**ADMINISTRADOR DE DJANGO**

1.Partimos del siguiente código:



2. Leemos la pagina 1 del manual de alumno.

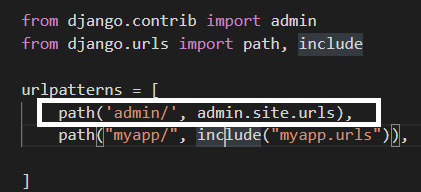
El administrador de django nos permite hacer CRUD (crear, ver, editar y eliminar) de las instancias de los modelos de nuestra aplicación.

3. Vemos que django.contrib.admin es el administrador de django y se encuentra en INSTALLED\_APPS de settings.py. Para terminar de configurar el admin de django debemos incorporar a nuestro proyecto las direcciones de URL de esta aplicación (django.contrib.admin).

En **practica1/urls.py** vemos que ya contiene la línea:

 path('admin/', admin.site.urls),

de forma tal que **urls.py** (del proyecto) nos queda:



De esta manera las direcciones del administrador de django comenzaran con un **admin/**.

4. Starteamos el server.

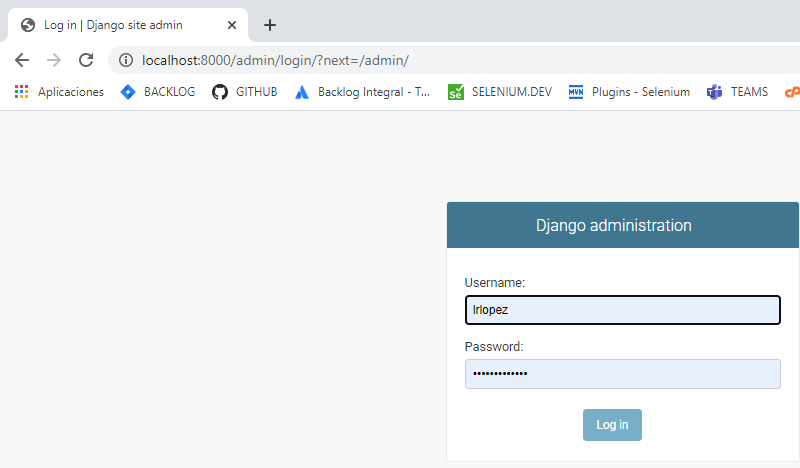
C:\Users\lrlopez\Desktop\Desarrollo\_Web\_Python\_Django\CLASE 6\MODELOS>python manage.py runserver

5. Creamos **usuario** y **contraseña**:

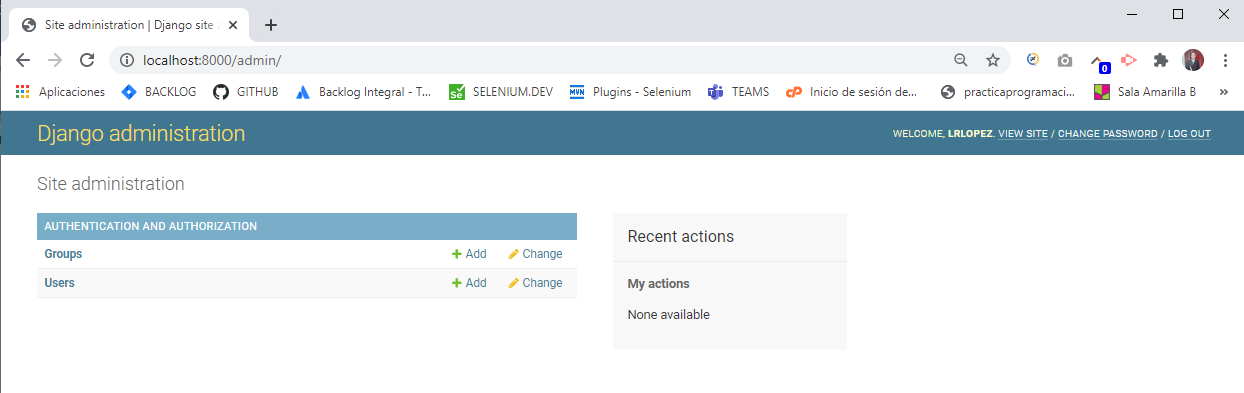
Esto posibilitara que un usuario pueda crear las vistas para trabajar con nuestros modelos. Cargando cursos, profesores, y otros objetos.

**python manage.py createsuperuser** (crear un usuario)

6. <http://localhost:8000/admin/>



Y llegamos a:



7. Vemos los dos modelos definidos por la aplicación **django.contrib.auth** (ver INSTALLED\_APPS), **Groups** y **Users**. Gracias a estas aplicaciones podemos ejecutar operaciones **CRUD** sobre usuarios y grupos.

8. Cambiamos el idioma del administrador. **practica1/settings.py**

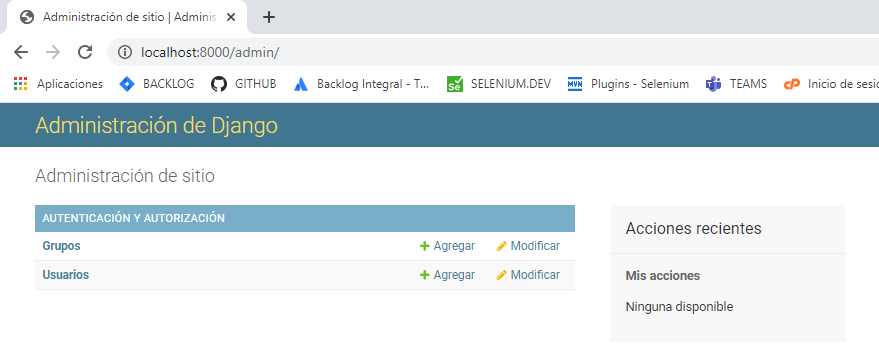
Cambiamos esto

LANGUAGE\_CODE = 'en-us'

Por esto:

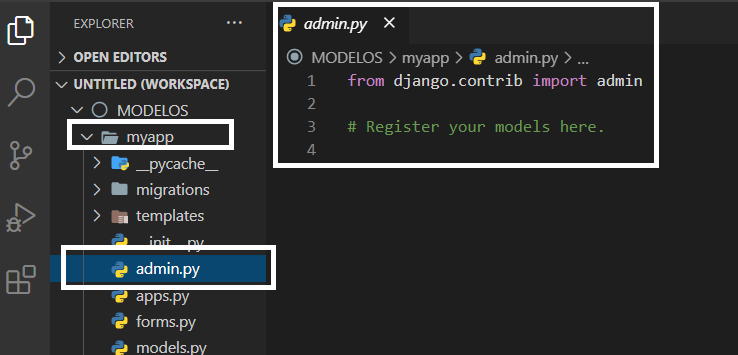
LANGUAGE\_CODE = 'es-ar'

9. Actualizamos <http://localhost:8000/admin/> y lo vemos en español.



10. Ahora bien, queremos que desde el administrador podamos gestionar **cursos** y **profesores.**

Registramos esos modelos en el administrador.



