

Panel de Administrador

Antonio Espín Herranz

Panel de Administrador

- Django proporciona utilidades para gestionar la administración de nuestro sitio de tal forma que nos podamos centrar en la parte pública.
- Simplemente le tenemos que decir a Django que queremos activar esta parte.
- No solo proporciona temas de seguridad pudiendo crear usuarios y grupos, si no que podemos añadir nuestro modelo para poder dar de alta, editar, borrar, buscar, etc. Dentro de los datos que vamos a publicar.

Panel de Administrador

- Para la activación tenemos que ir al fichero de settings.py y dentro de la lista de aplicaciones instaladas (variable **INSTALLED_APPS**) comprobar si se encuentra la aplicación:
 - **django.contrib.admin**
- Además de las **4 dependencias** que necesita:
 - django.contrib.auth
 - django.contrib.contenttypes
 - django.contrib.messages
 - django.contrib.sessions
- Por defecto, cuando se crea el proyecto ya lo activa django.

Panel de Administrador

- Si la activación de estas aplicaciones ya estaba de antes y ya habíamos lanzado el comando:
 - **python manage.py migrate**
 - Las tablas ya estarán creadas, si no, es así habrá que crear las migraciones y luego crearlas.
- El siguiente paso es crear el super usuario:
 - Comando:
 - **python manage.py createsuperuser**
 - Solicita un nombre, email y password.
 - Validación de contraseña.

Panel de Administrador

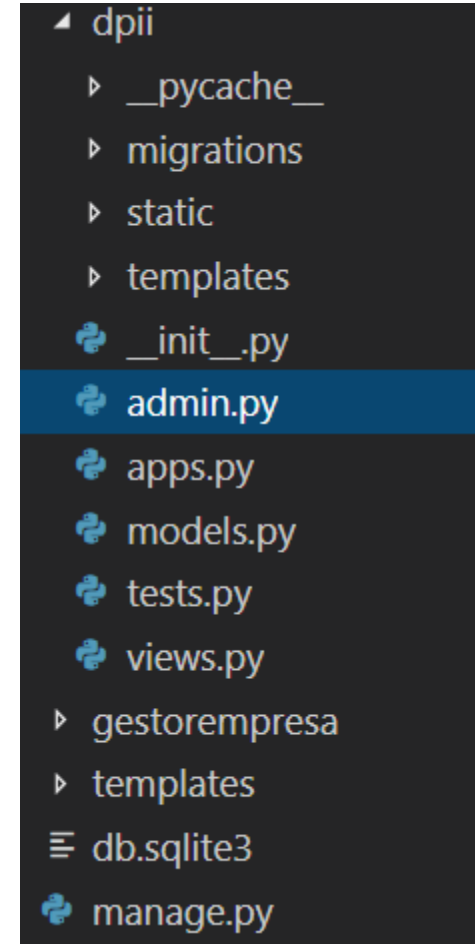
- También comprobar que tenemos activada la URL para poder entrar en el panel de administrador.
- Dentro del fichero: **urls.py**
 - Tendremos la entrada en **urlpatterns**:
 - **path('admin/', admin.site.urls),**
- Para probar si funciona lanzamos el servidor:
 - **python manage.py runserver**
 - En un navegador: <http://127.0.0.1:8000/admin>
 - Y nos pedirá los datos del usuario creado.
- El panel de administración por defecto aparece en inglés se puede cambiar modificando la variable:
 - **LANGUAGE_CODE = 'es'**
 - En el fichero de **settings.py**

Panel de Administración

- Desde nuestro sitio web nos puede interesar tener un enlace para poder acceder al panel de administración.
- Tendríamos que añadir el siguiente enlace:
`<a href="{% url 'admin:index' %}"`

Agregar los modelos

- El panel de administración puede gestionar nuestros modelos.
- Para ello tenemos que ir al **fichero admin.py** y **registrar** nuestros modelos.
- Este fichero se encuentra dentro de la carpeta de la aplicación que hemos creado (dentro del proyecto).



Agregar los modelos

- Tenemos que importarlos.
 - **from nombreApp.models import Modelo1**
 - Se importan directamente los nombres de las clases.
- Y posteriormente se registran:
 - **admin.site.register(Modelo1)**
 - Así con todos los que queremos que formen parte del panel de administración.
 - Construye formularios y **controla si los campos pueden estar o no en blanco** (estas características se indicaban al definir el modelo).
 - Si dejamos un campo obligatorio (se marcan en negrita para indicarlo) los mostrará en color rojo.

