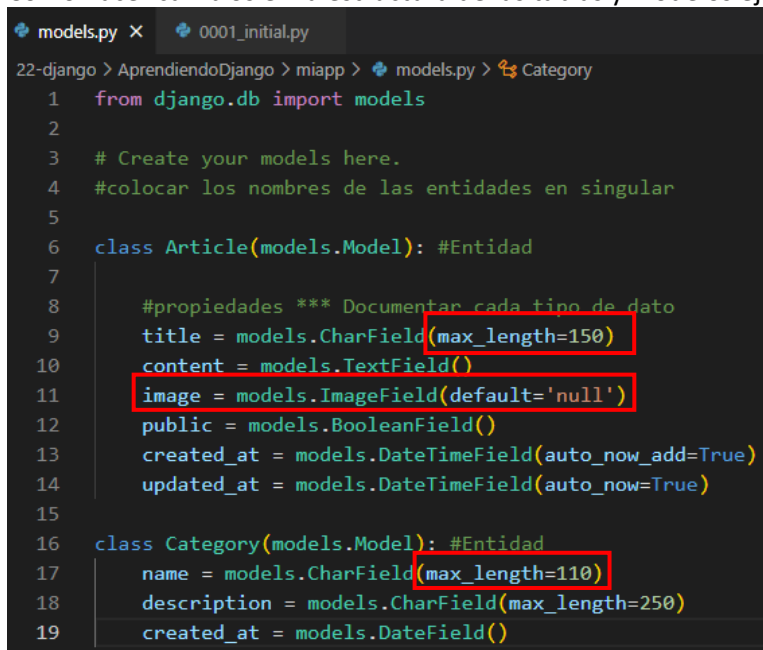
Descargar e instalar el programa **db browser for sqlite**



Se arrastra la db.sqlite3 al programa ejecutado, aparecen más tablas, porque Django tiene utilidades ya listas para el panel de administración que veremos mas adelante.



Como hacer cambios en la estructura de las tablas y modelos ejemplo: se plantea realizar estos cambios y adiciones



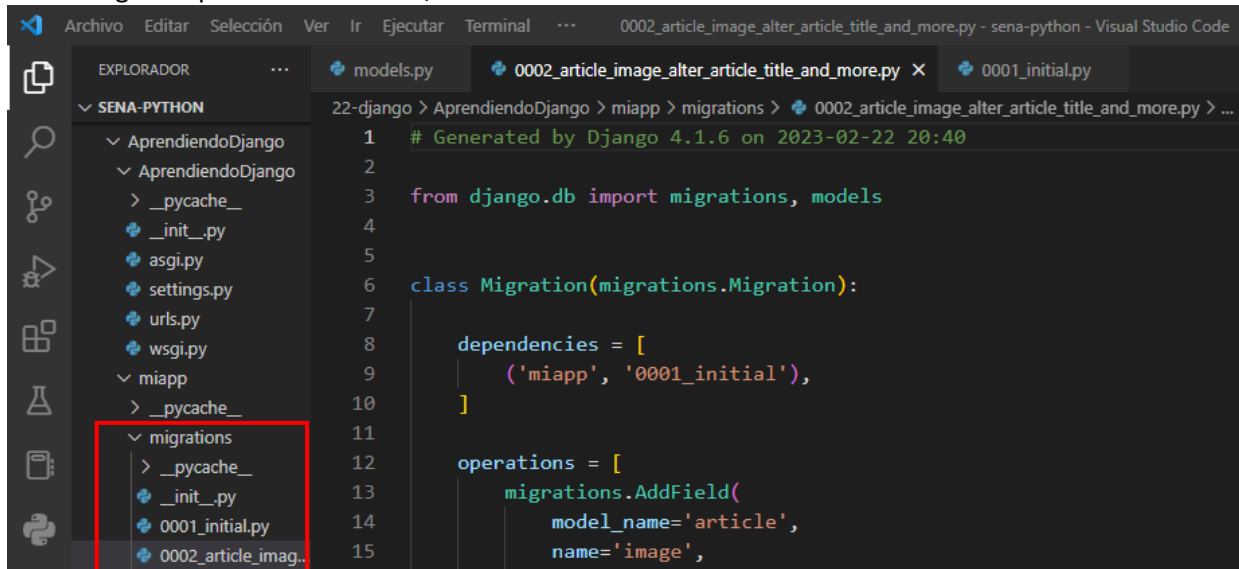


Luego se realiza una segunda migración ejecutando el comando **python manage.py makemigrations**

```
Simbolo del sistema
c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>python manage.py makemigrations
Migrations for 'miapp':
  miapp\migrations\0002_article_image_alter_article_title_and_more.py
    - Add field image to article
    - Alter field title on article
    - Alter field name on category

c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>
```

En la imagen se puede los AddField, AlterField



Paso siguiente se genera el **sql** a ejecutar correspondiente a la migración realizada anteriormente:

python manage.py sqlmigrate miapp 0002

```
Simbolo del sistema
c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>python manage.py sqlmigrate miapp 0002
BEGIN;
--
-- Add field image to article
--
CREATE TABLE "new_miapp_article" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "image" var
char(100) NOT NULL, "title" varchar(100) NOT NULL, "content" text NOT NULL, "public" bool NOT N
ULL, "created_at" datetime NOT NULL, "updated_at" datetime NOT NULL);
INSERT INTO "new_miapp_article" ("id", "title", "content", "public", "created_at", "updated_at"
```

