
AWAKELAB
Desarrollo Web Python con Django
Desarrollador de aplicaciones
Full Stack
Python Trainee

CONTROL DE SEGURIDAD DE ACCESO CON DJANGO

Contenido 1: Implementar módulo de administración de usuarios y permisos utilizando el módulo pre construido de Django para este fin

Objetivo de la jornada

1. Reconoce las características básicas del sitio administrativo de Django y su utilidad en el modelo Auth
2. Implementa el módulo administrativo de Django para la administración de usuarios
3. Opera el módulo administrativo de Django realizando la administración de usuarios y grupos

7.1.1.- El sitio administrativo de Django

Cuando la gente pregunta: "¿Cuáles son los beneficios de Django sobre otros Frameworks web?" el sitio administrativo es lo que suele venir a la mente.

Imagínese que cada torta viniera con una interfaz de administración. No solo podrá ver la lista de ingredientes, sino también agregar/editar/eliminar ingredientes. Si alguien está jugando con su torta de una manera que no le gusta, puede limitar o revocar su acceso.

Aunque este ejemplo parezca bastante surrealista, eso es lo que sienten los desarrolladores web que vienen de otros orígenes cuando usan por primera vez la interfaz de administración de Django. Le brinda mucho poder sobre su aplicación web automáticamente, con poco trabajo requerido.

La interfaz de administración de Django está diseñada para administradores de sitios, no para usuarios. Es un lugar para que los administradores de su sitio agreguen/editen/eliminen datos y realicen tareas de administración del sitio.

Aunque es posible convertirlo en algo que sus usuarios puedan usar, realmente no debería hacerlo. Simplemente no está diseñado para que lo utilicen todos los visitantes del sitio.

Para tener el sitio de administración disponible, debemos tener disponibles en **INSTALLED_APPS** las siguientes dependencias

- `django.contrib.auth`
- `django.contrib.contenttypes`
- `django.contrib.messages`
- `django.contrib.sessions`

En la variable **TEMPLATES** bajo **OPTION** y **context_processors** debemos tener configurados:

- `django.template.context_processors.request`
- `django.contrib.auth.context_processors.auth`
- `django.contrib.messages.context_processors.messages`

La variable lista **MIDDLEWARE** debería contener los siguientes valores:

- `django.contrib.auth.middleware.AuthenticationMiddleware`
- `django.contrib.messages.middleware.MessageMiddleware`

Por último necesitamos que **urls.py** del proyecto contengan la ruta hacia el sitio de administración:

```
urlpatterns = [
    path'admin/', admin.site.urls),
    …
]
```

Usualmente todos estos parámetros están configurados por defecto si iniciamos nuestro proyecto utilizando **startproject**

Ahora podemos dirigirnos en nuestro browser a http://127.0.0.1:8000/admin/

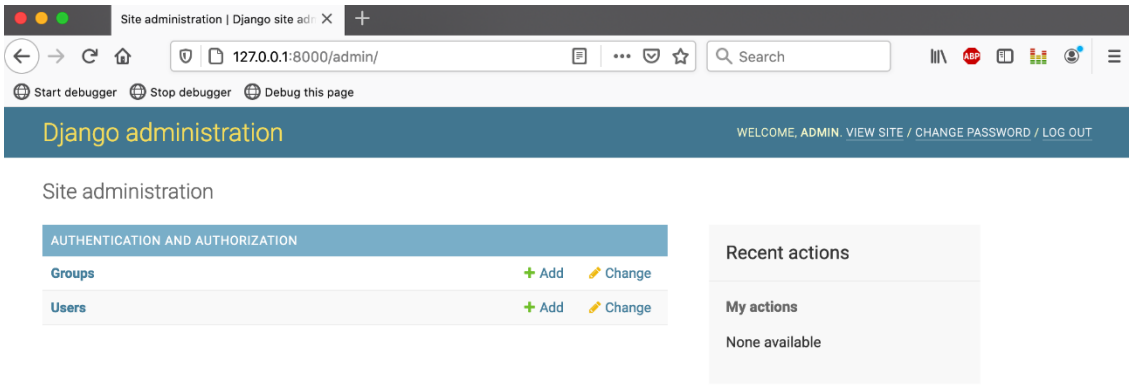
7.1.1.1 Creando usuarios

Antes que nada, necesitamos crear un super usuario, lo cual hacemos con el comando:

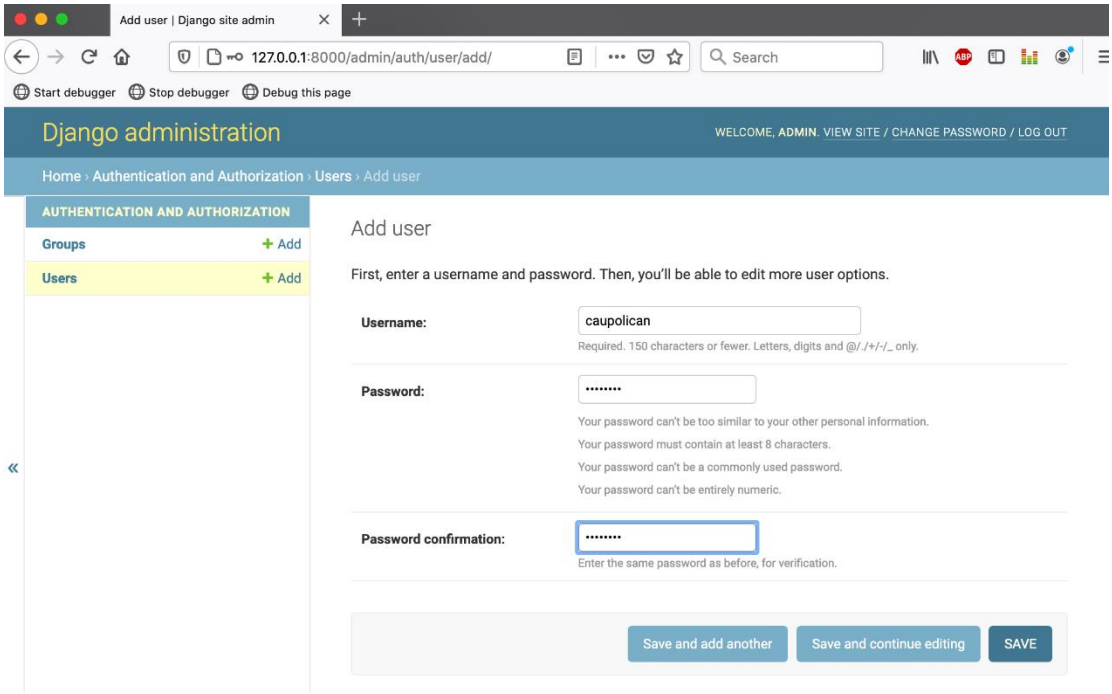
```
(env) $ python manage.py createsuperuser
Username (leave blank to use 'aschwarzenegger'): admin
Email address: ad@m.in
Password:Password (again):
```

```
...
Superuser created successfully.
```

