

## EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: ACTIVANDO EL SITIO ADMINISTRATIVO DE DJANGO.
- EXERCISE 2: CAMPOS ESPECIALES.

### EXERCISE 1: ACTIVANDO EL SITIO ADMINISTRATIVO DE DJANGO.

El objetivo del presente ejercicio es plantear una guía paso a paso del panel de administración que contiene funciones básicas, tales como: activar un modelo para luego crear, leer, editar y eliminar modelos, y otros campos especiales para la gestión en el panel de administración del Framework Django.

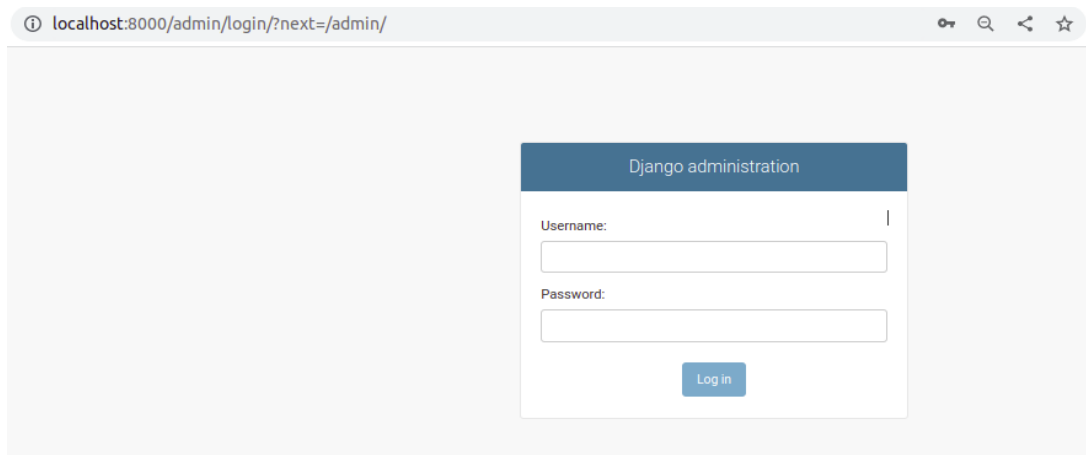
Requisitos previos:

- Tener conocimiento de un terminal o línea de comando e instalación de paquetes de software en el sistema operativo tanto en Windows 10, como en Linux (en este caso, Linux Ubuntu donde se desarrollará la práctica).
- En caso de Windows, se debe tener instalado el entorno virtual explicado en el Exercises del CUE03, y la configuración del proyecto en el CUE04.
- Tener previamente instalado la versión de Python 3, y el entorno virtual con (virtualenvwrapper).
- Hacer uso de la herramienta Visual Studio Code.

### ACTIVANDO EL SITIO ADMINISTRATIVO DE DJANGO

Existen dos formas de añadir registros a nuestra base de datos. La primera es crear las instancias manualmente, a través de formularios en vistas; y la otra es utilizar el panel de administrador. En esta parte haremos uso del panel de administrador, sabiendo que hemos realizado pequeños formularios para el registro de modelos.

El panel de administrador de Django es una funcionalidad que viene creada por defecto. Para acceder tenemos que entrar a la dirección: <http://localhost:8000/admin> de nuestro sitio.



Previamente, debemos tener configurado en la `urls.py` del proyecto, que está por defecto para ingresar en esa dirección como si se tratara de otra aplicación:

## SITE\_WEB\_DJANGO/URLS.PY

```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('', include('boards.urls')),
7
8 ]
```

Debemos asegurarnos que el archivo `settings.py` esté instalado, que lo trae por defecto al iniciar un proyecto Django:

## SITE\_WEB\_DJANGO/SETTINGS.PY

```
1 INSTALLED_APPS = [
2     'django.contrib.admin',
3     'django.contrib.auth',
4     'django.contrib.contenttypes',
5
6
7 MIDDLEWARE = [
```