Luego ejecutamos la migración de sql **python manage.py migrate**

```
Seleccionar Símbolo del sistema
c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, miapp, sessions
Running migrations:
  Applying miapp.0002_article_image_alter_article_title_and_more... OK

c:\xampp\htdocs\sena-python\22-django\AprendiendoDjango>
```





Así se ven los cambios en BD

| Name | Type |
|----------------------------|--------------|
| Tables (13) | |
| auth_group | |
| auth_group_permissions | |
| auth_permission | |
| auth_user | |
| auth_user_groups | |
| auth_user_user_permissions | |
| django_admin_log | |
| django_content_type | |
| django_migrations | |
| django_session | |
| miapp_article | |
| id | integer |
| content | text |
| public | bool |
| created_at | datetime |
| updated_at | datetime |
| image | varchar(100) |
| title | varchar(150) |

4. Guardar datos en la base de datos usando los modelos: usando las capas de abstracción

a. Creamos una vista en views.py `def crear_articulo`

```
views.py - sena-python - Visual Studio Code
53 return HttpResponse(layout+f"<h2>Contacto
54
55 def crear_articulo(request):
56     return HttpResponse("Usuario creado: ")
57
```

b. Creamos la ruta en urls.py

```
urls.py - sena-python - Visual Studio Code
31 path('pagina/', views.pagina, name="pagina"),
32 path('contacto/<str:nombre>/', views.contacto, name="contacto"),
33 path('pagina/', views.pagina, name="pagina"),
34 path('crear_articulo/', views.crear_articulo, name="crear_articulo"),
35
```

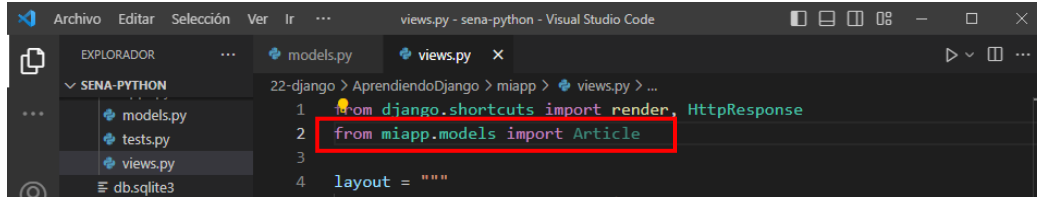
c. Arrancamos el servidor `python manage.py runserver` y se prueba la pagina en el navegador

```
python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 22, 2023 - 16:15:27
Django version 4.1.6, using settings 'AprendiendoDjango.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

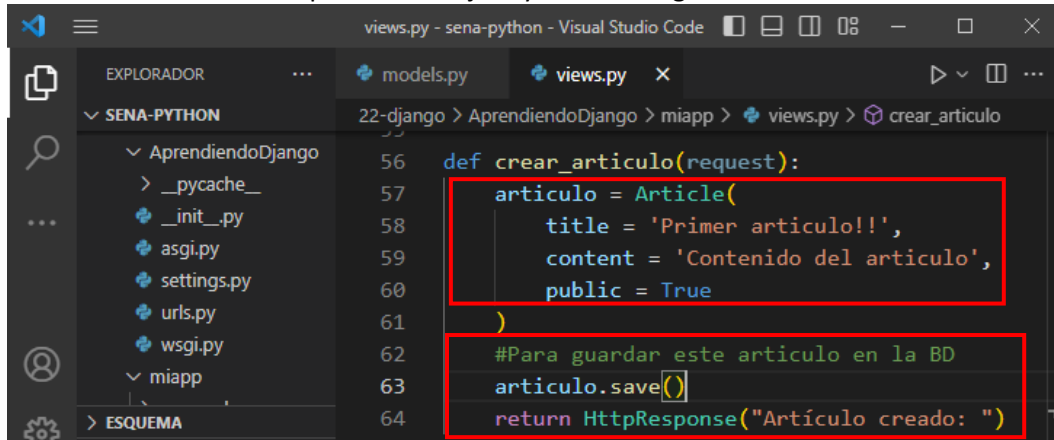


- d. Ahora hacemos uso de modelo, el cual se debe importar en **views.py** en la parte superior:



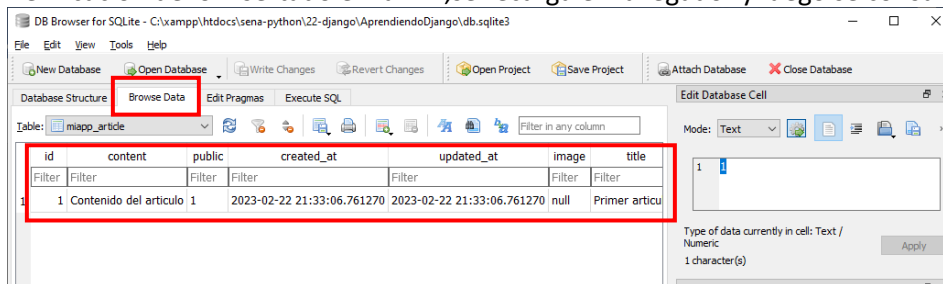
```
1 from django.shortcuts import render, HttpResponseRedirect
2 from miapp.models import Article
3
4 layout = ""
```

- e. Ahora usamos el modelo para crear objeto y crear un registro nuevo en BD