Con nuestras credenciales podemos ingresar al sitio de administración



Ahora con la interfaz proporcionada por el sitio de administración podemos crear usuarios estándar:



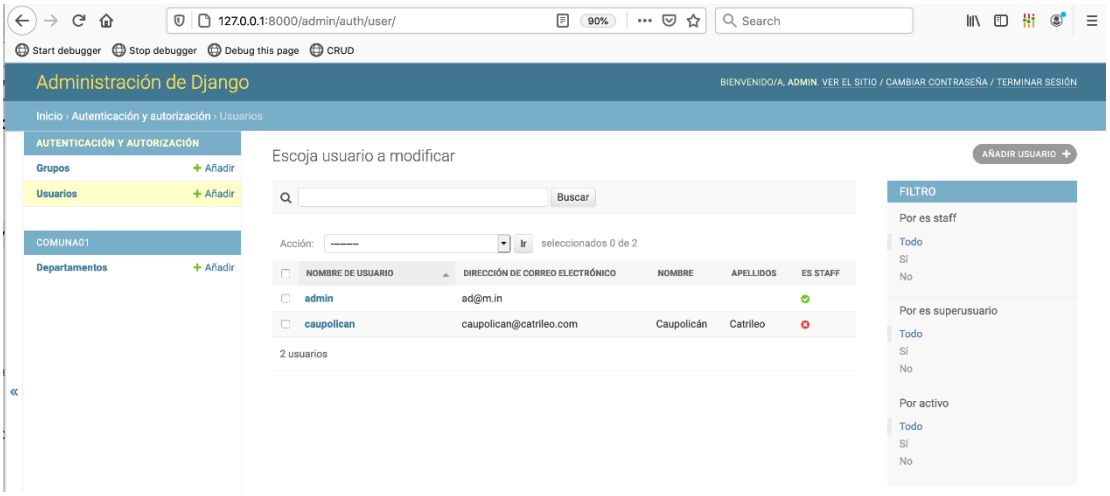
Más adelante veremos cómo los usuarios pueden cambiar sus contraseñas. Por lo general, no vale la pena personalizar mucho el sitio de administración de Django. A veces, crear una vista simple o un formulario desde cero da como resultado la misma funcionalidad deseada con mucho menos trabajo. Siempre hemos tenido mejores

resultados con la creación de paneles de administración personalizados para proyectos de clientes que con la modificación del administrador para que se ajuste a las necesidades del cliente.

Hemos visto algunas características de este sitio de administración, entre otras con la ayuda de esta utilidad podemos:

- Agregar y eliminar usuarios
- Editar usuarios existentes
- Restablecer contraseñas de usuario
- Asignar estado de personal y/o superusuario a un usuario
- Agregar o eliminar permisos de usuario
- Crear grupos de usuarios; y
- Agregar usuarios a un grupo

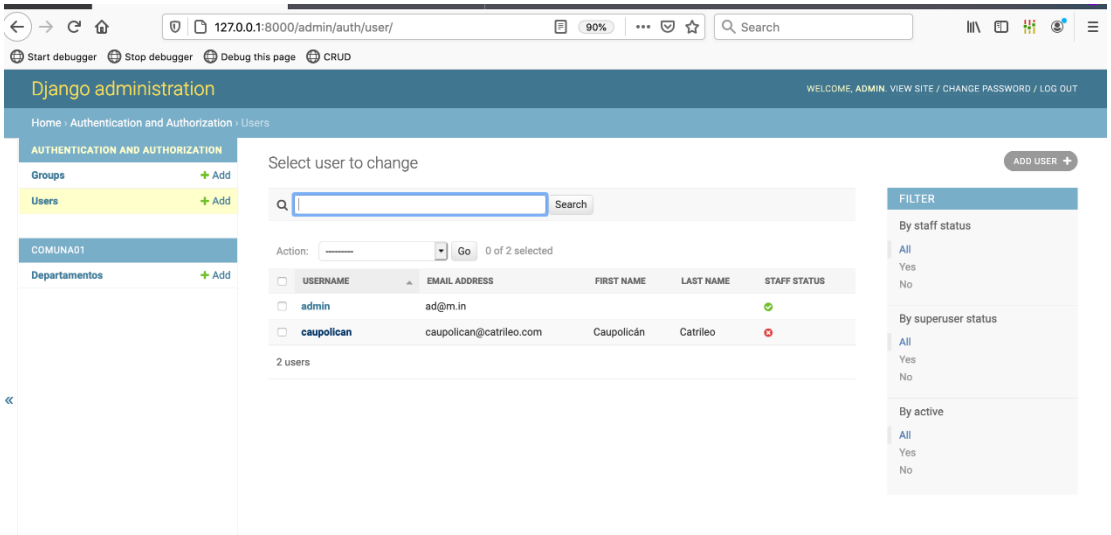
Nota: en este curso, dado que el framework está documentado/escrito en inglés (al igual que Python) hemos tratado de conservar este lenguaje en la mayoría de las vistas. No obstante, haciendo el cambio en **settings.py**: **LANGUAGE_CODE = 'es-cl'** usted podría ver el sitio de administración traducido al español chileno, ver figura:



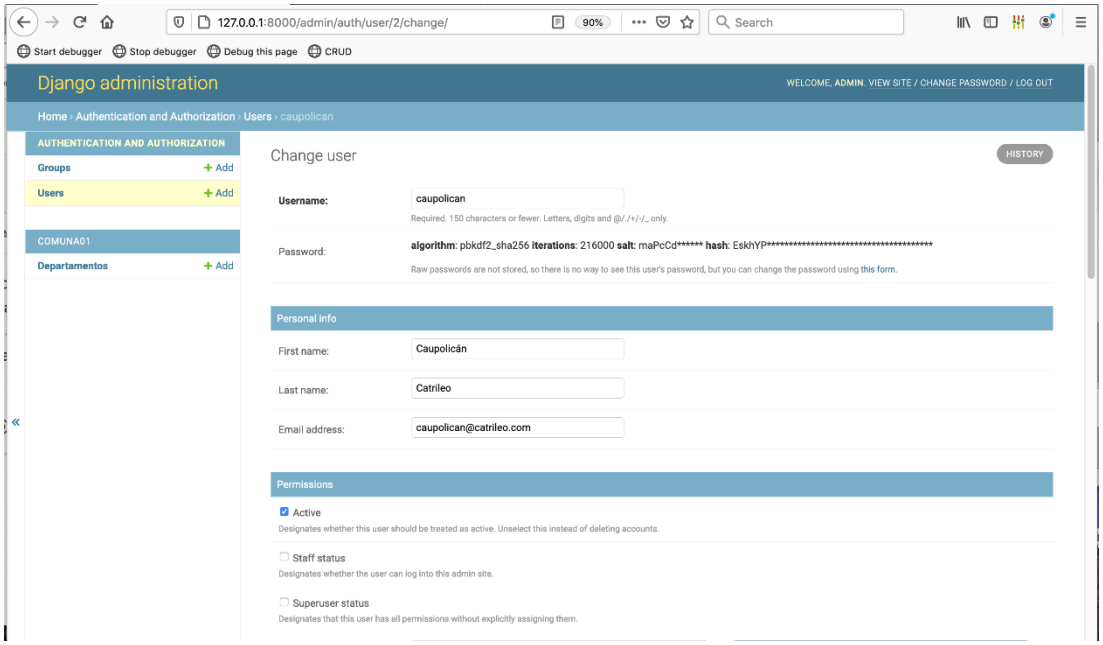
Manejo de permisos por usuario en Página administración de Django

¿Recuerda cuando creamos un superusuario administrador? Este usuario especial tiene acceso completo a todos los modelos en el sitio de administración y puede agregar, cambiar y eliminar cualquier registro de modelo. En una aplicación real, querrá limitar la cantidad de usuarios que tienen acceso completo a su sitio.

Agregar una nueva usuaria/usuario es fácil como ya vimos.



En la parte superior de la pantalla de edición del usuario, verá opciones para editar la contraseña y la información personal del usuario. Desplácese hacia abajo hasta la sección Permisos y asegúrese de que el estado del personal esté marcado y el estado de superusuario no esté marcado. Como aparece en la figura de abajo para nuestro usuario Caupolicán Catrileo.

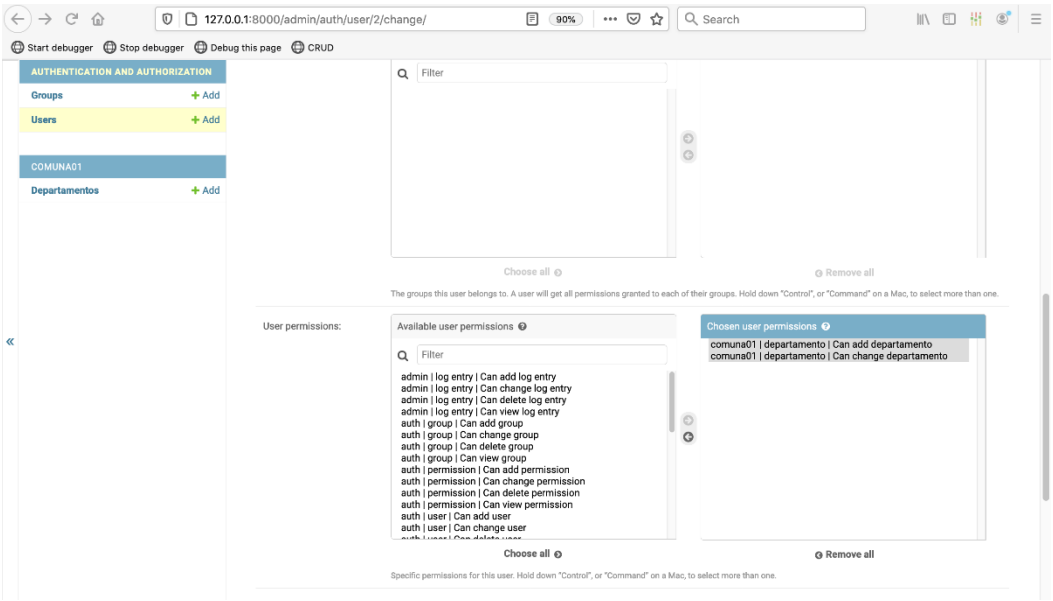


Lo que creamos aquí es un usuario administrador normal. Otorgamos a los usuarios administradores normales, es decir, -miembros activos del personal que no son superusuarios- acceso de administrador a través de permisos asignados. Cada objeto editable a través de la interfaz de administración tiene cuatro permisos: un permiso de creación, un permiso de visualización, un permiso de edición y un permiso de eliminación.

Permisos de modelo: Tenga en cuenta que estos permisos se definen por modelo, no por objeto. Por ejemplo, a un usuario se le pueden asignar permisos para cambiar cualquier cotización, pero no para cambiar cotizaciones enviadas por un cliente individual. Los permisos por objeto son más complicados y están fuera del alcance de este libro, pero se tratan en la documentación de Django (https://docs.djangoproject.com/en/3.0/topics/auth/customizing/)

Lo haremos ahora: crearemos un usuario parte del "staff" (marque esta casilla en permisos) con permiso para agregar y editar departamentos del sitio, pero no para eliminarlos. Desplácese hacia abajo en la página de edición hasta el panel de permisos de usuario y agregue los siguientes permisos usando el filtro horizontal:

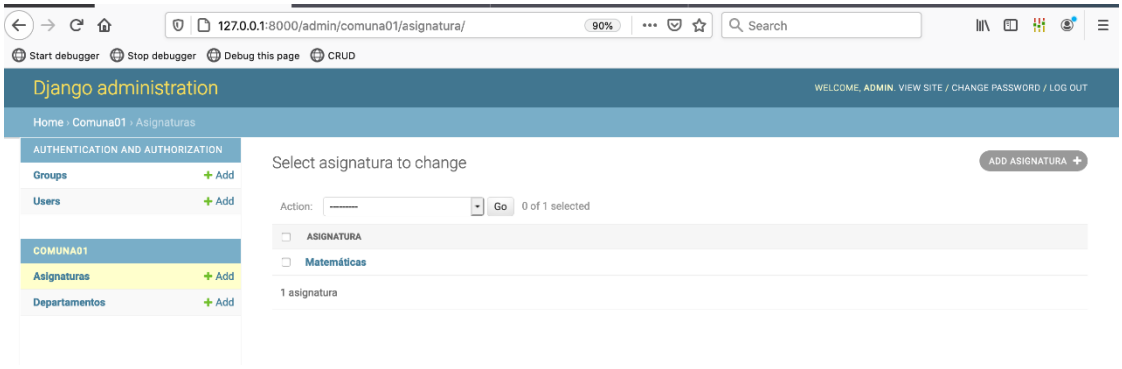
```
comuna01 | departamento | Can add departamento
comuna01 | departamento | Can change departamento
```



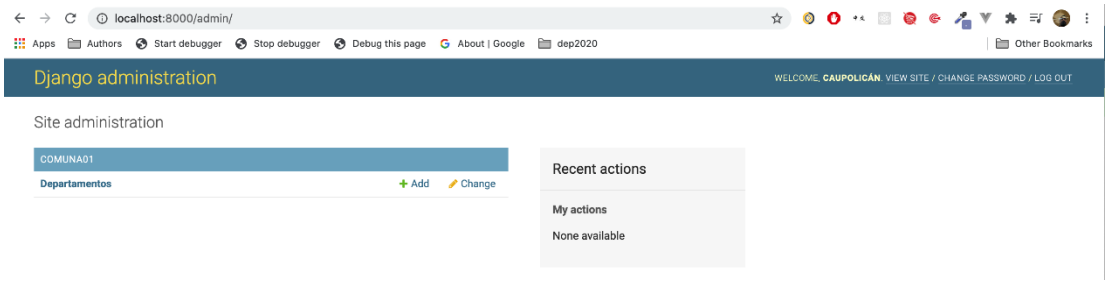
Además, agregaremos el modelo "Asignaturas", en **admin.py**, para que el ejemplo sea más claro, haga las siguientes modificaciones:

```
from .models import Departamento, Asignatura
...
admin.site.register(Asignatura,)
```

Si desea puede agregar contenido a esta categoría



Una vez que haya agregado los permisos, cierre la sesión y vuelva a iniciarla como el nuevo usuario que es parte del staff en http://localhost:8000/admin



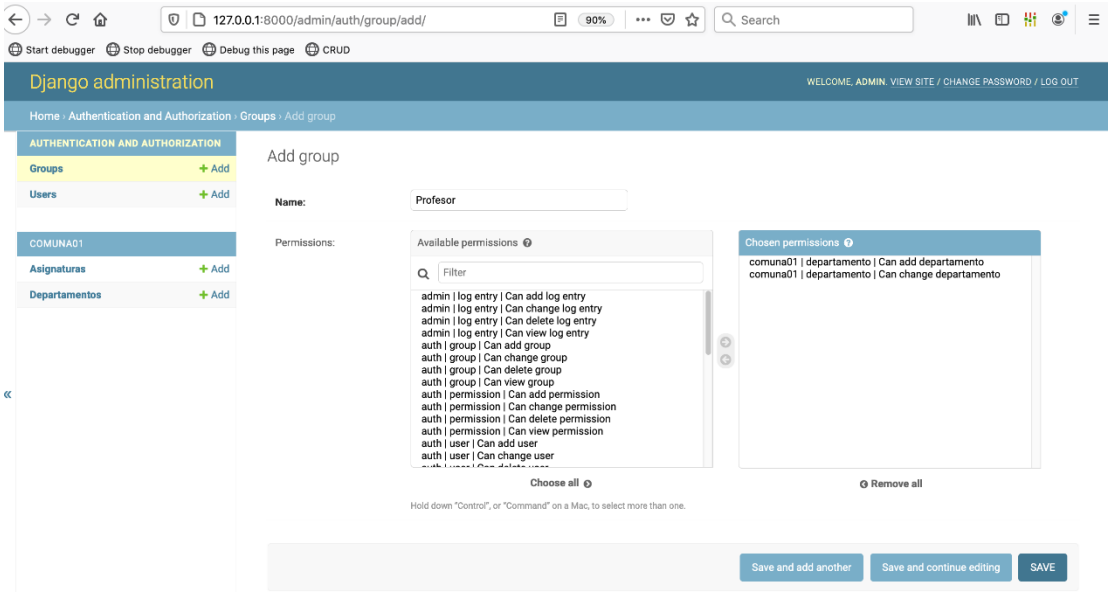
Nuestro usuario parte del staff (Caupolicán Catrileo en este caso) solo ve Departamentos y no tiene capacidades para eliminarlos, solo puede agregar y editar. Además, no es capaz de ver "Asignaturas".

7.1.1.2. Creando grupos

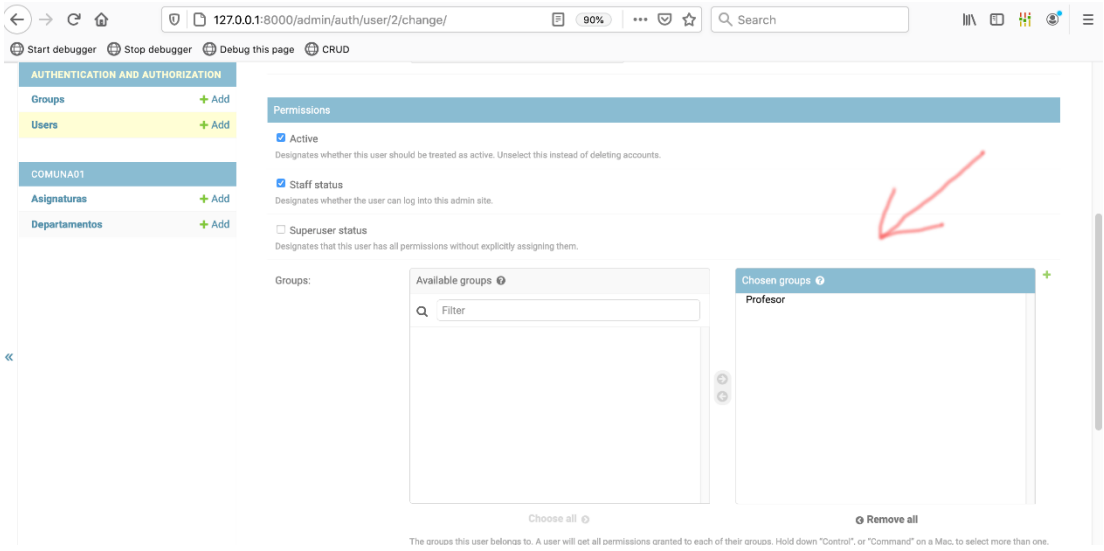
Lo que acabamos de hacer es bastante fácil, pero ¿qué sucede si tiene muchos profesores, por ejemplo, que desea agregar como usuarios? Lleva mucho tiempo agregar permisos de uno en uno para cada usuario. Afortunadamente, Django le permite crear grupos de usuarios, que es un conjunto de permisos que puede agregar a un usuario como grupo, en lugar de uno a la vez.