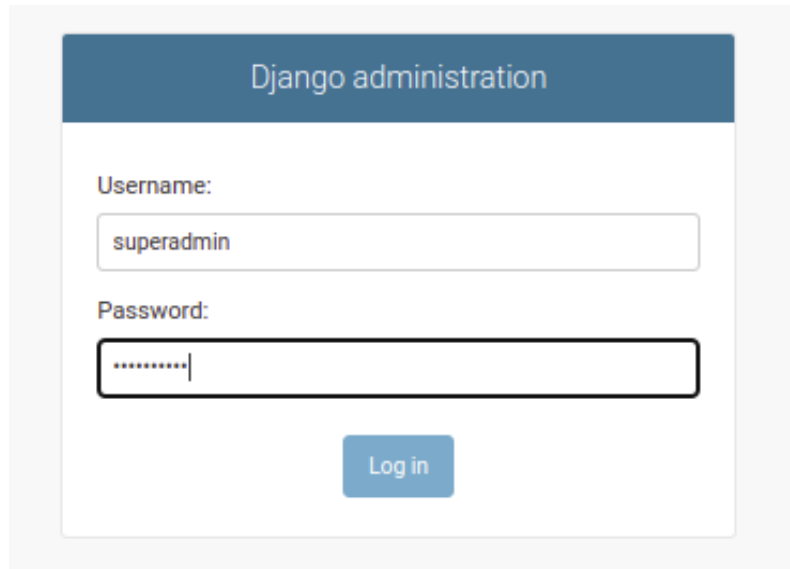
```
8 'django.middleware.security.SecurityMiddleware',
9 'django.contrib.sessions.middleware.SessionMiddleware',
10 'django.middleware.common.CommonMiddleware',
11
12
13 TEMPLATES = [
14     {
15         'BACKEND': 'django.template.backends.django.DjangoTemplates',
16         'DIRS': [BASE_DIR / 'templates'],
17         'APP_DIRS': True,
18         'OPTIONS': {
19             'context_processors': [
20                 'django.template.context_processors.debug',
21                 'django.template.context_processors.request',
22                 'django.contrib.auth.context_processors.auth',
23
```

Al corroborar que están las dependencias activas, procedemos a crear un nuevo usuario super user. Se recomienda no utilizar nunca el nombre de usuario admin más allá del desarrollo, pues es mejor definir uno por defecto personalizado:

## CONSOLA / TERMINAL

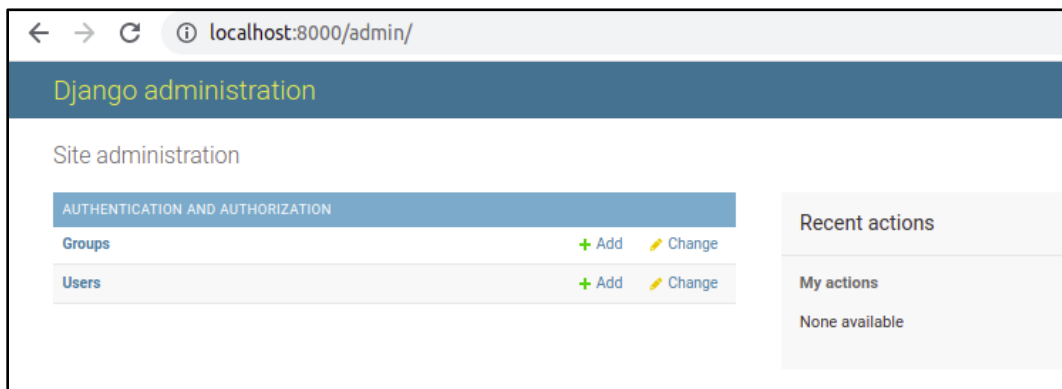
```
1 $ python manage.py createsuperuser
2 Username (leave blank to use 'luispc'): superadmin
3 Email address:
4 Password:
5 Password (again):
6
7 The password is too similar to the username.
8
9 Bypass password validation and create user anyway? [y/N]: y
10
11 Superuser created successfully.
12
13 (projects_django) (base) luispc@FullStack-Dev:site_web_django$
```

Accedemos al panel administrativo con el usuario y la contraseña creada:



The image shows the Django administration login interface. It features a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" with the value "superadmin" and "Password:" with masked characters ".....". A blue "Log in" button is positioned below the password field.