```
56 def crear_articulo(request):
57     articulo = Article(
58         title = 'Primer articulo!!',
59         content = 'Contenido del articulo',
60         public = True
61     )
62     #Para guardar este articulo en la BD
63     articulo.save()
64     return HttpResponseRedirect("Artículo creado: ")
```

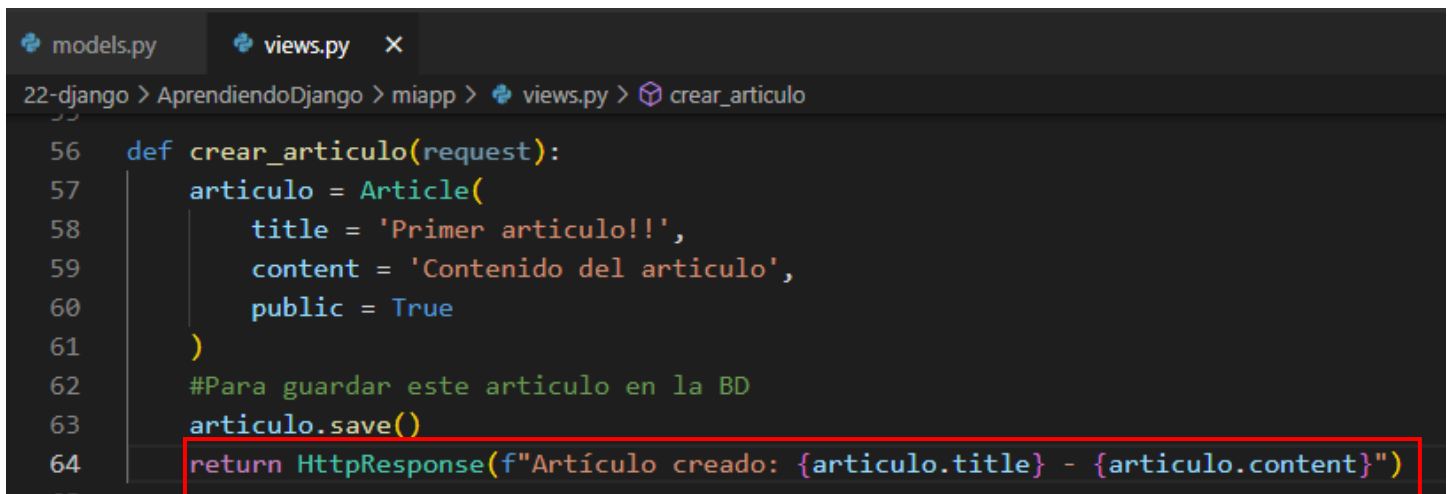
Verificación de lo insertado en la BD ,se recarga el navegador y luego se consulta los datos en la BD



| id | content | public | created_at | updated_at | image | title |
|----|------------------------|--------|----------------------------|----------------------------|-------|-----------------|
| 1 | Contenido del articulo | 1 | 2023-02-22 21:33:06.761270 | 2023-02-22 21:33:06.761270 | null | Primer articulo |

Guardar registros utilizando parámetros de la Url

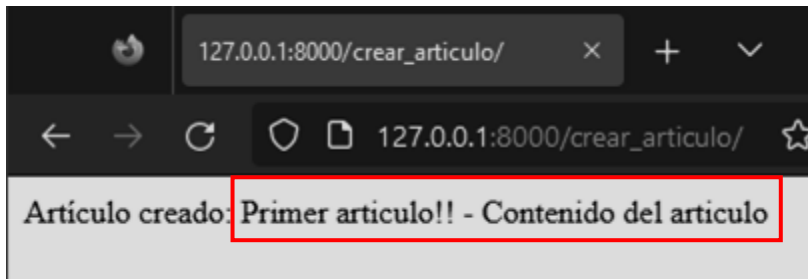
En el views.py se puede acceder al objeto y su información:



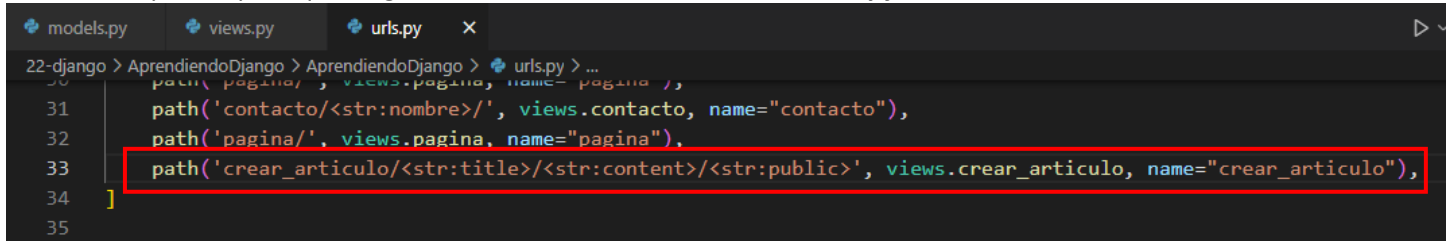