Creemos un grupo de profesores. Primero deberá cerrar la sesión del usuario que estaba usando y volver a iniciarla como usuario administrador.

Crear un grupo es como crear un usuario: vaya a la página principal del administrador y haga clic en el botón verde Agregar a la derecha de la lista de Grupos. Nombre su nuevo grupo Profesor, agregue los permisos del filtro horizontal y guarde su nuevo grupo. Ver figura:



Una vez que haya agregado el grupo, puede volver al usuario y editar sus permisos para agregar el nuevo grupo.



Ahora, cuando cierre la sesión y vuelva a iniciarla como usuario Profesor, tendrá la misma vista restringida del administrador que vimos.

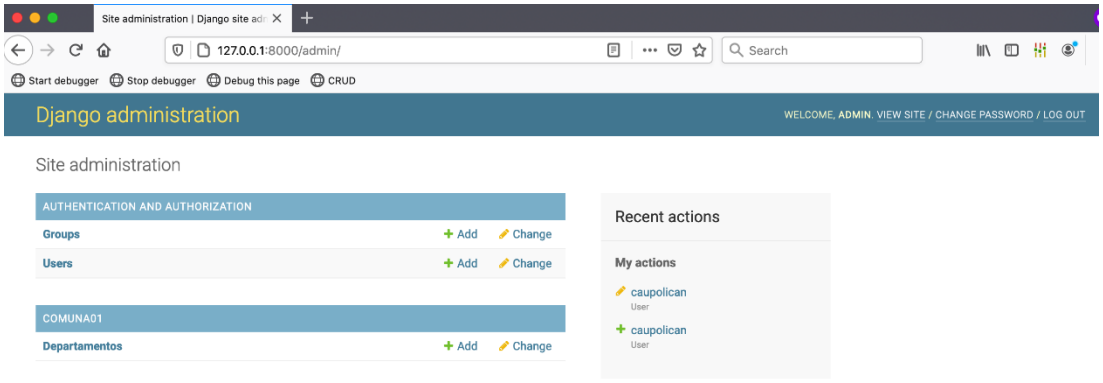
Eso es todo para agregar usuarios y permisos de usuario en el administrador. Naturalmente para sentirse cómodo con la interfaz es necesario crear y borrar algunos.

7.1.1.3. Utilizando nuestros modelos

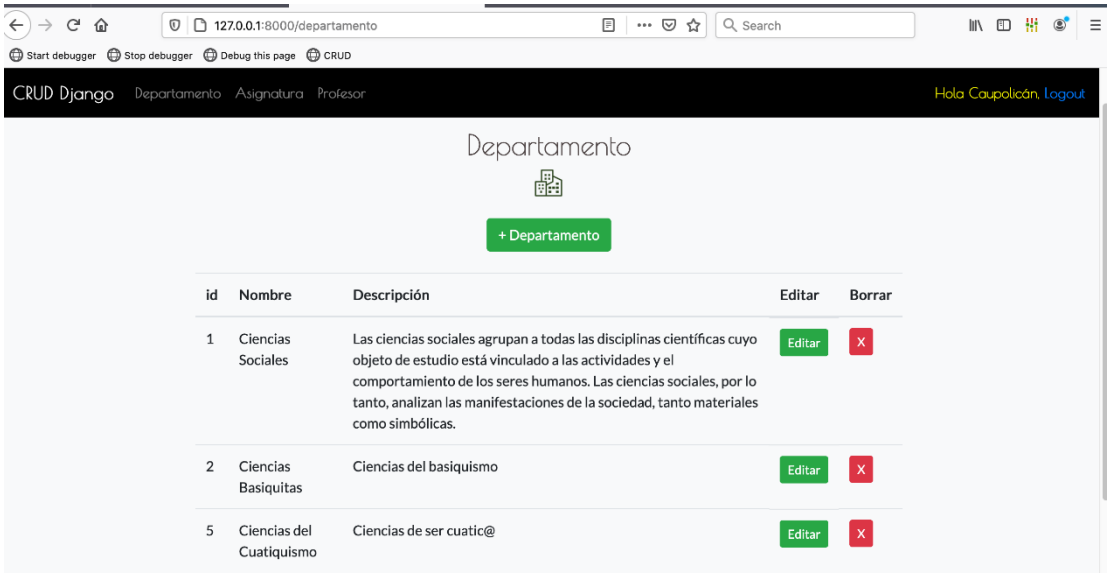
Hemos trabajado con nuestra propia aplicación y modelos, queremos contar con facilidades administrativas para intervenir nuestros modelos, tal fin lo podemos conseguir registrando nuestros modelos en **admin.py**:

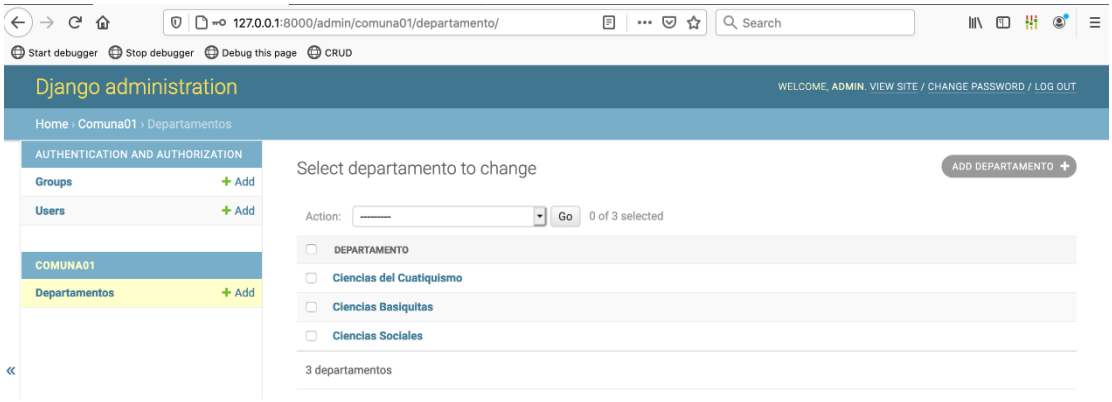
```
from django.contrib import admin
from .models import Departamento

admin.site.register(Departamento)
```

Luego, si vamos al sitio de administración veremos que nuestra aplicación y modelo son agregados en la parte inferior de panel.



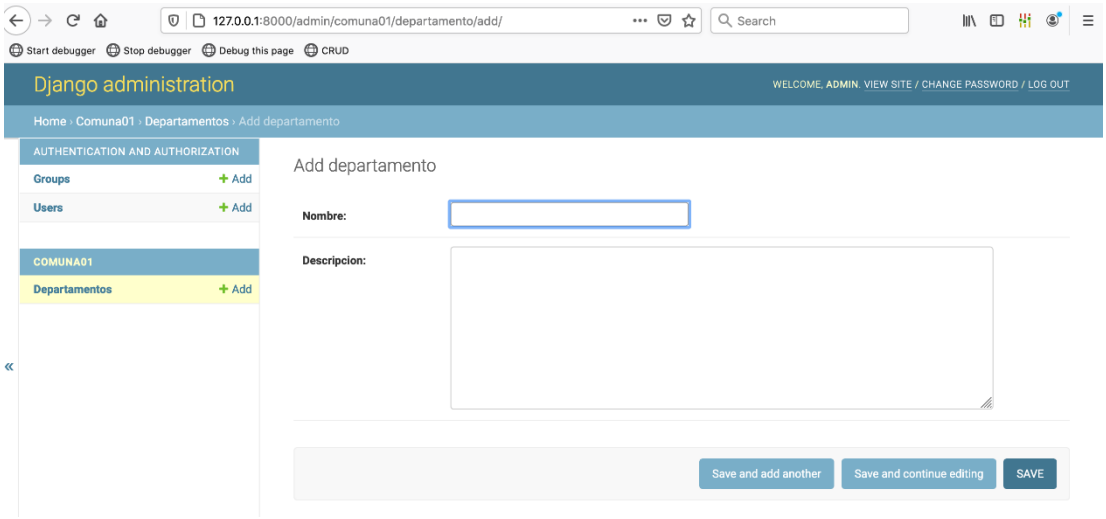
Las categorías que hemos creado en nuestra página para agregar Departamentos deberían ser las mismas que aparecen en nuestro panel cuando ingresamos a "Departamentos":



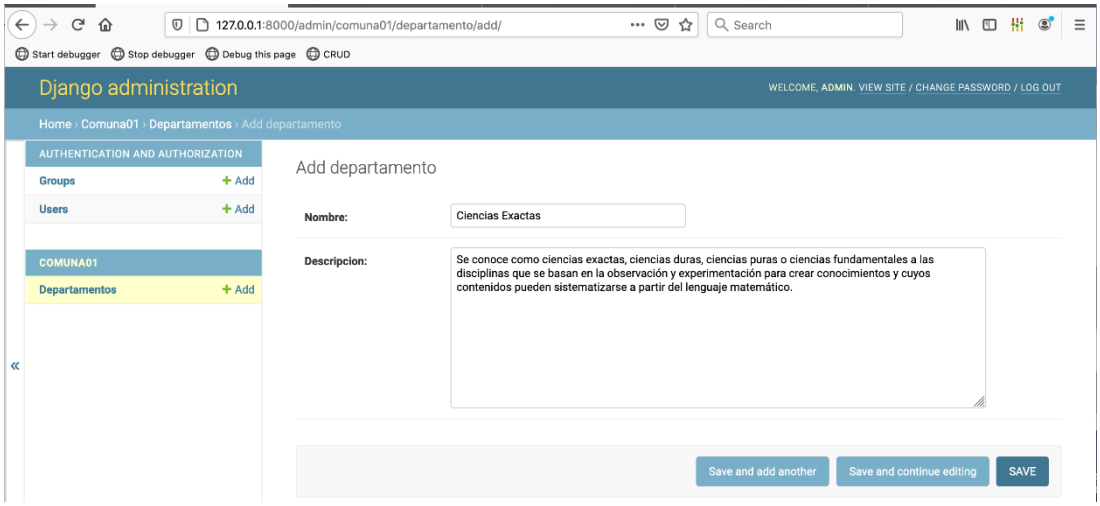


El sitio de administración está diseñado para ser utilizado por usuarios no técnicos, por lo que debe ser razonablemente auto explicativo. Sin embargo, veamos algunas de las características básicas. Cada tipo de datos en el sitio de administración de Django tiene una lista de cambios y un formulario de edición.