Observamos el sitio de administración de Django:



Actualizando el modelo **Boards**, y agregando al proyecto:

## BOARDS/MODEL.PY

```
1 from django.db import models
2
3 class BoardsModel(models.Model):
4     # Campos del modelo
5     titulo = models.CharField(max_length = 200)
6     descripcion = models.TextField()
7     valor = models.FloatField()
8     creado = models.DateTimeField(auto_now_add=True)
9     modificado = models.DateTimeField(auto_now = True)
10
11     class Meta:
12         permissions = (
13             ("es_miembro_1", "Es miembro con prioridad 1"),
14         )
15
16     def __str__(self):
17         return self.titulo
```

Actualizando el modelo con los nuevos cambios en los atributos:

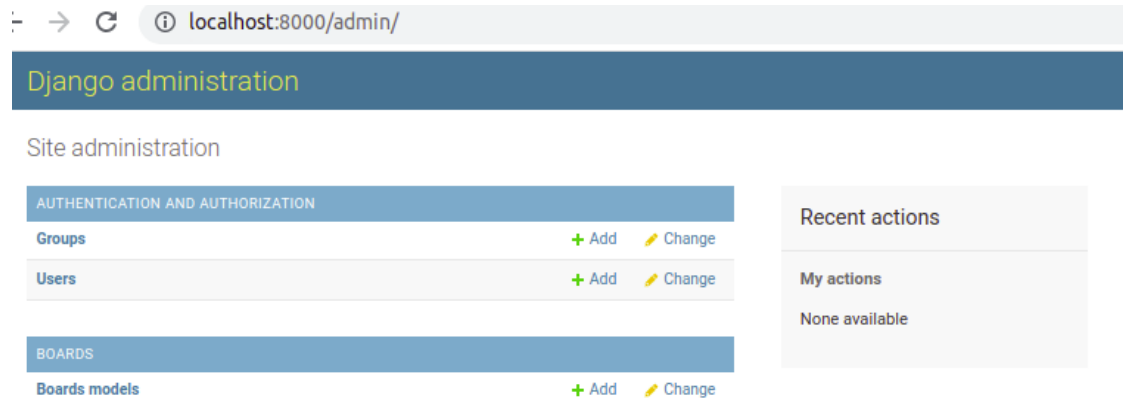
## TERMINAL / CONSOLA

```
1 python manage.py makemigrations
2 python manage.py migrate
```

Agregamos nuestro modelo para que aparezca en el administrador de Django.

## BOARDS/ADMIN.PY

```
1 from django.contrib import admin
2
3 # Register your models here.
4 from .models import BoardsModel
5 admin.site.register(BoardsModel)
```



La aplicación se llama Boards en inglés, y quizá queremos que en el administrador aparezca Tablero en español. Se agrega un campo `verbose_name` en el fichero `boards/apps.py`:

## BOARDS/APPS.PY

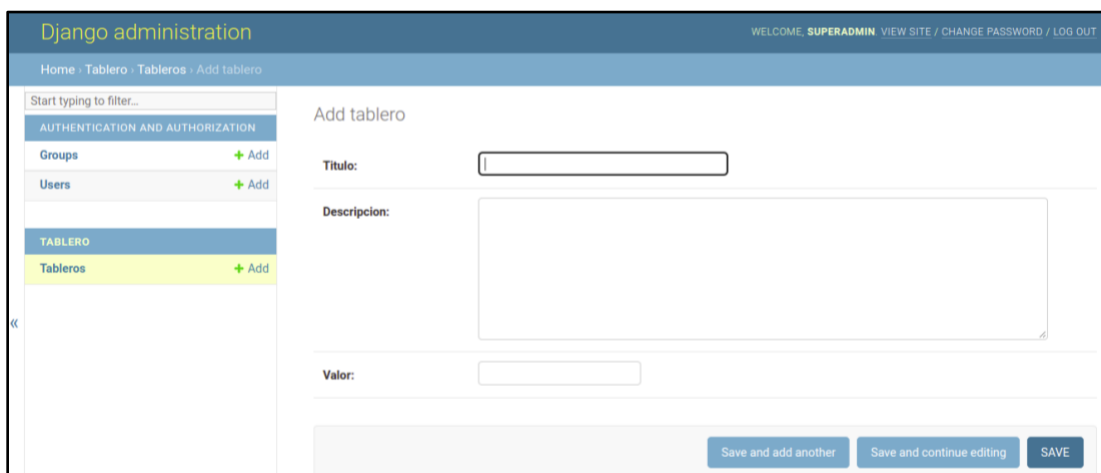
```
1 from django.apps import AppConfig
2
3 class BoardsConfig(AppConfig):
4     default_auto_field = 'django.db.models.BigAutoField'
5     name = 'boards'
6     verbose_name = 'Tablero'
```