```
56 def crear_articulo(request):
57     articulo = Article(
58         title = 'Primer articulo!!',
59         content = 'Contenido del articulo',
60         public = True
61     )
62     #Para guardar este articulo en la BD
63     articulo.save()
64     return HttpResponseRedirect(f"Artículo creado: {articulo.title} - {articulo.content}")
```

Aquí se muestra el articulo creado en return

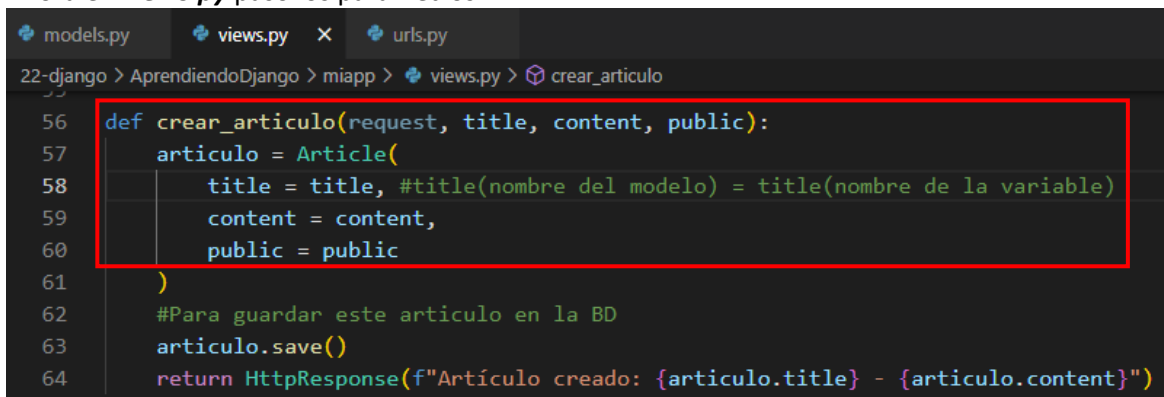




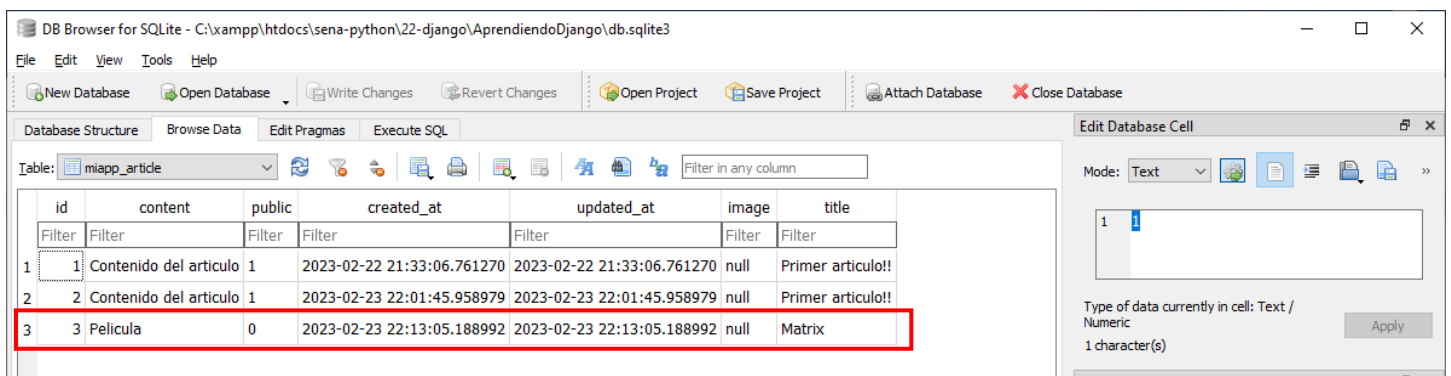
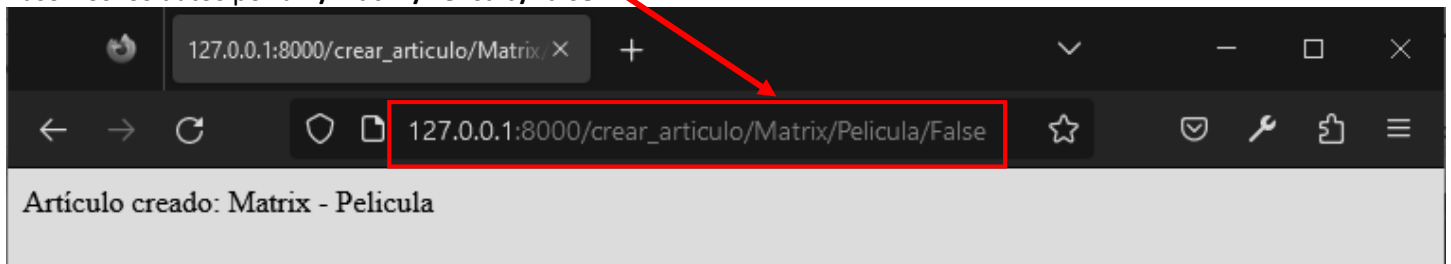
Pasar datos por url para que se guarden en la BD, esto en el archivo **urls.py**



Ahora en **views.py** paso los parámetros



Pasemos los datos por url **/Matrix/Película/False**





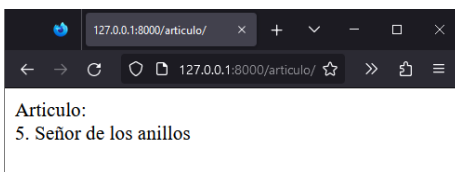
5. **Mostar datos y elementos de la base de datos:** creamos un nuevo método `def articulo()` para hacer la consulta a la BD

En la sentencia el `get(pk=3)`, `get(id=3)`, `get(title="Matrix")`