Agregar los modelos

- Dentro del panel de administración cuando veamos el listado de un determinado modelo, por ejemplo: las empresas, por defecto mostrará lo que devuelva el método **`__str__()`** pero se puede personalizar para mostrar más columnas de ese modelo.
- Para ello hay que crear una clase que **herede** de **`admin.ModelAdmin`** y dentro de esta se define una propiedad **`list_display`**, que será una **tupla** con los nombres de los atributos que queremos mostrar de este modelo.
- Luego esta clase **también hay que registrarla** en el fichero **`admin.py`**

Ejemplo

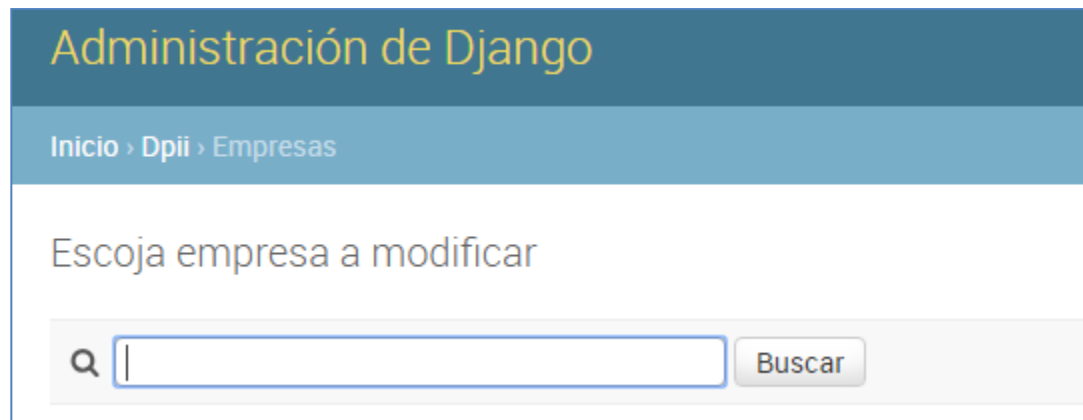
- Suponemos que tenemos dentro de models.py una clase empresa con una serie de atributos ...
- **Dentro de admin.py:**
class **EmpresaAdmin**(admin.ModelAdmin):
 list_display = ('empresa', 'sector', 'cif', 'telefono')
- Cuando se muestre el listado de empresas se verán los 4 campos indicados en la propiedad **list_display**.
- Y también se registra:
admin.site.register(Empresa, **EmpresaAdmin**)

Agregar campos de búsqueda

- Disponemos de las propiedades:
 - **list_filter**: muestra una lista para actuar como filtro.
 - **search_fields**: define campos de búsqueda.
 - Se especifican con una tupla.
- Estas se añaden también en las clases Admin del fichero **admin.py**
- Con la propiedad **search_fields** podemos añadir campos de búsqueda.
- Pueden ser uno o varios. Automáticamente añade una barra de búsqueda a la pantalla.

Agregar campos de búsqueda

- Por ejemplo, disponemos de un modelo que es Empresa que tiene varios campos como empresa (es el nombre de la empresa), cif, sector, etc.
- Si queremos realizar búsquedas por el nombre de la empresa:



The screenshot displays the Django Admin interface. At the top, there is a dark blue header with the text "Administración de Django" in yellow. Below this is a light blue breadcrumb trail showing "Inicio > Dpii > Empresas". The main content area has a heading "Escoja empresa a modificar". Below the heading is a search bar with a magnifying glass icon on the left and a "Buscar" button on the right. The search bar is currently empty.