11. **admin.py** indicamos que modelos queremos que sean manejados por el admin de django.

Agregamos lo siguiente:

from django.contrib import admin

from.models import Curso, Profesor

# Register your models here.

admin.site.register(Curso)

admin.site.register(Profesor)

12. Refrescamos <http://localhost:8000/admin/>

Notese que django eligio los nombres **Cursos** y **Profesors** con una **s** final. Para indicar el plural correcto del modelo **Profesor** vamos a **models.py** y agregamos:

   class Meta:

        verbose\_name\_plural = "Profesores"

de forma tal que el modelo **Profesor** nos queda:

class Profesor(models.Model):

    nombre = models.CharField(max\_length=128)

    monotributista = models.BooleanField()

    class Meta:

        verbose\_name\_plural = "Profesores"

13. Tambien podemos hacer diferir los nombres en singular o plural del nombre del modelo que figura en el código:

**models.py**

reemplazamos este código:

class Profesor(models.Model):

    nombre = models.CharField(max\_length=128)

    monotributista = models.BooleanField()

    class Meta:

        verbose\_name\_plural = "Profesores"

por:

class Profesor(models.Model):

    nombre = models.CharField(max\_length=128)

    monotributista = models.BooleanField()

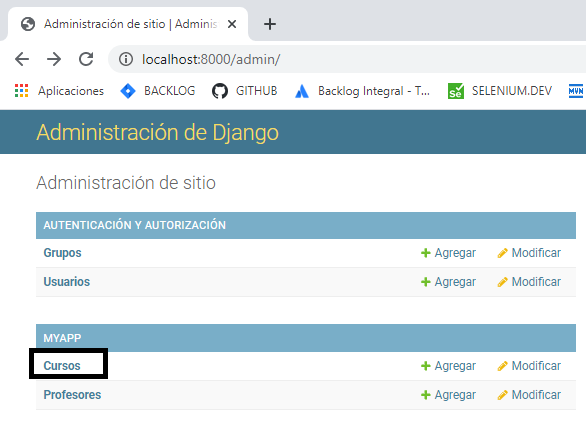
    class Meta:

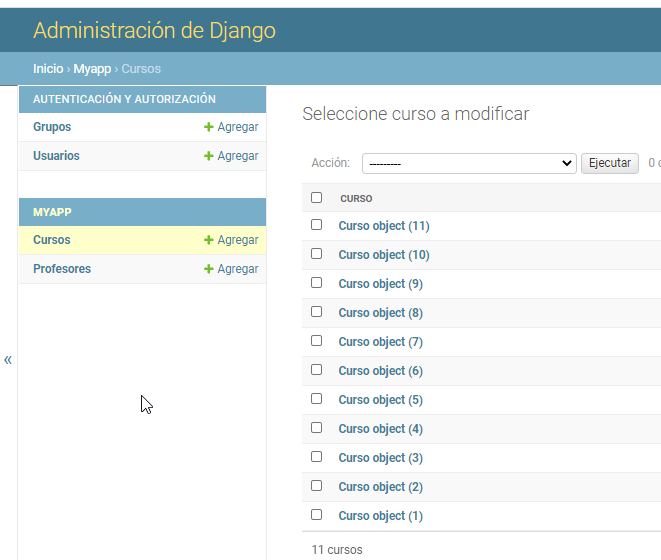
        verbose\_name\_plural = "Instructores"

        verbose\_name = "Instructor"

se modifico la **Meta**. Esta clase contiene ciertas opciones a nivel metadata relacionadas con el modelo en cuestión. Define permisos, nombres de tabla de BBDD asociadas, si el modelo es abstracto o no, versiones singulares y/o plurales del nombre del modelo (como en este caso), y otras.

14. En el admin presionamos **Cursos** y vemos una lista de los cursos almacenados en la BBDD.





15. Cada modelo tiene una representación como cadena, que se obtiene cuando convertimos una instancia a una cadena via **str()**. Por defecto, la representación de cada modelo es “NombreModelo object(id)”. Por eso vemos esa salida en la pantalla previa **Curso object(11)**.

Para cambiarla definimos la función especial **\_\_str\_\_()** dentro del modelo:

**models.py** => **Curso**

    def \_\_str\_\_(self):

        return self.nombre

de forma tal que el modelo **Curso** queda de la siguiente manera:

class Curso(models.Model):

    nombre = models.CharField("Nombre", max\_length=128)

    inscriptos = models.IntegerField("Inscriptos")

    TURNOS =(

        (1, "Mañana"),

        (2, "Tarde"),

        (3, "Noche")

    )

    turno = models.PositiveSmallIntegerField("Turno", choices=TURNOS, default=3)

    profesor = models.ForeignKey(Profesor, on\_delete=models.SET\_NULL, null=True, related\_name="cursos")

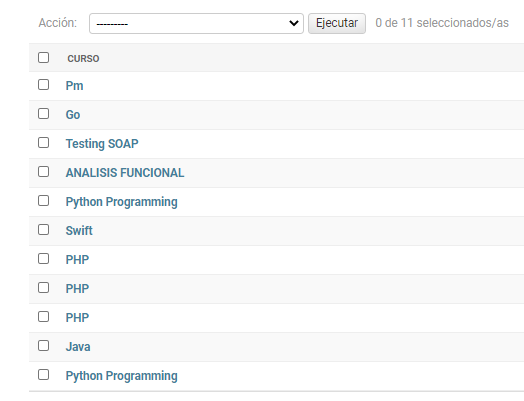
    def \_\_str\_\_(self):

        return self.nombre

Si **a** es un **Curso**, django llamara a la función **\_\_str\_\_(a)**. El argumento **self** obtiene la representación de **a**.