```
models.py views.py x urls.py
22-django > AprendiendoDjango > miapp > views.py > articulo
64 return HttpResponse(f"Artículo creado: {articulo.title} - {articulo.content}")
65
66 def articulo(request):
67     #creamos una variable llamada articulo y hacemos uso del modelo con objects
68     articulo = Article.objects.get(title="Señor de los anillos", public=True) #get saca un solo objeto o registro de la BD
69     return HttpResponse(f"Artículo: <br/> {articulo.id}. {articulo.title}") #se define la vista como se mostrara el articulo
```

Luego creamos la ruta la url

```
models.py views.py urls.py x
22-django > AprendiendoDjango > AprendiendoDjango > urls.py > ...
33 path('crear_articulo/<str:title>/<str:content>/<str:public>', views.crear_articulo, name="crear_articulo"),
34 path('articulo/', views.articulo, name="articulo")
35 ]
```



Captura de excepciones:

Suponga el caso donde se altere el código para que resulte una excepción (error)

```
models.py views.py x urls.py
22-django > AprendiendoDjango > miapp > views.py > ...
64 return HttpResponse(f"Artículo creado: {articulo.title} - {articulo.content}")
65
66 def articulo(request):
67     #creamos una variable llamada articulo y hacemos uso del modelo con objects
68     articulo = Article.objects.get(title="Señor de los anillos", public=False) #get
69     return HttpResponse(f"Artículo: <br/> {articulo.id}. {articulo.title}") #se def
70
```



Corrección del código anterior para capturar la excepción:

```
models.py | views.py | urls.py
22-django > AprendiendoDjango > miapp > views.py > articulo
64 | return HttpResponse(f"Artículo creado: {articulo.title} - {articulo.content}")
65 |
66 | def articulo(request):
67 |     #creamos una variable llamada articulo y hacemos uso del modelo con objects
68 |     #captura de excepcion
69 |     try:
70 |         articulo = Article.objects.get(title="Señor de los anillos", public=False) #get saca un solo objeto o registro de la BD
71 |         response = f"Artículo: <br/> {articulo.id}. {articulo.title}" #se define la vista como se mostrara el articulo
72 |     except:
73 |         response = "<h1> Artículo no encontrado</h1>"
74 |     return HttpResponse(response)
75 |
```



Realizar el cambio en el código para cuando encuentra el artículo.

6. Actualizar registros en una tabla de la base de datos

Creamos una nueva *url* y una nueva *vista* que se llamara *editar_articulo*

Vista – *views.py*

```
74 | return HttpResponse(response)
75 |
76 | # Actualizar Artículo
77 | def editar_articulo(request, id): #este id se pasa por la url del articulo que quiero editar
78 |     #aqui se buscar el articulo por el id
79 |     #creamos una variable articulo
80 |     articulo = Article.objects.get(pk=id) #se busca ese articulo por primary key pk = id
81 |     #Aqui se hace un cambio pero con valores dados desde aqui, como ejemplo
82 |     articulo.title= "Iron Man"
83 |     articulo.content = "La película de superhéroes que lanzó la mayor franquicia de Hollywood"
84 |     articulo.public = True
85 |     #asi se guardan los cambios
86 |     articulo.save()
87 |     return HttpResponse(f"Artículo Editado: {articulo.title} - {articulo.content}")
88 |
```



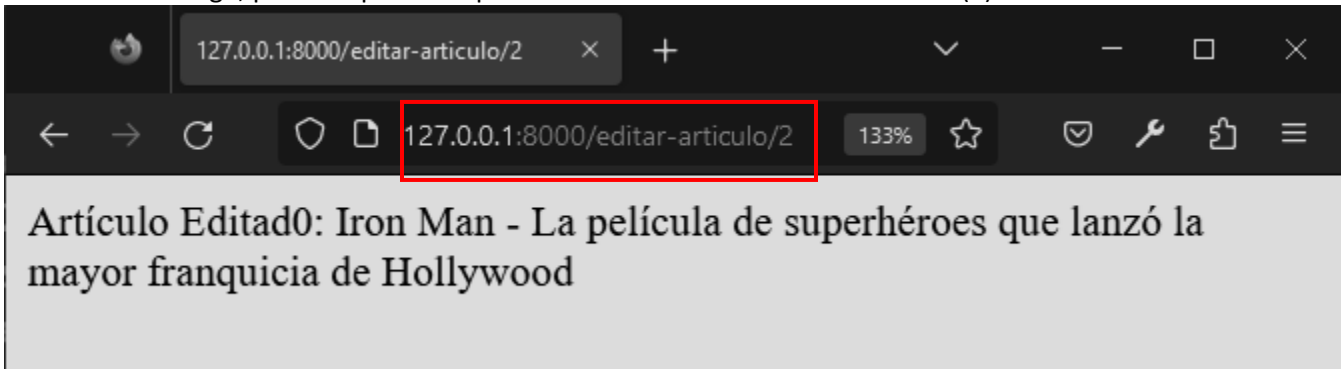
Creación de url - url.py