Agregar campos de búsqueda

- Hace una búsqueda tipo **like**.
- Dentro del fichero admin.py se especifica la clase EmpresaAdmin, luego habrá que registrarla:

```
class EmpresaAdmin(admin.ModelAdmin):  
    # Indicamos las columnas a mostrar en el listado:  
    list_display = ('empresa', 'sector', 'cif', 'telefono')  
    # Indicamos los campos de búsqueda:  
    search_fields = ('empresa',)
```

```
admin.site.register(Empresa, EmpresaAdmin)
```

OJO! Estas propiedades son tuplas ,

Agregar campos de búsqueda

- Cuando el campo de búsqueda está definido como **ForeignKey** dentro del modelo.
- En el caso del modelo Empresa, sector está definido en el modelo así:
 - **sector = models.ForeignKey(Sector, on_delete=models.PROTECT)**
- Cuando queremos hacer búsquedas por este campo, tenemos que indicar:
 - **NombrePropiedad__nombrePropiedad(en la tabla de la clave foránea)**

- Ahora en empresa queremos filtrar por nombre de empresa (campo empresa) y por sector.
 - La búsqueda la hará en las dos columnas.
 - Además, si utilizamos la propiedad **list_filter** a la derecha muestra la lista de valores para poder seleccionarlo directamente o se puede teclear en la barra.

```
class EmpresaAdmin(admin.ModelAdmin):  
    list_display = ('empresa', 'sector', 'cif', 'telefono')  
    # la propiedad sector dentro de Empresa y luego también se llama sector dentro del modelo Sector  
    search_fields = ('empresa', 'sector__sector',)  
    # Si además queremos una lista que haga de filtro:  
    list_filter = ('sector__sector',)
```

– El modelo de Sector estaba definido:

```
class Sector(models.Model):  
    sector = models.CharField(max_length=50)  
  
    class Meta:  
        ordering = ["sector"]  
        verbose_name_plural = "Sectores"  
  
    def __str__(self):  
        return self.sector
```



Si en **list_filter** ponemos varios campos, la posición en la tupla determina la posición en la pantalla.

Campos opcionales

- La interface del panel de administración gestiona si los campos son obligatorios o no.
 - El control se lleva con las características especificadas en el modelo.
- Si el campo es de tipo **texto** para **aceptar blancos** se indica en el modelo:
 - **blank=True**
- Si el campo es **numérico** o de tipo **fecha**:
 - **blank=True** y **null=True**

Personalizar las etiquetas de los campos

- Cuando Django crea de forma automática las etiquetas de los campos, lo que hace es sustituir los guiones del nombre por espacios en blanco y la primera letra la pone en mayúsculas.
- Se puede indicar otra etiqueta predefinida cuando se indica el campo en el modelo.
- Se utiliza el parámetro:
 - **verbose_name='titulo'**
 - O lo indicamos en la primera posición sin el nombre del parámetro.

Fecha Jerárquicas

- Se pueden crear jerarquías con las fechas.
- Se indica con la propiedad: **date_hierarchy**

```
class GastoAdmin(admin.ModelAdmin):
```

```
    list_display = ('fecha','concepto','periodo','baseimponible','porcentajeiva',  
                    'iva','total','fechagrabacion')
```

```
    date_hierarchy = 'fecha'
```



Escoja gasto a modificar

2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018

Personalizar formularios de edición

- Se pueden **definir clases personalizadas** para la interface del **panel de administración**.
- **Establecer un orden en los campos** que vamos a editar en un formulario.
- **O excluir un campo del formulario**, si no se añade a la lista, django lo trata como None y el campo debería aceptar nulos (*verlo en el modelo*).

```
class LibroAdmin(admin.ModelAdmin):  
    list_display = ('titulo', 'editor', 'fecha_publicacion')  
    list_filter = ('fecha_publicacion',)  
    date_hierarchy = 'fecha_publicacion'  
    ordering = ('fecha_publicacion',)  
    fields = ('titulo', 'autores', 'editor', 'fecha_publicacion')
```

Personalizar formularios de edición

- Los **campos** que representan relaciones **de muchos a muchos** también se pueden personalizar.
- Dentro de la clase **Admin** disponemos de la **propiedad**:
filter_horizontal('campo', ...)

```
class LibroAdmin(admin.ModelAdmin):  
    list_display = ('titulo', 'editor', 'fecha_publicacion')  
    list_filter = ('fecha_publicacion',)  
    date_hierarchy = 'fecha_publicacion'  
    ordering = ('fecha_publicacion',)  
    filter_horizontal = ('autores',)
```

Personalizar formularios de edición

- Campo de muchos a muchos:



También ofrece la posibilidad en vertical con la propiedad: **filter_vertical**

Personalizar formularios de edición

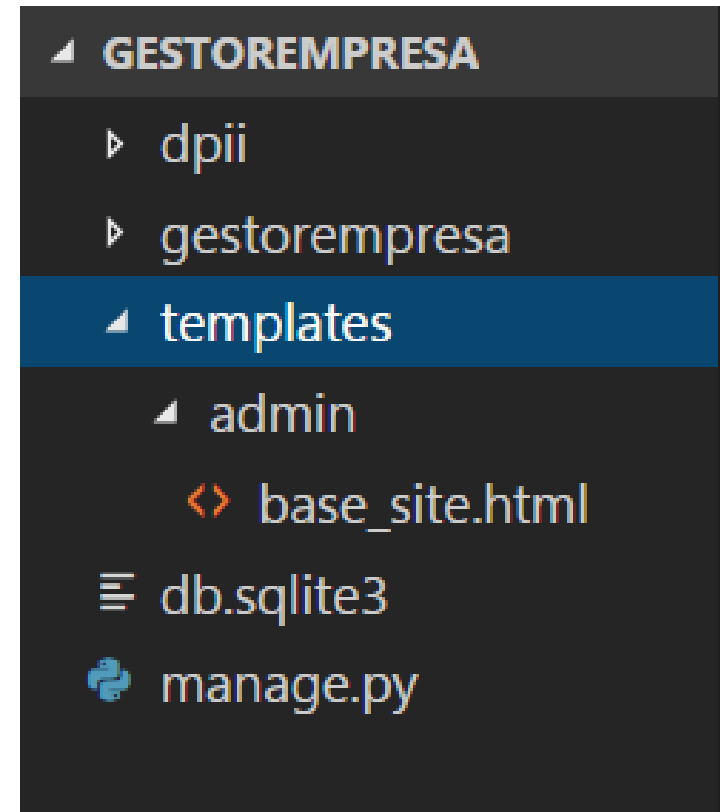
- Las **foreingkey** se suele mostrar en un campo **select** de **HTML** (un desplegable), si la BD tiene muchos registros, puede llevar **tiempo** la **carga**.
- Se pueden sustituir por un campo de texto, se configura con la propiedad:
 - **raw_id_fields = ('editor',)**
 - **Almacenaría un id, con la lupa que añade podemos seleccionar el editor y no es necesario teclear el id.**

Personalizar la interface de administración

- La interface de administración viene con unas plantillas por defecto.
- Estas plantillas se ubican en la carpeta de instalación de Django:
 - *django/contrib/admin/templates*
- Una ruta de instalación completa puede ser:
 - *C:/Program Files (x86)/Python36-32/Lib/site-packages/django/contrib/admin/templates/admin*
- ***Si estamos en un entorno virtual:***
 - *entorno/Lib/sites-packages/django/contrib/admin/templates/admin*
- Si copiamos la plantilla ***base_site.html*** dentro de nuestro **proyecto** en: ***templates/admin/base_site.html*** podemos modificarla para cambiar el título.
 - ***Ojo, esta carpeta se queda a nivel de proyecto NO de aplicación.***

Personalizar la apariencia de la interfaz de administración

- Para realizar cambios en esta interfaz necesitamos copiar la plantilla base que utiliza Django a nuestro proyecto e indicarle en el fichero **settings.py** la ruta donde se encuentra la plantilla.
- Creamos un directorio **templates** a la altura del fichero **manage.py**
- Una buena práctica es organizar las **plantillas en carpetas**.



Personalizar la apariencia de la interfaz de administración

- La plantilla la copiamos de:
 - **django/contrib/admin/templates**
 - Se llama: **base_site.html**
 - Dentro de la plantilla sustituimos:
 - `{{site_header|default:_('Djangoadministración')}}`
 - **Por nuestro texto!**
- En el fichero **settings.py**: modificamos la ruta de **TEMPLATES DIRS**

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, 'templates')],  
        ...  
    }  
]
```

Personalizar la página index del administrador

- Se dispone de la plantilla: **.../admin/index.html** se copia a `templates/admin/index.html` para personalizarla.
- Ojo, siempre copiar el original al proyecto y luego modificar.

Cambiar formato Fechas

- Dentro del fichero **settings.py** podemos cambiar el formato de las fechas.
- Con la propiedad **USE_L10N = True** que es como viene por defecto, las muestra con el nombre del mes: 24 de Septiembre de 2018
- Para poner otro formato, anulamos la propiedad:
 - *from django.conf.locale.es import formats as es_formats*
 - *USE_L10N = False*
 - *es_formats.DATE_FORMAT = "d/m/Y"*