Lo que queremos es que la representación de cada curso sea su nombre, que es una cadena.

16. Refrescamos <http://localhost:8000/admin/myapp/curso/> y obtenemos la siguiente salida:



17. Hacemos lo mismo con los **profesores**:

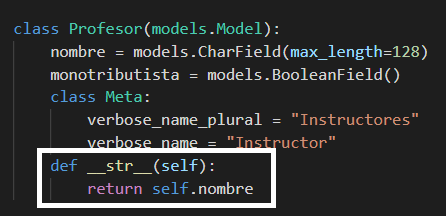
**models.py** => **Profesor**

Agregamos:

def \_\_str\_\_(self):

        return self.nombre

de forma tal que el modelo queda:

****

y en el admin nos queda:

****

Tambien probamos como cambiando

class Meta:

        verbose\_name\_plural = "Instructores"

        verbose\_name = "Instructor"

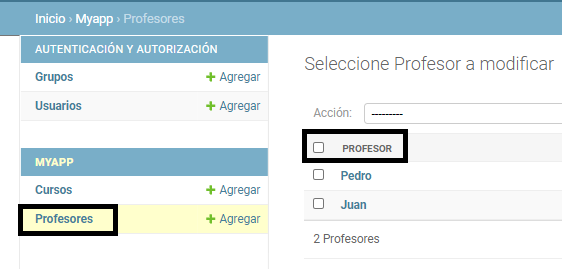
a

class Meta:

        verbose\_name\_plural = "Profesores"

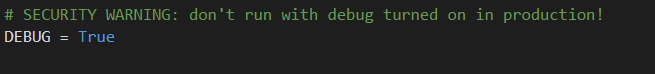
        verbose\_name = "Profesor"

Y vemos como queda:



**MODO DEPURACION**

1.Chequeamos **settings.py** que **DEBUG = True** (por defecto)

****

Por defecto, los proyectos corren en este modo. Esto implica que las excepciones lanzadas por códigos en las vistas u otras partes del framework serán mostradas en el navegador web.

2. Por ejemplo, vamos a **index()** y cambiamos esto:

def index ( request ):

    ctx = {

        "alumnos": ["Juan", "Sofia", "Matias"]

    }

    return render(request, "myapp/index.html", ctx)

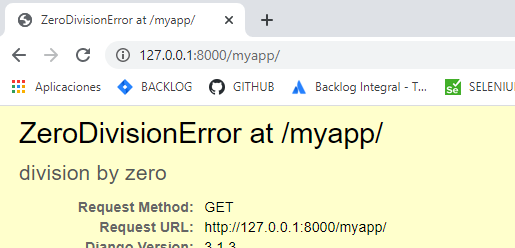
por esto:

def index ( request ):

    x = 7 / 0

lo cual genera una **ZeroDivisionError** exception.

4. <http://127.0.0.1:8000/myapp/> y observamos la excepción:



Esta salida es propia de Django. La aplicación solo debe correr en modo depuración únicamente en el momento del desarrollo y en forma local. Para evitar que se muestre al usuario en Produccion (modo depuración desactivado).

5. Probamos desactivar el modo depuracion.

a. **settings.py**:

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = False

b. Y también debemos setear **ALLOWED\_HOST** duando **DEBUG** esta en **False**.

ALLOWED\_HOSTS = ['127.0.0.1', 'localhost']

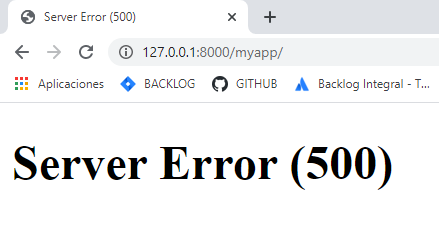
**Nota**: Debe contener nombres de host calificados, no urls. Entonces debemos sacar el protocolo y el puerto dejando únicamente el nombre del host.

O bien:

ALLOWED\_HOSTS = ['\*']

Para que tome cualquier host calificada.

6. Probamos nuevamente <http://127.0.0.1:8000/myapp/>:



El desarrollador podrá acceder a la información completa del error utilizando los archivos de registro. La salida previa es como los usuarios ven el resultado de la excepción en Produccion.

7. Checkpoint code:



**EJERCICIO\_1**

1.Creamos un proyecto desde cero:

django-admin startproject Ejercicio\_1\_Leo

2. Creamos la aplicación **myapp**.

python manage.py startapp myapp

3. Le indicamos a nuestro proyecto que hemos creado la aplicación myapp.

**Settings.py => INSTALLED\_APPS**

 "myapp.apps.MyappConfig",

4. Incorporamos a nuestro proyecto las direcciones URL de la aplicación **django.contrib.admin** (consola de Django):

Reemplazamos este código:

from django.contrib import admin

from django.urls import path

urlpatterns = [

    path('admin/', admin.site.urls),

]

Por este otro:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path("myapp/", include("myapp.urls"))

]

Esto significa que asi como las direcciones URL y vistas en myapp comenzaban con el prefijo *myapp/* , las direcciones del administrador de Django las configuramos para que comiencen con *admin/.*

5. Cambiamos el idioma al español: **settings.py**

LANGUAGE\_CODE = 'es-ar'

6. Creamos el archivo **urls.py** en myapp

from django.urls import path

from . import views

urlpatterns = [

    path("", views.index, name="index")

7. En **views.py** colocamos una función temporal (para que nos tome la configuración del punto 6 y 4):

def index():

    print("Hola mundo")

8. Aplicamos los modelos de las aplicaciones de django (preinstaladas) para poder crear un superuser

python manage.py migrate

9. Creamos un superUser:

python manage.py createsuperuser

10. **python manage.py runserver** y lugeo intentamos ingresar a <http://127.0.0.1:8000/admin/>

11. Creamos los modelos **Autor** y **Libro:**

from django.db import models

# Create your models here.

class Autor(models.Model):

    nombre = models.CharField(max\_length=128)

    apellido = models.CharField(max\_length=128)

class Libro(models.Model):

    titulo = models.CharField(max\_length=128)

    editorial = models.CharField(max\_length=128)

    autor = models.ForeignKey(Autor, on\_delete=models.SET\_NULL, null=True)