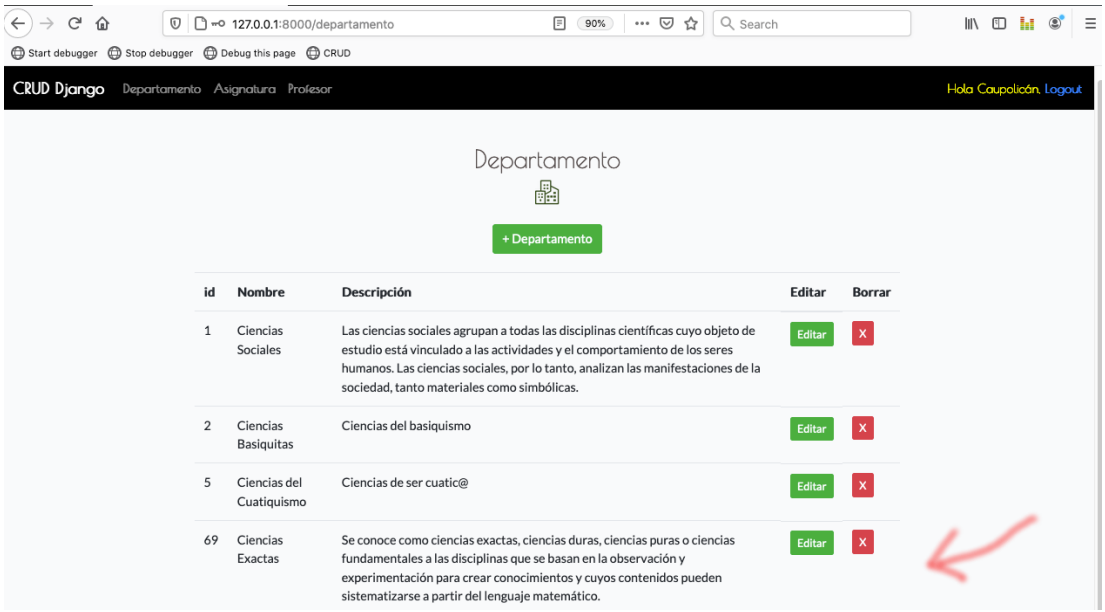
Las listas de cambios le muestran todos los objetos disponibles en la base de datos y los formularios de edición le permiten agregar, cambiar o eliminar registros en su base de datos. La Figura abajo muestra el formulario de edición que se abre cuando se hace clic en el enlace Add (Agregar).



Agregaremos un Departamento "Ciencias Exactas" a modo de ejemplo:



La nueva adición queda disponible como si la hubiésemos hecho en nuestro formulario disponible para el usuario común:

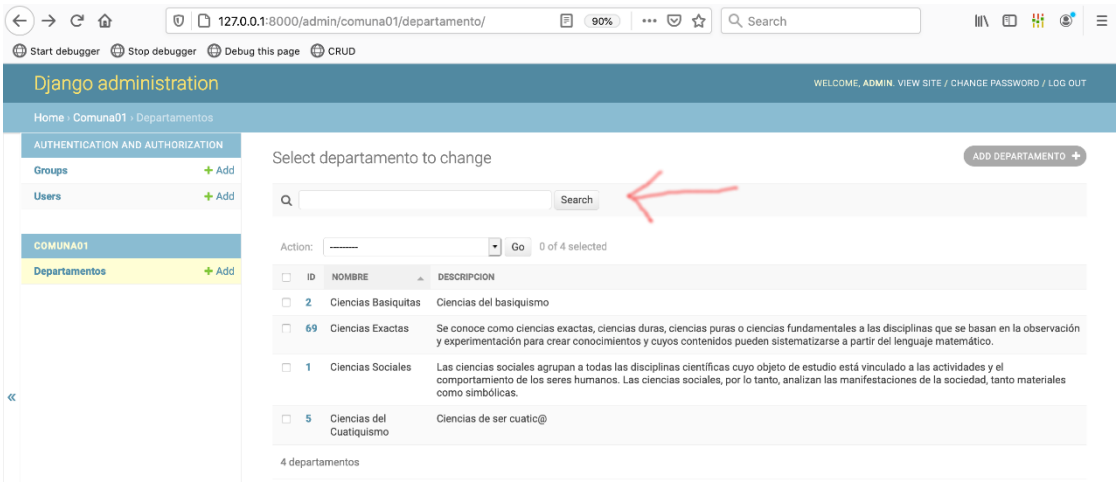


Ahora vamos a suponer que necesitamos buscar palabras precisas en la descripción de nuestros departamentos, podemos tener muchos y quizás esta búsqueda sea difícil de hacer solo mirando las descripciones, para tales efectos sería sin lugar a dudas bueno contar con un mecanismo de búsqueda y ordenar los "nombres" alfabéticamente. Haremos modificaciones a nuestro archivo **admin.py** para dejarlo de esta forma:

```
from django.contrib import admin
from .models import Departamento


class DepartamentoAdmin(admin.ModelAdmin):
```

```
    list_display = ('id', 'nombre', 'descripcion',)
    ordering = ('nombre',)
    search_fields = ('descripcion',)


admin.site.register(Departamento, DepartamentoAdmin)
```

Tales modificaciones nos permiten obtener lo que buscamos:



Nota: Django nos ofrece una tabla donde están definidos permisos de acceso para nuestra aplicación, así como también otras tablas que pueden ser examinadas si quiere seguir el código para inspeccionar las aplicaciones que se instalan por defecto con mayor detalle.

```
municipio01=> SELECT * FROM auth_permission;
 id |      name                | content_type_id |      codename
----+--------------------------+-----------------+--------------------
...
25 | Can add departamento     |               7 | add_departamento
26 | Can change departamento  |               7 | change_departamento
27 | Can delete departamento  |               7 | delete_departamento
28 | Can view departamento    |               7 | view_departamento
```

7.1.1.4. Utilizando manage.py

En la última clase del módulo anterior creamos un pequeño comando al estar inspeccionando las sesiones, el cual ejecutamos con la ayuda **manage.py**. Ahora, por completitud, veremos algunas un poco que hay detrás de estas utilidades de consola.

Nota: **manage.py** es equivalente a **django-admin** excepto por la variable de configuración **DJANGO_SETTINGS_MODULE**. En nuestro caso en **manage.py**

podemos encontrar la línea **os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'municipio01.settings')**

La mejor forma de saber cómo nos puede ayudar esta utilidad es ver que tiene disponible

```
$ python manage.py help

Type 'manage.py help <subcommand>' for help on a specific subcommand.

Available subcommands:

[auth]
    changepassword
    createsuperuser

[comuna01]
    leer_sesion

[contenttypes]
    remove_stale_contenttypes
...
```

La mayoría de los comandos son auto explicativos y/o tienen muy buena documentación. Si se da cuenta nuestro comando **leer_sesion** aparece en la lista bajo **comuna01**, eso le indica que cada uno de estos comandos está vinculado con una aplicación correspondiente. Recuerde que nuestro comando es solo para descubrir si alguien trató de escribir "rechazo" en departamentos, no hemos abordado cada caso, por lo tanto, si lo corre a estas alturas lo más probable es que encuentre un error, es sólo un comando ilustrativo que usamos para un caso particular, lo importante es el concepto. No obstante, le haremos unas pequeñas modificaciones ilustrativas.