Cuando se creó el proyecto, se colocó el nombre de Boards para seguir una lógica en todo el proyecto, pero puede ser cambiado el nombre a mostrar en el panel. Esto se realiza creando una subclase con metainformación: `boards/models.py`

```
1 from django.db import models
2
3 # Create your models here.
4
5 class BoardsModel(models.Model):
6     # Campos del modelo
7     titulo = models.CharField(max_length = 200)
8     descripcion = models.TextField()
9     valor = models.FloatField()
10    creado = models.DateTimeField(auto_now_add=True)
11    modificado = models.DateTimeField(auto_now = True)
12
```

```

13 class Meta:
14     verbose_name = "tablero"
15     verbose_name_plural = "tableros"
16     permissions = (
17         ("es_miembro_1", "Es miembro con prioridad 1"),
18     )
19 def __str__(self):
20     return self.titulo
  
```



The screenshot shows the Django administration interface. The top navigation bar includes the 'Django administration' logo and user information: 'WELCOME, SUPERADMIN VIEW SITE / CHANGE PASSWORD / LOG OUT'. The breadcrumb trail is 'Home > Tablero > Tableros > Add tablero'. On the left sidebar, there are sections for 'AUTHENTICATION AND AUTHORIZATION' (Groups, Users) and 'TABLERO' (Tableros). The main content area is titled 'Add tablero' and contains a form with the following fields: 'Titulo:' (a text input), 'Descripcion:' (a large text area), and 'Valor:' (a text input). At the bottom right of the form are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'.

Ordenación por defecto en un modelo. Por ejemplo, por fecha de creación:

## BOARDS/MODELS.PY

```

1 class Meta:
2     verbose_name = "tablero"
3     verbose_name_plural = "tableros"
4     ordering = ["-creado"]
5     permissions = (
6         ("es_miembro_1", "Es miembro con prioridad 1"),
7     )
  
```

Ordering es un campo del tipo lista, el cual permite ordenar con prioridades entre distintos campos. El guion (-) delante del nombre del campo hace posible ordenar de forma revertida, es decir, indicamos que nos muestre primero los tableros recientemente creados.

### Select tablero to change

Action:

☐ TABLERO

☐ Tablero 2

☐ Tablero 1

2 tableros

El método especial `__str__` se utiliza para devolver la cadena que nosotros queramos. Por ejemplo, por descripción:

#### BOARDS/MODELS.PY

```
1 def __str__(self):  
2     return self.descripcion
```

### Select tablero to change

Action:   0 of 2 selected

☐ TABLERO

☐ Descripción 2

☐ Descripción

2 tableros



## EXERCISE 2: CAMPOS ESPECIALES.

Se puede observar que los campos de creado y modificado tipo fecha automatizados no aparecen en el administrador, pues Django los oculta para evitar su modificación. En este sentido, podemos mostrarlos como campos de tipo "sólo lectura". En el archivo admin.py dentro de nuestra aplicación de Django vamos a heredar de la clase `admin.ModelAdmin`. Esto es lo siguiente:

### BOARDS/ADMIN.PY

```
1 from django.contrib import admin
2 # Register your models here.
3 from .models import BoardsModel
4
5 class BoardsAdmin(admin.ModelAdmin):
6     readonly_fields = ('creado', 'modificado')
7
8 admin.site.register(BoardsModel, BoardsAdmin)
```

La propiedad `list_display`, indicará al admin qué campos queremos listar en el administrador. La propiedad `search_field` especificará sobre qué campos va a efectuarse la búsqueda.