12. Registramos nuestros modelos en **admin.py**:

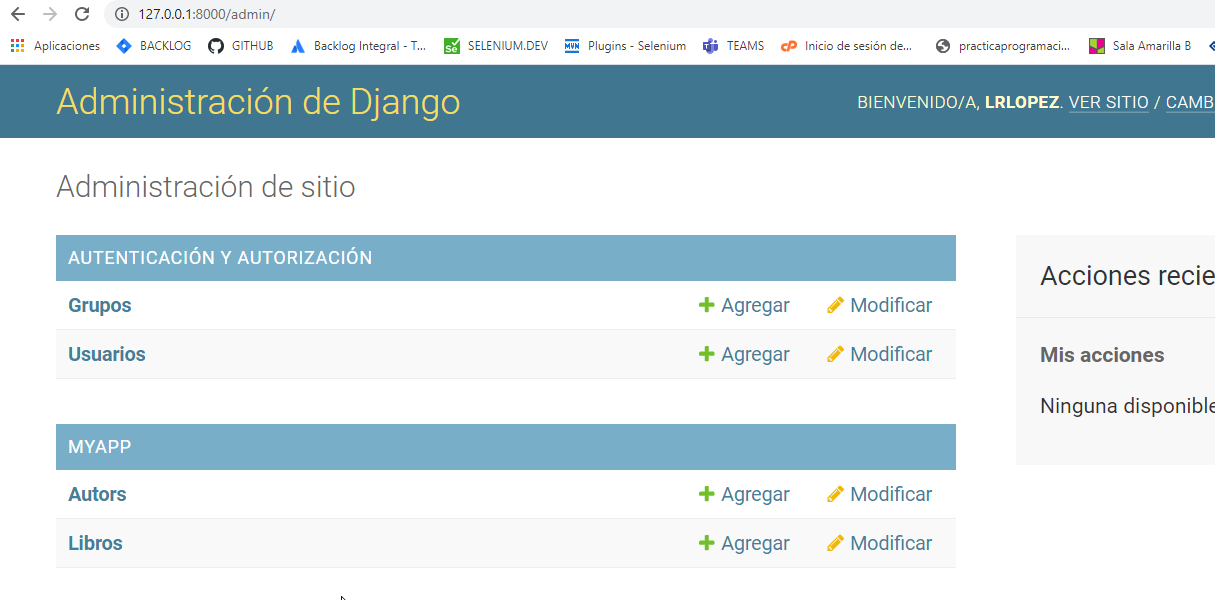
from django.contrib import admin

from.models import Autor, Libro

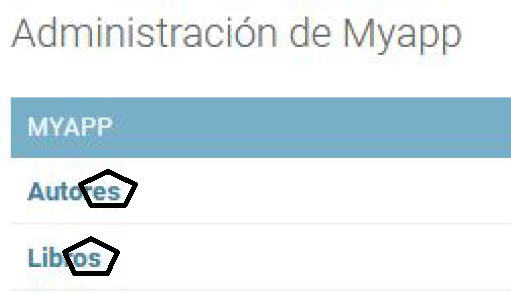
# Register your models here.

admin.site.register(Autor)

admin.site.register(Libro)



13. Cumplimos con lo que nos pide el ejercicio de que al ingresar en <http://127.0.0.1:8000/admin/myapp/> se observe lo siguiente:



Entonces agregamos las siguientes líneas al modelo **Autor**:

 class Meta:

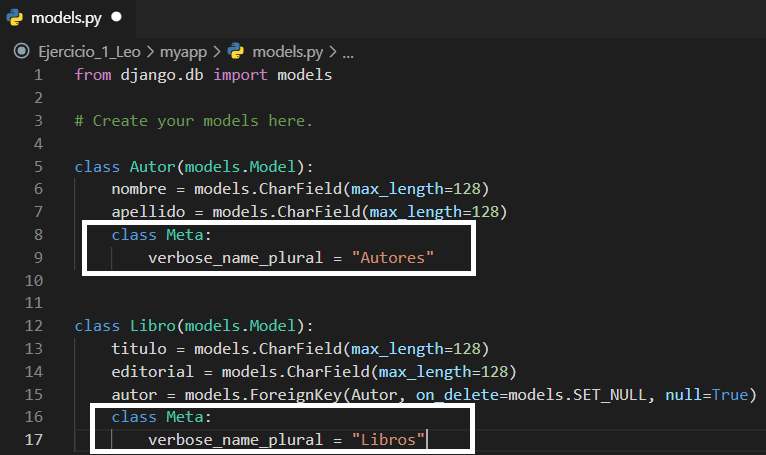
        verbose\_name\_plural = "Autores"

y a **Libro**

class Meta:

        verbose\_name\_plural = "Libros"

De forma tal que nos queda:



**Migramos:**

python manage.py makemigrations

python manage.py migrate

**Iniciamos:**

python manage.py runserver

**Vemos como queda:**

<http://127.0.0.1:8000/admin/myapp/>

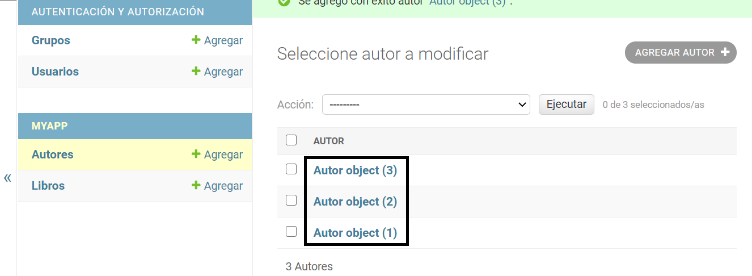


14. Ingresamos 3 autores:

1. Vamos a <http://127.0.0.1:8000/admin/>.
2. Agregamos los 3 autores propuestos por el ejercicio.

Sigmund Freud ; William Shakespeare ; Friedrich Nietzsche.