Para lo cual usaremos un poco de TDD o desarrollo guiado por pruebas (Test-driven development). Cuando iniciamos nuestro proyecto el archivo **tests.py** fue creado automáticamente, en tal archivo pondremos nuestro código de pruebas.

Primero veremos si nuestro entorno de pruebas funciona adecuadamente, escribiendo un test "de juguete".

```
from django.test import TestCase


class Comuna01TestCase(TestCase):

    def test_dummy(self):
        self.assertTrueTrue)
```

Es probable que al correr este test (**$ python manage.py test**, si quiere ser totalmente específicopuede usar **$ python manage.py test comuna01.tests.Comuna01TestCase**) vea:

```
Creating test database for alias 'default'...
Got an error creating the test database: permission denied to create
database
```

Cuando Django ejecuta el conjunto de pruebas, crea una nueva base de datos. El usuario de postgres con el nombre de **usuario alcalde** no tiene permiso para crear una base de datos, de ahí el mensaje de error. Cuando ejecuta **migrate**, por ejemplo, Django no intenta crear una base de datos, por lo que no obtiene ningún error.

Puede agregar el permiso crear base de datos al usuario que tiene configurado en **settings.py** ejecutando el siguiente comando en el shell de postgres como *8superusuario**:

```
$ psql -d municipio01
municipio01=# ALTER USER alcalde CREATEDB;
ALTER ROLE
```

Ahora si trata de correr los tests no deberían existir problemas:

```
$ python manage.py test comuna01.tests.Comuna01TestCase
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.003s

OK
Destroying test database for alias 'default'...
```

Lo importante es nuestro test de juguete que comprueba que "True es igual a True" (por eso es de juguete) funciona lo vemos en las líneas:

```
Ran 1 test in 0.003s
OK
```

Nota: la base de datos se crea una vez que comienzan los tests y cuando estos finalizan se destruye, por eso no se ve listada en postgres.

Sabemos que en nuestro sitio los usuarios "loggeados" pueden realizar acciones, por lo tanto, primero que nada, necesitamos un usuario capaz, y lo probaremos así:

```
from django.contrib.auth.models import User
...
class Comuna01TestCase(TestCase):
            def test_login(self):
        user = User.objects.create(username='testuser')
        user.set_password('12345')
        user.save()

        c = Client()
        logged_in = c.login(username='testuser', password='12345')
        self.assertTrue(logged_in)
```

El test debería ser exitoso si usted lo ejecuta.

Nota: este no es un curso de desarrollo basado en tests TDD, lo cual demandaría mucho más tiempo, amén de que nos vamos a saltar algunos pasos intermedios, vamos a apuntar más a lo que funcional.

Como nuestro test fue exitoso sería bueno contar con este usuario para todo test que queramos y requiera un usuario autenticado por lo tanto es un buen candidato para utilizarlo en **setUp** que es ejecutado antes de cada test. Dado lo anterior vamos a refactorizar nuestro test:

```
    ...
def setUp(self) -> None:
    user = User.objects.create(username='testuser')
    user.set_password('12345')
    user.save()
    self.cliente = Client()
    self.logged_in = self.cliente.login(username='testuser',
password='12345')

def test_login(self):
    self.assertTrue(self.logged_in)
```

Recordemos nuestro comando **leer_sesion**:

```
from django.core.management.base import BaseCommand
from django.contrib.sessions.models import Session
from django.utils import timezone

class Command(BaseCommand):
```

```
    help = "Buscamos sesiones sospechosas..."

    def handle(self, *args, **options):
        sessions = Session.objects.filter(expire_date__gt=timezone.now())
        for session in sessions:
            data = session.get_decoded()
            print("Incidente a las ", data["incidente"])
```

Lo que vamos a probar es que si alguien escribió "Rechazo" en Departamento se debería producir un Incidente informado en pantalla. Este es nuestro test:

```
def test_alguien_escribio_rechazo_en_departamentos(self):
    # enviando un request con el usuario ya autenticado
    # contiene la palabra "rechazo"
    self.cliente.post('/departamento',
            {'nombre': 'Departamento de rechazo',
             'descripcion': 'Probando probando 1 2 3'})
    departamentos = Departamento.objects.get()
    self.assertEquals("Probando probando 1 2 3", departamentos.descripcion)
# probando el comando que creamos anteriormente
    with open('/tmp/inspectdb', 'w+') as f:
        call_command('leer_sesion', stdout=f)
        var = f.readlines()
    var = " ".join(var)
    self.assertIn(var, "Incidente")
```

Guardamos la información de la consola en un archivo y luego lo leemos.

Ahora vamos a arreglar nuestro comando para que no se produzcan errores si no detectamos incidentes, nuestro test para tales efectos es:

```
def test_nadie_escribio_rechazo_en_departamentos(self):
    # enviando un request con el usuario ya autenticado
    # contiene la palabra "rechazo"
    self.cliente.post('/departamento',
                      {'nombre': 'Departamento de apruebo',
                       'descripcion': 'Probando probando 5 6 7'})
    departamentos = Departamento.objects.get()
    self.assertEquals("Probando probando 5 6 7", departamentos.descripcion)
# probando el comando que creamos anteriormente
    with open('/tmp/inspectdb', 'w+') as f:
        call_command('leer_sesion', stdout=f)
        var = f.readlines()
    var = " ".join(var)
    print(var)
    self.assertIn(var, "Todo bien hasta ahora")
```

A partir de este test podemos refactorizar nuestro codigo para el comando:

```
class Command(BaseCommand):
    help = "Buscamos sesiones sospechosas..."

    def handle(self, *args, **options):
        sessions = Session.objects.filter(expire_date__gt=timezone.now())
        for session in sessions:
            data = session.get_decoded()
            if "incidente" in data:
                print("Incidente a las ", data["incidente"])
            else:
                print("Todo bien hasta ahora")
```

7.1.2. Referencias

[1] Django Documentation
https://docs.djangoproject.com/en/3.1/

[2] D. Feldroy & A. Feldroy , Two Scoops of Django 3.x, Best Practices for the Django Web Frame-work, 2020.

[3] William S. Vincent, Django for Beginners Build websites with Python & Django, 2020.