### BOARDS/ADMIN.PY

```
1 from django.contrib import admin
2 # Register your models here.
3 from .models import BoardsModel
4
5 class BoardsAdmin(admin.ModelAdmin):
6     readonly_fields = ('creado', 'modificado')
7     list_display = ('titulo', 'valor')
8     search_fields = ('titulo', 'descripcion')
9     ordering = ('valor',)
10
11 admin.site.register(BoardsModel, BoardsAdmin)
```

Select tablero to change

Q  Search

Action:  Go 0 of 2 selected

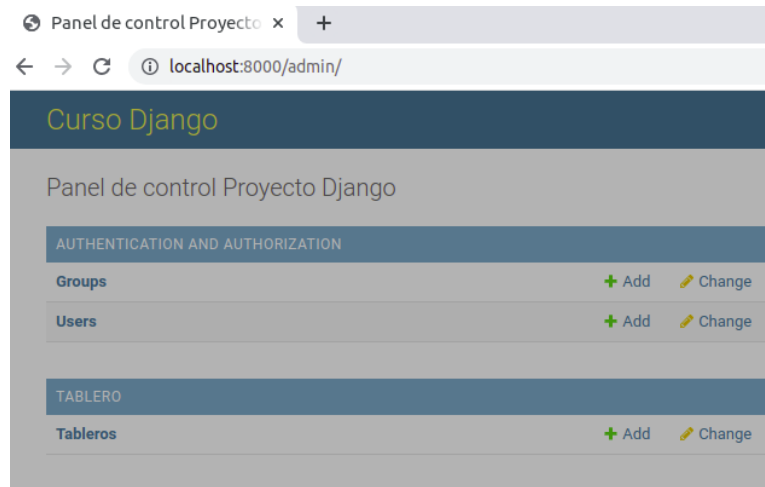
<input type="checkbox"/>	TITULO	VALOR
<input type="checkbox"/>	Tablero 2	4.0
<input type="checkbox"/>	Tablero 1	5.0

2 tableros

Ajustando la presentación de la cabecera, titulo y descripción de administrador:

## BOARDS/ADMIN.PY

```
1 from django.contrib import admin
2 # Register your models here.
3 from .models import BoardsModel
4
5 admin.site.site_header = 'Curso Django'
6 admin.site.index_title = 'Panel de control Proyecto Django'
7 admin.site.site_title = 'Administrador Django'
8
9 class BoardsAdmin(admin.ModelAdmin):
10     readonly_fields = ('creado', 'modificado')
11     list_display = ('titulo', 'valor')
12     search_fields = ('titulo', 'descripcion')
13     ordering = ('valor',)
14
15 admin.site.register(BoardsModel, BoardsAdmin)
```



Aplicando filtros. Por ejemplo, por fecha de creación y valor:

## BOARDS/ADMIN.PY

```
1 from django.contrib import admin
2 # Register your models here.
3 from .models import BoardsModel
4
5 admin.site.site_header = 'Curso Django'
6 admin.site.index_title = 'Panel de control Proyecto Django'
7 admin.site.site_title = 'Administrador Django'
8
9 class BoardsAdmin(admin.ModelAdmin):
10     readonly_fields = ('creado', 'modificado')
11     list_display = ('titulo', 'valor')
12     search_fields = ('titulo', 'descripcion')
13     ordering = ('valor',)
14     list_filter = ('creado', 'valor')
15
16 admin.site.register(BoardsModel, BoardsAdmin)
```

Para crear campos dinámicos personalizados, los cuales no forman parte del modelo, y que se generan de manera dinámica de acuerdo con la información. En este caso, se clasifican los tableros según su valor.

Para crear este campo dinámico, debemos agregar el nombre del campo a `list_display` (para indicarle al admin que lo muestre), y creamos un método con el mismo nombre, el cual recibe el objeto individual y debe retornar lo que queremos que se muestre en pantalla.

## BOARDS/ADMIN.PY

```
1 class BoardsAdmin(admin.ModelAdmin):
2     readonly_fields = ('creado', 'modificado')
3     list_display = ('clasificacion', 'titulo', 'valor')
4     search_fields = ('titulo', 'descripcion')
5     ordering = ('valor',)
6     list_filter = ('creado', 'valor')
7
8     def clasificacion(self, obj):
9         return "Alto" if obj.valor >= 5 else "Bajo"
10
11 admin.site.register(BoardsModel, BoardsAdmin)
```