Veamos como nos queda:



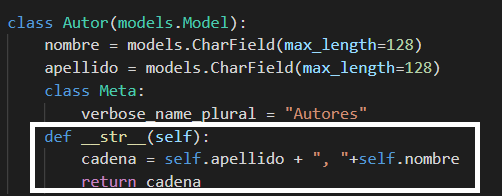
1. Para que nos quede como lo pide el ejercicio agregamos la siguiente función **Autor**

def \_\_str\_\_(self):

    cadena = self.apellido + ", "+self.nombre

    return cadena

de forma tal que nos queda:



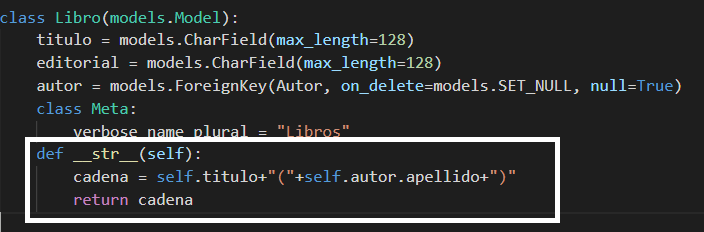
1. Y para el modelo **Libro**:

def \_\_str\_\_(self):

        cadena = self.titulo+"("+self.autor.apellido+")"

        return cadena

de forma tal que nos queda:



**Nota:**

Tambien podemos usar las interpolaciones a String de Python (f-strings)

def \_\_str\_\_(self):

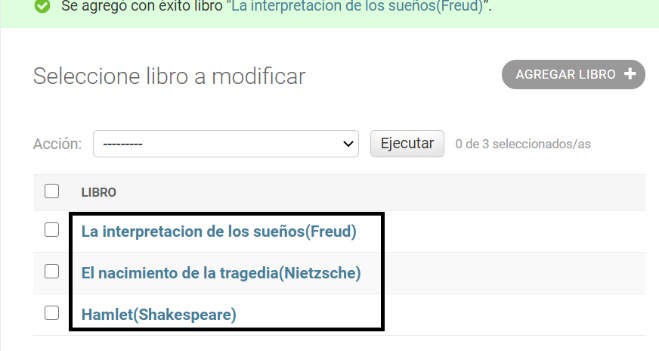
return f"{self.apellido}, {self.nombre}"

def \_\_str\_\_(self):

return f"{self.titulo} ({self.autor.apellido})"

respectivamente

1. De tal forma que nos queda:



**Nota**: El nombre del autor entre paréntesis no es parte del titulo del libro.

15. Codigo Resultante:



**PROYECTO\_INTEGRADOR**

1.Partimos del siguiente código:



2. Quitamos la vista **nueva\_remera()** (solo esta función).

3. Quitamos la plantilla **nueva\_remera.html**.

4. Quitamos el formulario, eliminamos el archivo **forms.py**.

5. Quitamos en **urls.py** la siguiente url:

path("nueva-remera", views.nueva\_remera, name="nueva\_remera")

Nota: Los puntos 2, 3, 4 y 5 los realizamos porque a partir de ahora usaremos el administrador de Django para crear, ver, editar y eliminar remeras.

6. En **urls.py** de nuestro proyecto incorporamos o linkeamos la dirección URL de la aplicación django.contrib.admin para que después el usuario pueda acceder a las urls correspondientes (<http://127.0.0.1:8000/admin/>...):

path('admin/', admin.site.urls)

de forma tal que el código nos queda:

from django.urls import path, include

from django.contrib import admin

urlpatterns = [

    path("shop/", include("shop.urls")),

    path('admin/', admin.site.urls)

]

7. **settings.py** configuramos el idioma:

LANGUAGE\_CODE = 'es-ar'

8. Creamos un superusuario:

cmd => python manage.py createsuperuser

9. Registramos el modelo Remera. En **admin.py** nos queda:

from django.contrib import admin

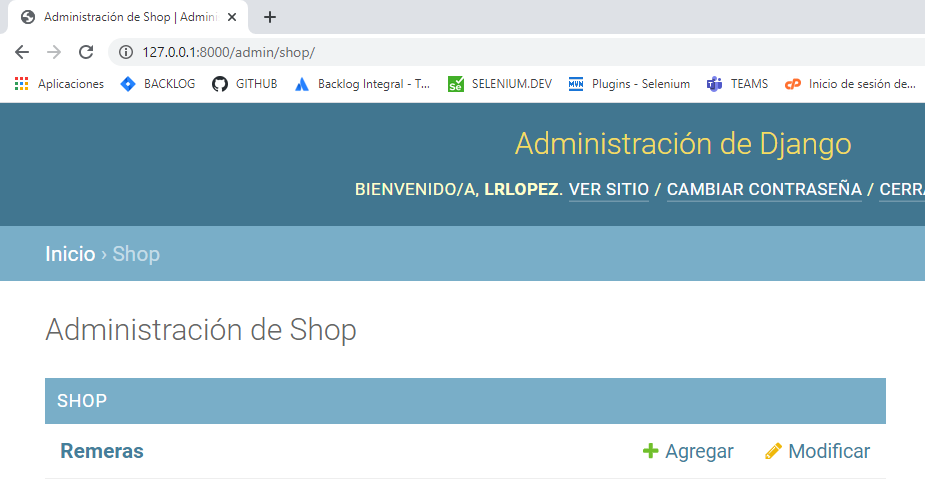
from .models import Remera

# Register your models here.

admin.site.register(Remera)

10. Accedemos a la consola

**python manage.py runserver** y luego <http://127.0.0.1:8000/admin/>



11. Para que quede como nos pide el integrador, es decir “Marca, Color, Tamaño” debemos realizar la siguiente modificación:

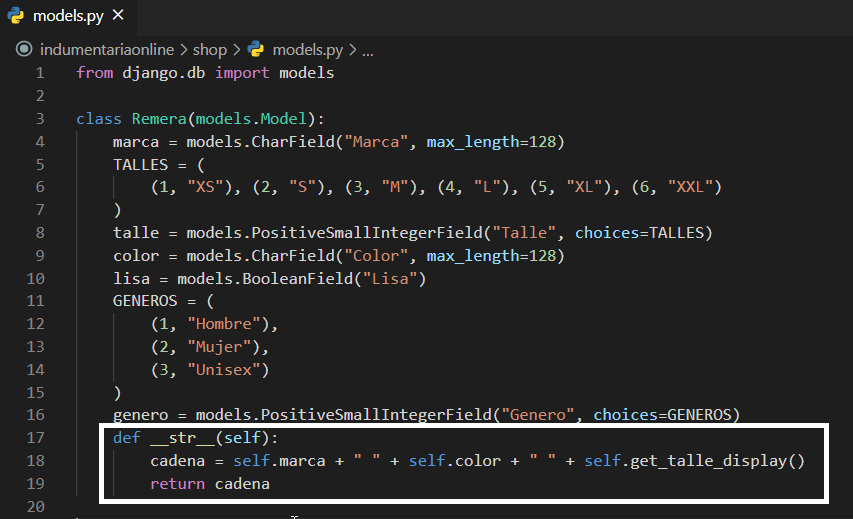
a. **models.py:** agregamos la siguiente función

  def \_\_str\_\_(self):

        cadena = self.marca + " " + self.color + " " + self.get\_talle\_display()

        return cadena

de forma tal que nos quede:



b. Tambien podríamos haber definido la función \_\_str\_\_ de la siguiente forma:

 def \_\_str\_\_(self):

        return f"{self.marca} {self.color} {self.get\_talle\_display()}"

y es exactamente lo mismo.

12. **views.py** borramos todos los import salvo

from django.shortcuts import render

from .models import Remera

13. Probamos crear nuevas remeras, modificar, eliminar y consultar.

14. Codigo resultante:



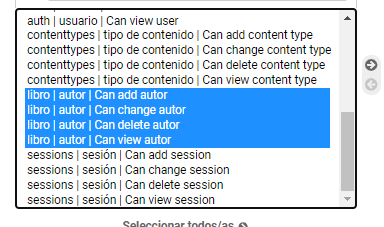
**EJERCITACION ADICIONAL:**

1.Creamos un grupo de usuarios desde la consola.

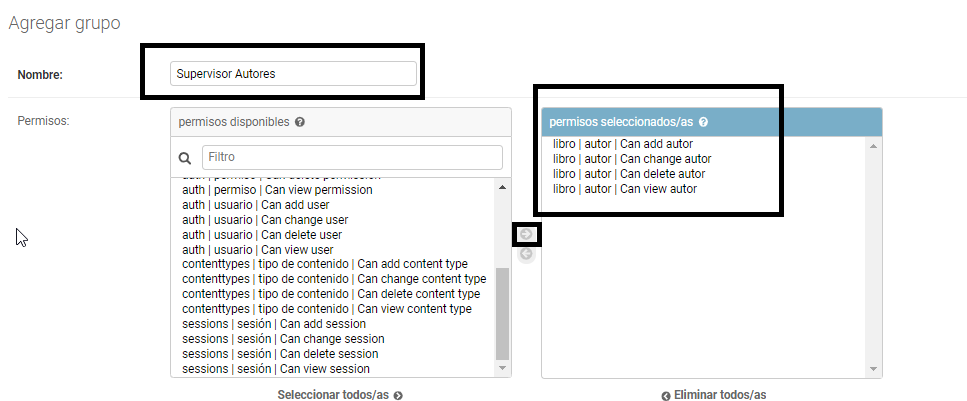
2. Grupo > Añadir Grupo:

Cuando se migra una aplicación lo primero que hace es migrar permisos. Aquí definimos los mismos junto a los usuarios.

Cuando migramos Django crea 3 permisos a los usuarios de manera automática. 1. Añadir modelo. 2. Cambiar modelo 3. Eliminar modelo.



Ahora no lo vamos a hacer pero podemos crear un grupo



3. Para que el nombre del usuario y del grupo de usuarios no nos salga con una **s** final hacemos la siguiente modificación en **models.**py:

****

class Meta:

        verbose\_name = 'Remera'

        verbose\_name\_plural = 'Remeras'

4. A medida que vaya creaciendo el nro de Remeras y se agregan sucesivamente se pueden ordenar en base a uno de los campos (alfabéticamente):

ordering = ['marca']#ordena las remeras alfabeticamente

de forma tal que **models.py** queda:

    class Meta:

        verbose\_name = 'Remera'

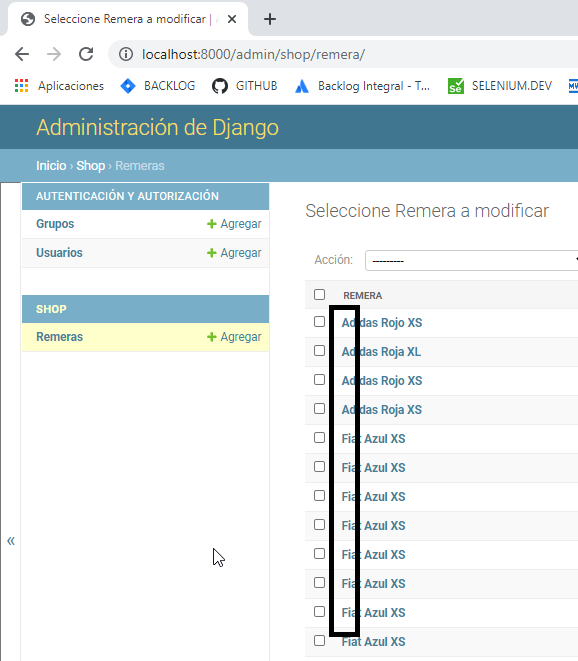
        verbose\_name\_plural = 'Remeras'

        ordering = ['marca']#ordena las remeras alfabeticamente

python manage.py makemigrations

python manage.py migrate

de forma tal que nos queda:

****

5. **Registrar una clase ModelAdmin**.

a. Para cambiar la forma en la que un modelo se despliega en la interfaz de administración, se debe definir una clase **ModelAdmin** (que describe el diseño) y registrarla con el modelo.

b. Vamos a **admin.py** y comentamos la siguiente línea:

admin.site.register(Remera)

reemplazandola por una nueva clase llamada **RemeraAdmin** la cual registramos.