```

33 path('crear_articulo/<str:title>/<str:content>/<str:public>', views.crear_articulo, name="crear_articulo"),
34 path('articulo/', views.articulo, name="articulo"),
35 #ruta para editar un articulo y se le pasa el id por la url
36 path('editar-articulo/<int:id>', views.editar_articulo), #el name no es obligatorio
37 ]
38

```

Probamos el código, pasando por url el parámetro a editar en nuestro caso el id (2)



Vista del cambio en BD

| Table: miapp_article | | | | | | | |
|----------------------|--------|---|--------|----------------|----------------|--------|----------------------|
| | id | content | public | created_at | updated_at | image | title |
| | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | Contenido del articulo | 1 | 2023-02-22 ... | 2023-02-22 ... | null | Primer articulo!! |
| 2 | 2 | La película de superhéroes que lanzó la mayor franquicia de Hollywood | 1 | 2023-02-23 ... | 2023-03-06 ... | null | Iron Man |
| 3 | 5 | Saga | 1 | 2023-02-24 ... | 2023-02-24 ... | null | Señor de los anillos |

7. Listar todos los artículos:

Creamos una vista (views.py) artículos, de esta manera pasamos los objetos que están en la base de datos a nuestro template

```

86 articulo.save()
87 return HttpResponse(f"Artículo Editado: {articulo.title} - {articulo.content}")
88
89 #Mostrar articulos
90 def articulos(request):#creacion de la vista articulos
91     #Creamos una template para listar varos articulos
92     #1. hacemos la peticion a la base de datos
93     #Creamos la variable articulos y Llamamos al modelo Article, usamos objects para poder hacer una consulta
94     articulos = Article.objects.all() #En este caso no usamos el metodo get si no el metodo all() para sacar toda la información
95     #dentro de articulos tenemos una lista de objetos un array de objetos
96     return render(request, 'articulos.html', { #pasamos el request y la template articulos.html y como tercer parametro
97         'articulos' : articulos #pasamos un diccionario con las variables que deseamos mostrar
98     })
99

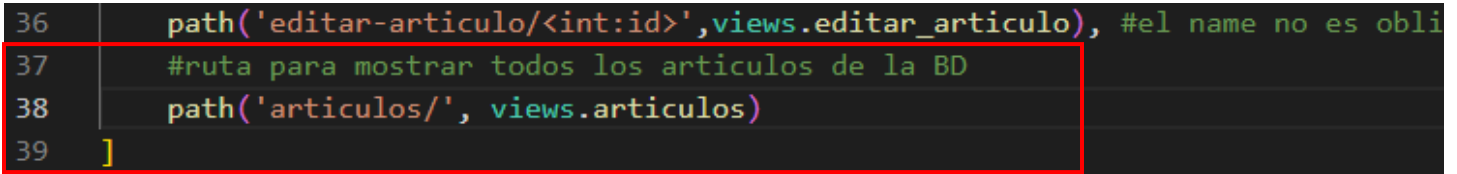
```




Creamos el template **artículos.html**



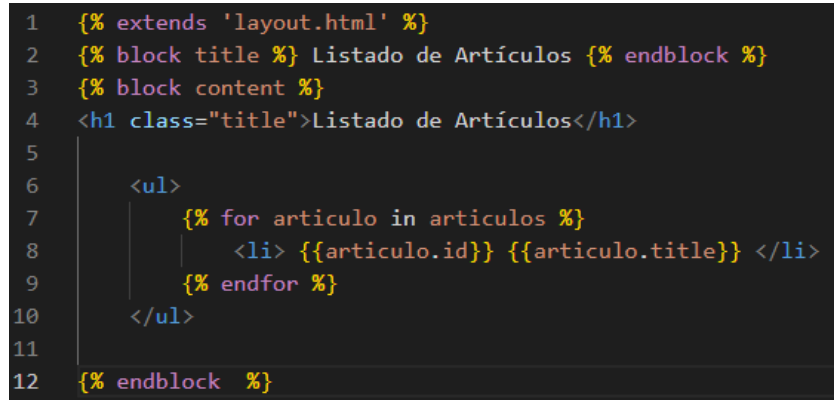
Ahora creamos la url



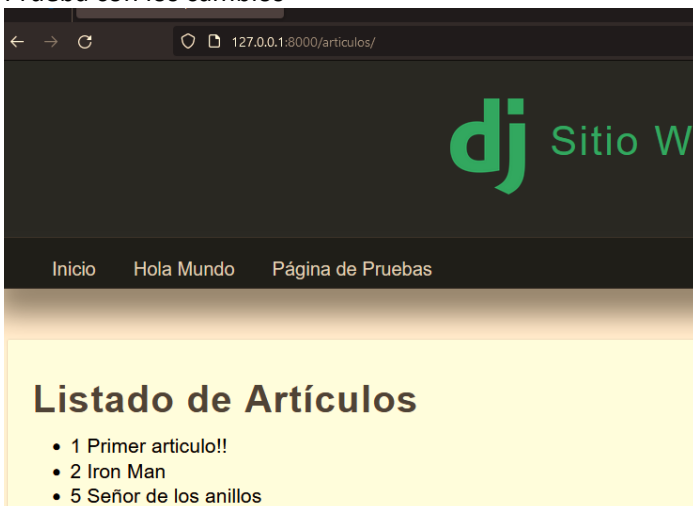
Prueba de código



Para mostrar todos los artículos solo se recorrer el listado de objetos



Prueba con los cambios





Algunas opciones con relación a uso de `all()`, `order_by()` donde se pueda ordenar el listado por cualquier atributo o campo por ejemplo: también realizar el cambio con `order_by('-title')`, `limit[:3]` ó `[3:5]`