#Definimos la admin class

class RemeraAdmin(admin.ModelAdmin):

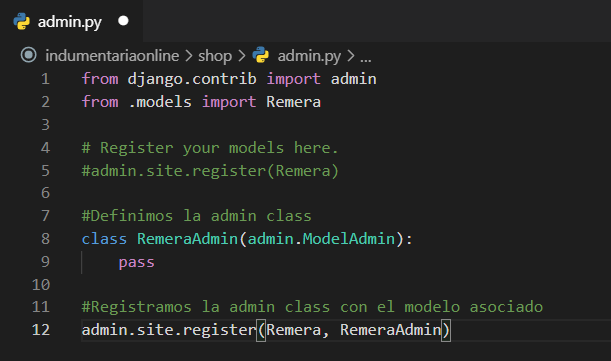
    pass

#Registramos la admin class con el modelo asociado

admin.site.register(Remera, RemeraAdmin)

**Nota:** Pass indica que nuestra clase esta vacia.

De forma tal que nos queda:



6. De esta forma ya estamos preparados para trabajar en la configuración en que se muestra el modelo en la admin console web.

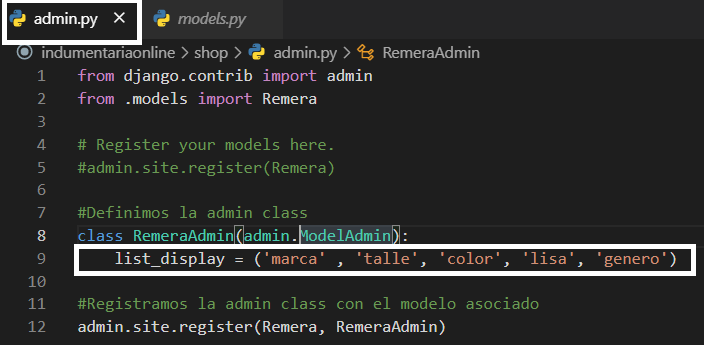
7. **Configuracion de vistas de lista:**

El ­­­­­­\_\_str\_\_() del modelo funciona bien cuando solo existen unas pocas Remeras. Pero cuando tenemos muchas remeras se puede llegar a tener duplicados. Para diferenciarlas, o simplemente mostrar información mas interesante sobre cada Remera, se puede usar un **list\_display** para añadir otros campos a la vista.

8. En la clase **RemeraAdmin** reemplazamos **pass** por el siguiente código:

list\_display = ('marca' , 'talle', 'color', 'lisa', 'genero')

de forma tal que nos queda:



9. De esta forma vemos como nos queda la salida:



10. Añadir **filtros de lista**:

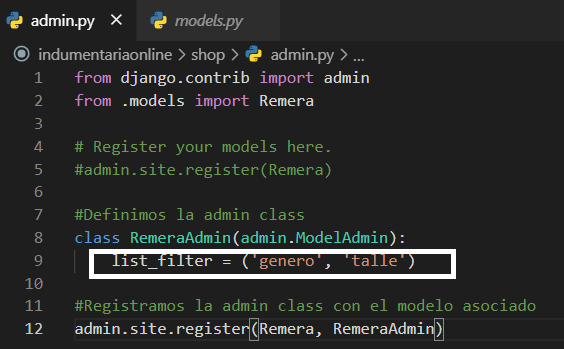
Cuando tenemos muchos ítems en una lista puede ser útil filtrar los ítems que se despliegan.

Esto se hace listando campos en el atributo **list\_filter.** Reemplazamos la clase **RemeraAdmin** con el siguiente fragmento de código:

class RemeraAdmin(admin.ModelAdmin):

    list\_filter = ('genero', 'talle')

de forma tal que nos queda:



La vista de lista incluirá ahora un cuadro de filtrado a la derecha.



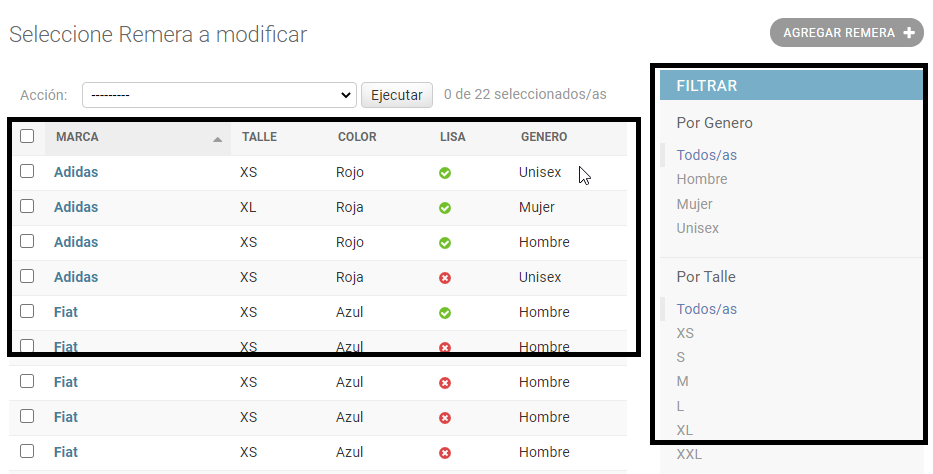
11. Podemos combinar ambas situaciones, es decir plantear una lista con un filtro:

class RemeraAdmin(admin.ModelAdmin):

    list\_display = ('marca' , 'talle', 'color', 'lisa', 'genero')

    list\_filter = ('genero', 'talle')

Obteniendo la siguiente salida:



12. Codigo Resultante:



13. Cuando queremos mostrar los campos de una determinada manera cuando se produce un ADD o un CHANGE hacemos lo siguiente:

En el modelAdmin colocamos el siguiente código:

    list\_filter = ('marca', 'talle', 'color', 'lisa', 'genero')

    fieldsets = (

        (None, {

            'fields' : ('marca', 'genero')

        }),

        ('Aqui estan los talles: ',

        {

            'fields': ('talle', 'lisa')

        }

        ),

    )

De forma tal que nos queda:

from django.contrib import admin

from .models import Remera

class RemeraAdmin(admin.ModelAdmin):

    list\_filter = ('marca', 'talle', 'color', 'lisa', 'genero')

    fieldsets = (

        (None, {

            'fields' : ('marca', 'genero')

        }),

        ('Aqui estan los talles: ',

        {

            'fields': ('talle', 'lisa')