```

89 #Mostrar articulos
90 def articulos(request):#creacion de la vista articulos
91     #Creamos una template para listar varos articulos
92     #1. hacemos la peticion a la base de datos
93     #Creamos la variable articulos y llamamos al modelo
94     articulos = Article.objects.order_by('title') #En
95     #dentro de articulos tenemos una lista de objetos
96     return render(request, 'articulos.html', { #pasamos
97         'articulos' : articulos #pasamos un diccionario
98     })

```

Listado de Artículos

- 2 Iron Man
- 1 Primer articulo!!
- 5 Señor de los anillos

-title

Listado de Artículos

- 5 Señor de los anillos
- 1 Primer articulo!!
- 2 Iron Man

limitar el número de elementos que aparecen `[:2]` ó `[2:3]`

```

89 #Mostrar articulos
90 def articulos(request):#creacion de la vista articulos
91     #Creamos una template para listar varos articulos
92     #1. hacemos la peticion a la base de datos
93     #Creamos la variable articulos y llamamos al modelo
94     articulos = Article.objects.order_by('-title')[:2] #E
95     #dentro de articulos tenemos una lista de objetos un
96     return render(request, 'articulos.html', { #pasamos e
97         'articulos' : articulos #pasamos un diccionario d
98     })

```

Listado de Artículos

- 5 Señor de los anillos
- 1 Primer articulo!!

8. **Borrar elementos:** Creamos una nueva vista `borrar_articulo()`

```

100 #Borrar articulo
101 def borrar_articulo(request, id):
102     #Asi selecciono el articulo dependiendo de lo que se pase por la url
103     articulo = Article.objects.get(pk=id)
104     #despues de seleccionado se borra de la siguiente manera
105     articulo.delete()
106     #redireccionar a la pagina de mostrar articulos para ver los cambios
107     return redirect('articulos')

```

Creamos la url

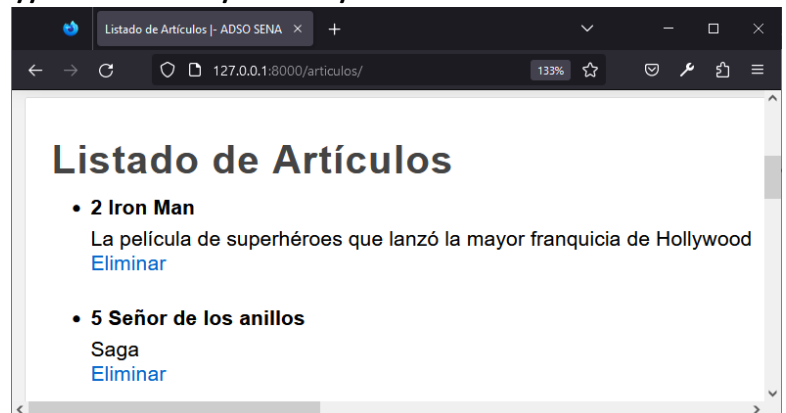


```
38 path('articulos/', views.articulos, name="articulos"),
39 #ruta borrar articulo
40 path('borrar-articulo/<int:id>', views.borrar_articulo, name="borrar")
41 ]
42
```

Creamos un enlace para borrar y ejecutamos la url <http://127.0.0.1:8000/articulos/> →

Listado de Artículos

- 1 Primer artículo!!
Contenido del artículo
[Eliminar](#)
- 2 Iron Man
La película de superhéroes que lanzó la mayor franquicia de Hollywood
[Eliminar](#)
- 5 Señor de los anillos
Saga
[Eliminar](#)



| Table: miapp_article | | Filter in any column | | | | | |
|----------------------|--------|---|--------|----------------|----------------|--------|----------------------|
| | id | content | public | created_at | updated_at | image | title |
| | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 2 | La película de superhéroes que lanzó la mayor franquicia de Hollywood | 1 | 2023-02-23 ... | 2023-03-06 ... | null | Iron Man |
| 2 | 5 | Saga | 1 | 2023-02-24 ... | 2023-02-24 ... | null | Señor de los anillos |



Se podría crear en el layout un nuevo elemento en el nav

```

25     <nav>
26         <ul>
27             <li>
28                 <a href="{% url 'inicio' %}">Inicio</a>
29             </li>
30             <li>
31                 <a href="{% url 'articulos' %}">Artículos</a>
32             </li>
33             <li>
34                 <a href="{% url 'hola_mundo' %}">Hola Mundo</a>
35             </li>
36             <li>
37                 <a href="{% url 'pagina' %}">Página de Pruebas</a>
38             </li>
39         </ul>
40     </nav>
41 
```



9. Consultas y condiciones

Vamos a **vistas** y trabajamos sobre la vista de artículos

```

94     articulos = Article.objects.all() #En este caso no usamos el metodo get s
95     #dentro de articulos tenemos una lista de objetos un array de objetos
96
97     #Consultas con condiciones filter para filtrar por un valor especifico
98     articulos = Article.objects.filter(title = "Iron Man")

```

Ahora se realiza un cambio en **artículos.html**

```

6     <ul>
7         {% for articulo in articulos %}
8             <li>
9                 <h4>{{articulo.id}} {{articulo.title}}</h4>
10                <span>{{articulo.created_at}}</span>
11            <p>
12                {{articulo.content}}
13                <br/>
14                <a href = "{% url 'borrar' id=articulo.id %}">Eliminar</a>
15            </p>

```





Otro filtro – cambiar la palabra Tiburón por alguna de su BD para comprobar el código

title__contains - title__exact - title__iexact

```

89 #Mostrar articulos
90 def articulos(request):#creacion de la vista articulos
91     #Creamos una template para listar varos articulos
92     #1. hacemos la peticion a la base de datos
93     #Creamos la variable articulos y Llamamos al modelo Article, usamos ob
94     articulos = Article.objects.all() #En este caso no usamos el metodo get
95     #dentro de articulos tenemos una lista de objetos un array de objetos
96
97     #Consultas con condiciones filter para filtrar por un valor especifico
98     articulos = Article.objects.filter(title__contains = "Tiburón")
99     return render(request, 'articulos.html', { #pasamos el request y la tem
100         'articulos' : articulos #pasamos un diccionario con las variables q
101     })

```

documentation django lookups filter

Ejemplo: Entrada. objetos . filtro (id__gt = 4) SQL equivalente: SELECCIONE ... DONDE id > 4 ; gte

Buscar en la documentación de django los equivalentes en SQL de los siguientes lookups y realizar un ejemplo de cada uno con los registros de su BD. Además, consultar cómo se utiliza el and ó el or

Gte lt lte istartswith endswith distinct

10.Exclude

En la vista realizamos otra consulta





Cambio para la nueva consulta:

```

97 #Consultas con condiciones filter para filtrar por un valor específico
98 articulos = Article.objects.filter(title__contains = "Tiburón")
99
100 #Realizar una consulta que muestre solo los estén publicados y excluya bajo una condición
101 articulos = Article.objects.filter(title = "Tiburón",).exclude(public=True)
102
103 return render(request, 'articulos.html', { #pasamos el request y la template articulos.html

```

Resultado: un solo registro debido a la condición y lo que se encuentra en BD



| id | content | public | created_at | updated_at | image | title |
|----|--|--------|----------------|----------------|-------|-----------------------|
| 1 | La mejor película sobre la esclavitud americana jamás hecha, y encim... | 1 | 2023-02-28 ... | 2023-02-28 ... | null | Déjame salir |
| 2 | Clint Eastwood se pone un poncho y se convierte en un icono america... | 1 | 2023-03-02 ... | 2023-03-02 ... | null | El bueno, el feo y... |
| 3 | Es probablemente una de las películas más terroríficas de este siglo, y... | 1 | 2023-03-06 ... | 2023-03-06 ... | null | Expediente Warren |
| 4 | La película de superhéroes que lanzó la mayor franquicia de Hollywood | 1 | 2023-02-23 ... | 2023-03-06 ... | null | Iron Man |
| 5 | En el mundo del crimen hay transgresiones de todo tipo, pero hay un ... | 1 | 2023-02-23 ... | 2023-02-23 ... | null | John Wick |
| 6 | Moonlight se las arregla para examinar la intersección de ser un homb... | 1 | 2023-02-24 ... | 2023-02-24 ... | null | Moonlight |
| 7 | The Lord of the Rings) es una novela de fantasía épica escrita por el ... | 1 | 2023-02-24 ... | 2023-02-24 ... | null | Señor de los anillos |
| 8 | Es posible que esta imagen capture a la perfección el alma de la ... | 1 | 2023-03-06 ... | 2023-03-06 ... | null | Tiburón |
| 9 | El tiburón era una marioneta gigante con dientes falsos, y aún así te ... | 0 | 2023-02-24 ... | 2023-02-24 ... | null | Tiburón |

```

9 <h4>{{articulo.id}} {{articulo.title}}</h4>
10 <span>{{articulo.created_at}}</span>
11 {% if articulo.public %}
12     <strong>Publicado</strong>
13     {% else %}
14     <strong>Privado</strong>
15     {% endif %}
16 <p>
17     {{articulo.content}}
18     <br/>
19     <a href = "{% url 'borrar' id=articulo.id %}">Eliminar</a>

```





11. Ejecutar SQL desde Django

```

100 #Realizar una consulta que muestre solo los esten publicados y excluya bajo una condición
101 articulos = Article.objects.filter(title = "Tiburón",).exclude(public=True)
102
103 #Consulta ejecutando SQL
104 articulos = Article.objects.raw("SELECT * FROM miapp_article WHERE title='Tiburón' AND public=1")
105
106 return render(request, 'articulos.html', { #pasamos el request y la template articulos.html y como

```



12. OR en consultas con el ORM

Para utilizar el OR debemos importar `from django.db.models import Q`

```

2 from miapp.models import Article
3 from django.db.models import Q

```

Y realizamos la consulta:

```

105 articulos = Article.objects.raw("SELECT * FROM miapp_article WHERE title='Tiburón' AND public=1")
106
107 #Consulta utilizando el OR con ORM
108 articulos = Article.objects.filter(
109     Q(title__contains="John") | Q(title__contains="Señor")
110 )

```