        }

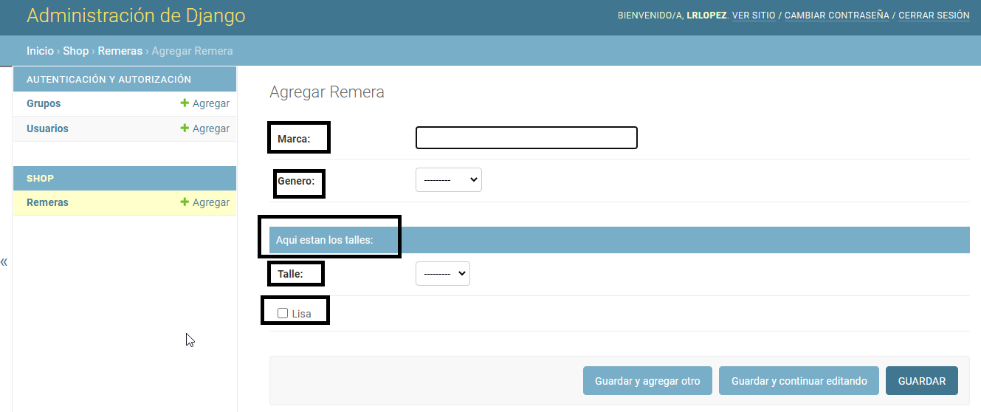
        ),

    )

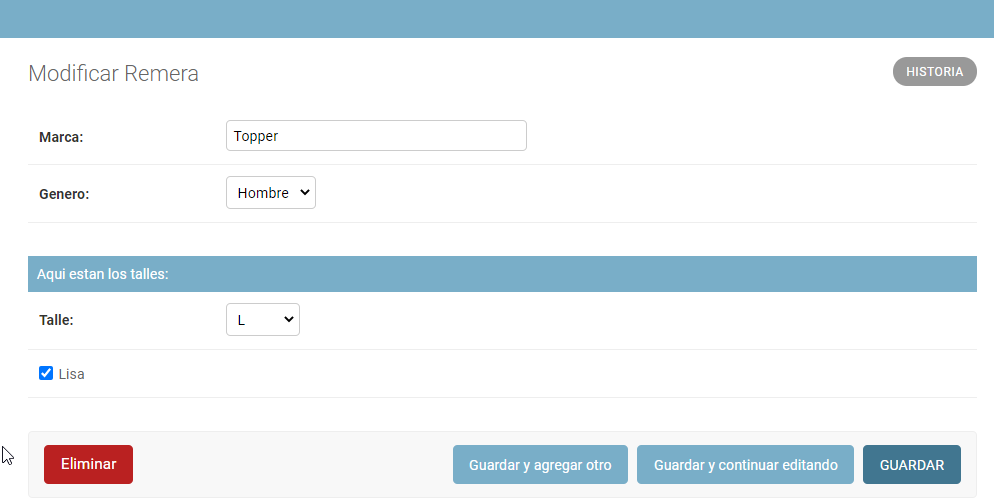
admin.site.register(Remera, RemeraAdmin)

Entonces esto afecta a cuando queremos agregar o modificar una remera.

Ej de agregar:

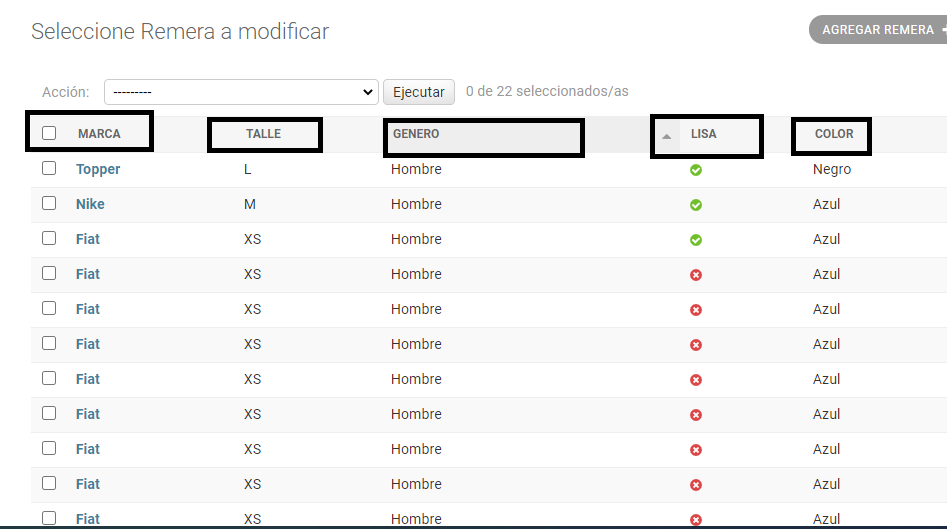


Ej de modificar:



14. Si queremos que en la pagina de “Seleccione Remera a Modificar” se vea una serie de campos ordenados en forma horizontal agregamos la siguiente línea:

list\_display = ('marca','talle','genero','lisa','color')



ahora una vez que se selecciona un objeto en particular nos aparece uno o varios campos en forma vertical que definimos de la siguiente manera (recordar hacer click en una remera):



El código resultante es:

from django.contrib import admin

from .models import Remera

class RemeraAdmin(admin.ModelAdmin):

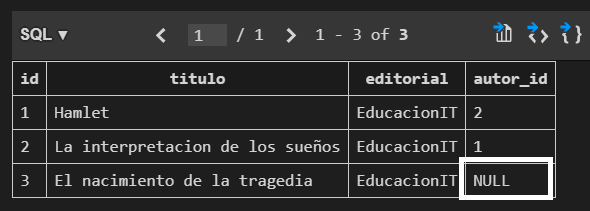
    list\_display = ('marca','talle','genero','lisa','color')

    fields = ['marca', 'talle']

admin.site.register(Remera, RemeraAdmin)

**WORKAROUND EJERCICIO\_1** error ocurrido al borrar el autor Nietzche, Friederich.

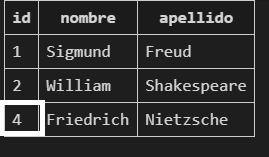
El problema era que como habíamos borrado el autor, entonces la tabla quedo con un NULL:



Entonces para solucionarlo le hice un update a la tabla:

update myapp\_libro set autor\_id=4 WHERE id = 3

puse autor\_id=4 porque como el id de la tabla myapp\_autor es autoincremental al borrar y luego crear el autor Nietzsche Friedrich le creo un nuevo id, veamos la tabla de autor:



Entonces al realizar el update previo se soluciono el tema y se pudo cargar nuevamente la pagina:



Codigo Resultante:

