



EBook Gratis

APRENDIZAJE django-admin

Free unaffiliated eBook created from
Stack Overflow contributors.

#django-
admin

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con django-admin.....	2
Observaciones.....	2
Recursos.....	2
Versiones.....	2
Examples.....	3
Configurar Django Admin.....	3
Agregar un modelo a las páginas de administración.....	4
Eliminar un modelo de las páginas de administrador.....	5
Personalizar django User Admin Model.....	5
Capítulo 2: Acciones de administrador.....	6
Observaciones.....	6
Examples.....	6
Acción de descuento del producto.....	6
Los basicos:.....	7
El modelo:.....	7
La meta:.....	7
Configuración (aplicación, modelo y administrador de Django).....	7
Generar algunos productos falsos.....	9
¡Uf! Final las acciones.....	10
Creditos.....	13

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [django-admin](#)

It is an unofficial and free django-admin ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official django-admin.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con django-admin

Observaciones

Django Admin es la interfaz CRUD del marco web de [Django](#) . En su mayoría se genera automáticamente, pero se puede personalizar ampliamente. Sin embargo, debe tener en cuenta que está diseñado solo **para usuarios de confianza** y tiene sus límites. En cualquier caso, **nunca debe dar acceso de administrador a usuarios que no sean de confianza** .

Django Admin proporciona un alto nivel de personalización, pero tenga cuidado de no perder demasiados detalles de personalización. Si lo hace, es probable que sea hora de crear su propia interfaz personalizada sin Django Admin.

Recursos

- [Django Admin Official Introduction](#)
- [Django Admin Official Tutorial](#)
- [Documentación Oficial Django Admin](#)
- [Código fuente de Django Admin](#)

Versiones

Versión	Fecha de lanzamiento
1.10	2106-08-01
1.9	2015-12-01
1.8	2015-04-01
1.7	2014-09-02
1.6	2013-11-06
1.5	2013-02-26
1.4	2012-03-23
1.3	2011-03-23
1.2	2010-05-17
1.1	2009-07-29
1.0	2008-09-03

Examples

Configurar Django Admin

Todo lo que necesita para comenzar con el administrador de Django ya está configurado en el diseño de proyecto predeterminado de Django. Esto incluye:

```
# settings.py

# `django.contrib.admin` and its dependancies.
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    ...,
]

MIDDLEWARE = [
    ...
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    ...
]

TEMPLATES = [
    {
        ...,
        'OPTIONS': {
            'context_processors': [
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                ...
            ],
        },
    },
]
```

Tenga cuidado con `urls.py` que es ligeramente diferente en Django >= 1.9 que en versiones anteriores.

1.9

```
from django.conf.urls import url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
]
```

1.9

```
from django.conf.urls import url, include
from django.contrib import admin
```

```
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
]
```

La versión con `include` todavía funcionará en Django 1.9 pero está obsoleta y se eliminará en el futuro.

Si aún no lo ha hecho, debe aplicar las migraciones base:

```
$ python manage.py migrate
```

Para acceder al administrador, también tienes que crear un superusuario con:

```
$ python manage.py createsuperuser
```

Una vez hecho esto, puede ejecutar su servidor:

```
$ python manage.py runserver
```

Y visite la página de administración en <http://127.0.0.1:8000/admin/>

Agregar un modelo a las páginas de administración

Cuando creaste tus propios Modelos en una aplicación, aún deben *registrarse* para que estén disponibles en las páginas de administración.

Esto se hace en el submódulo de `admin`. Si su aplicación se creó utilizando `manage.py startapp`, ya debería `admin.py` un archivo `admin.py` en su módulo de aplicación. De lo contrario crearlo.

```
#myapp/admin.py
from django.contrib import admin
from myproject.myapp.models import MyModel

admin.site.register(MyModel)
```

Todas las opciones están definidas en la subclase `ModelAdmin`. algunas opciones:

```
class MyCustomAdmin(admin.ModelAdmin):
    list_display = ('name', 'age', 'email') # fields to display in the listing
    empty_value_display = '-empty-' # display value when empty
    list_filter = ('name', 'company') # enable results filtering
    list_per_page = 25 # number of items per page
    ordering = ['-pub_date', 'name'] # Default results ordering

# and register it
admin.site.register(MyModel, MyCustomAdmin)
```

Una forma más concisa de registrar un modelo es usar el decorador `admin.register`:

```
@admin.register(MyModel)
```

```
class MyCustomAdmin(admin.ModelAdmin)
    ...
```

Eliminar un modelo de las páginas de administrador

Django Admin viene con algunos modelos registrados de forma predeterminada. Existen algunas ocasiones en las que es posible que desee eliminar un Modelo de las páginas de administración.

Esto se hace en el submódulo de `admin`. Si su aplicación se creó con `manage.py startapp`, el archivo `admin.py` ya debería estar en su módulo de aplicación. De lo contrario crearlo.

```
#myapp/admin.py
from django.contrib import admin
from django.contrib.auth.models import User

admin.site.unregister(User)
```

Personalizar django User Admin Model

```
from django.contrib.auth.models import User
class UserAdmin(admin.ModelAdmin):
    list_display = ('email', 'first_name', 'last_name')
    list_filter = ('is_staff', 'is_superuser')

admin.site.unregister(User)
admin.site.register(User, UserAdmin)
```

Necesitamos cancelar el registro antes de registrar `UserAdmin` personalizado porque en django `User Model Admin` ya está registrado, por lo tanto, primero tenemos que cancelar el registro del `User Model` en nuestro `admin.py` y luego podemos registrar el `User Model` con un `ModelAdmin` personalizado.

Lea Empezando con django-admin en línea: <https://riptutorial.com/es/django-admin/topic/3499/empezando-con-django-admin>

Capítulo 2: Acciones de administrador

Observaciones

No creo que necesites `get_price (self)` especialmente cuando no estás haciendo ningún cambio en la variable de precio. Quitaría el método `get_price` ya que puede obtener el mismo valor del precio en el modelo del producto. Podría estar equivocado. Simplemente no vea el valor del método `get_price` aquí.

Examples

Acción de descuento del producto

Un día tuve una conversación con un amigo mío que usa el framework PHP [Laravel](#) en su trabajo. Cuando le dije que Django tiene su propio sistema HTML CRUD todo incluido, para interactuar con la base de datos, llamado [administrador de Django](#), ¡sus ojos saltaron! Me dijo: " *Me tomó meses construir una interfaz de administración para mi aplicación web actual y ¿estás diciendo que tienes todo esto sin tener que escribir una sola línea de código?*". Respondí " *¡Yeap!*"

El administrador de Django es una característica poderosa de [Django](#) que proporciona muchas novedades. Una de estas son las [acciones](#).

Pero, ¿qué son las " acciones "?

Supongamos que tiene un modelo y ya ha agregado algunas entradas (quizás cientos, quizás miles) en él. Ahora, desea aplicar una regla-acción a al menos una de ellas, **sin** usar la consola (`python manage.py shell`):

```
# python manage.py shell (interactive console)

from math import ceil

from my_app.models import Product

DISCOUNT = 10 # percentage

for product in Product.objects.filter(is_active=True):
    """ Set discount to ALL products that are flagged as active """
    multiplier = DISCOUNT / 100. # DISCOUNT / 100 in python 3 (without dot)
    old_price = product.price
    new_price = ceil(old_price - (old_price * multiplier)) # seller wins :)
    product.price = new_price
    product.save(update_fields=['price'])
```

¿Te diste cuenta de que aplicamos el descuento a **todos los** productos? ¿Y si quisiéramos aplicar esta lógica a unas específicas? ¿O si quisiéramos ingresar manualmente el valor del descuento y luego aplicar este valor a algunos productos? Todo esto a través del administrador de Django! Estás en el camino correcto. Acciones Django FTW. Veamos un ejemplo completo.

Los basics:

- Python 3.4.3
- Django 1.10
- SQLite (construido en Python, sin necesidad de instalación-configuración adicional)

El modelo:

- Representando un producto para nuestra tienda virtual.
- El precio de un producto sería entero (no decimales)

La meta:

- Para poder, a través de la interfaz de administración de Django, aplicar un descuento fijo a una o más entradas de productos.

Configuración (aplicación, modelo y administrador de Django)

Suponiendo que ya ha [iniciado un proyecto](#), vaya al directorio donde vive `manage.py` y cree una aplicación con el nombre de `stock`, escribiendo:

```
python manage.py createapp stock
```

Django creará automáticamente una estructura de directorios para usted. Ve y edita el archivo `models.py` y agrega:

```
# models.py

from django.db import models

class Product(models.Model):
    name = models.CharField('Product name', max_length=100) # required field
    price = models.PositiveIntegerField('Product price')

    def __str__(self): # __unicode__ in Python 2
        return self.name
```

Nuestro modelo de `Product` ha sido creado, pero nada en la base de datos todavía. Para que las migraciones funcionen, nuestra aplicación debe incluirse en la lista `INSTALLED_APPS`.

Edite su archivo `settings.py` y en la lista `INSTALLED_APPS` agregue:

```
# settings.py
```

```
INSTALLED_APPS = [  
    # ... previous Django apps  
    'stock.apps.StockConfig',  
]
```

Ahora ejecuta:

```
python manage.py makemigrations  
python manage.py migrate
```

Después del comando `migrate`, su base de datos tiene ahora una tabla llamada `product` que contiene tres columnas, `id`, `name` y `price`. ¡Hasta ahora tan bueno!

Si no ha cambiado nada en su archivo `ROOT_URLCONF`, que generalmente se encuentra dentro de la carpeta `<your_project_name>/<your_project_name>/`, entonces la URL que apunta al sitio de administración de Django debe ser:

```
# urls.py  
  
urlpatterns = [  
    # ... other URLs  
    url(r'^admin/', admin.site.urls),  
]
```

Hasta ahora, no hemos visto nada específico sobre las acciones de administración de Django. Da un paso final y agrega estos dentro de tu `stock/admin.py` archivo:

```
# admin.py  
  
from django.contrib import admin  
  
from .models import Product  
  
@admin.register(Product)  
class ProductAdmin(admin.ModelAdmin):  
    pass
```

DE ACUERDO. La configuración está hecha. Solo para asegurarte de que todo funcione, corre:

```
python manage.py runserver
```

y con su navegador favorito visite la página `127.0.0.1:8000/admin/`. Debería ver la página de administración brillante-glamour-excelente Django, el cual le permite a **C**reate- **E**AD- **T**pdate- **D**ELETE su **R** `Product` modelo! Si por casualidad, la página anterior le solicita un nombre de usuario / contraseña y no tiene ninguno, no hay problema. Usted simplemente no ha creado un `User` para iniciar sesión en el administrador. Simplemente ejecute:

```
python manage.py createsuperuser
```

ingrese su nombre, correo electrónico, nombre de usuario y contraseña (dos veces) y listo.

Generar algunos productos falsos.

Hasta ahora hemos creado el modelo, pero aún no hay entradas (no hay productos). Necesitamos algunas entradas para arrojar luz sobre el poder de las acciones de administración de Django.

Vamos a crear 100 productos y trabajar con estos. Pero, en lugar de presionar manualmente el botón AGREGAR e ingresar un `name` y un `price`, escribiremos un guión para hacer el trabajo por nosotros.

Ejecute `python manage.py shell` y escriba lo siguiente:

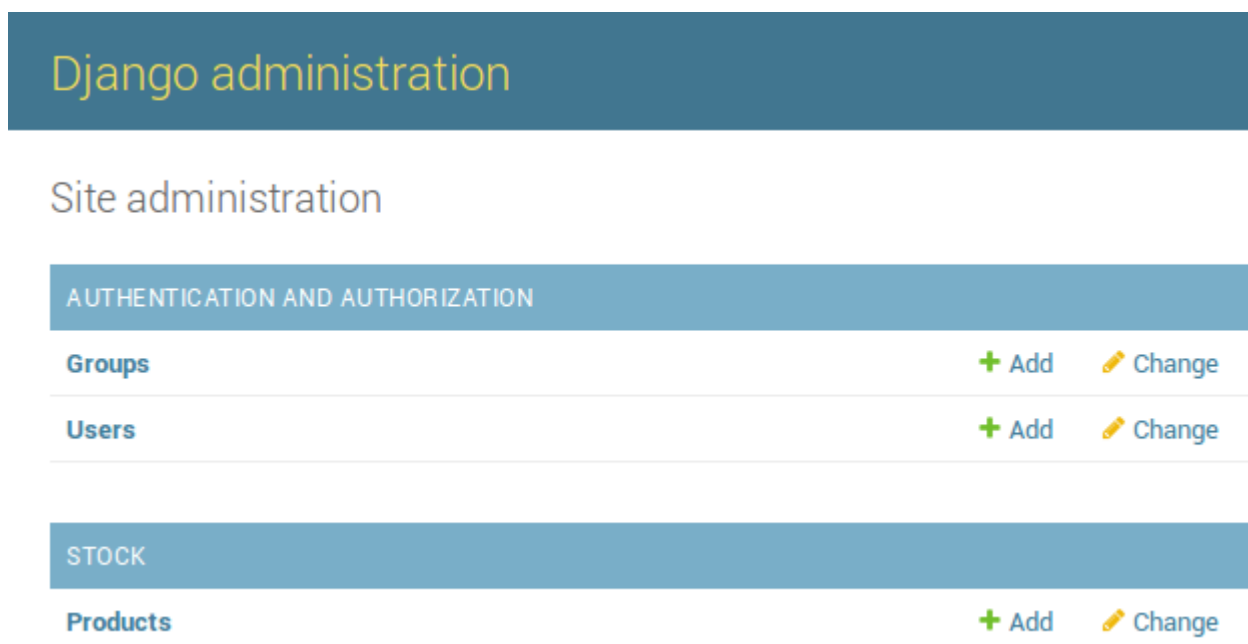
```
# python manage.py shell

from stock.models import Product

for i in range(1, 101):
    p = Product.objects.create(name='Product %s' % i, price=i)
```

Lo anterior `for` creaciones `for` bucle (lo que significa que los datos se guardan en la base de datos) 100 productos (entradas) con nombres `Product 1`, `Product 2`, ... `Product 100` y precios `1`, `2`, ..., `100`.

Para ver estos productos a través de la página de administración de Django, simplemente visite nuevamente `127.0.0.1:8000/admin/` y haga clic en el enlace `Products`:



The screenshot shows the Django administration interface. At the top, there is a dark blue header with the text "Django administration" in yellow. Below this is a section titled "Site administration" in a light blue box. Underneath, there are two main sections: "AUTHENTICATION AND AUTHORIZATION" and "STOCK". The "AUTHENTICATION AND AUTHORIZATION" section contains two items: "Groups" and "Users". Each item has a green plus icon and the text "Add" next to it, and a yellow pencil icon and the text "Change" next to it. The "STOCK" section contains one item: "Products", which also has a green plus icon and the text "Add" next to it, and a yellow pencil icon and the text "Change" next to it.

Disfruta de tus 100 productos autogenerados:

Select product to change

Action:

<input type="checkbox"/>	PRODUCT
<input type="checkbox"/>	Product 100
<input type="checkbox"/>	Product 99
<input type="checkbox"/>	Product 98
<input type="checkbox"/>	Product 97
<input type="checkbox"/>	Product 96
<input type="checkbox"/>	Product 95
<input type="checkbox"/>	Product 94
<input type="checkbox"/>	Product 93
<input type="checkbox"/>	Product 92
<input type="checkbox"/>	Product 91
<input type="checkbox"/>	Product 90

Esto se conoce como la página de `change list` de Django. Ahora, para que esto se vea mucho más bonito, edite su `stock/admin.py` e ingrese:

```
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'price')
```

Ahora, pulse actualizar y debería ver una segunda columna que muestra los precios.

¡Uf! Final las acciones

Para recapitular, tenemos el modelo y tenemos las entradas. A continuación, queremos crear una acción que, una vez seleccionada, hará un descuento del 30% a los productos seleccionados.

¿Notó que ya existe un cuadro de selección en la parte superior de la página de la `change list`, con la etiqueta `Action`? Django agrega automáticamente una acción predeterminada en cada entrada que realiza la acción `Delete`.

Las acciones de administración de Django se escriben como funciones simples. `stock/admin.py`. Edite el `stock/admin.py` y agregue lo siguiente:

```
# admin.py

@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'price')
    actions = ['discount_30']

    def discount_30(self, request, queryset):
        from math import ceil
        discount = 30 # percentage

        for product in queryset:
            """ Set a discount of 30% to selected products """
            multiplier = discount / 100. # discount / 100 in python 3
            old_price = product.price
            new_price = ceil(old_price - (old_price * multiplier))
            product.price = new_price
            product.save(update_fields=['price'])
        discount_30.short_description = 'Set 30%% discount'
```

Algunas cosas a tener en cuenta aquí:

- La clase `ProductAdmin` tiene un atributo adicional (`actions`) que es una lista de cadenas (cada cadena es el nombre de la función que representa la acción).
- La función de acción que es un método de la clase `ProductAdmin`. Toma como argumentos la instancia de `ModelAdmin` (`self` ya que este es un método), el objeto de `HTTP request` y el conjunto de `queryset` (una lista de los objetos-entradas-productos seleccionados).
- La última línea es un atributo de función (`short_description`) que establece el nombre mostrado dentro del cuadro de selección de acciones (hay un `doble%` para escapar del `único%`).

Dentro de la función-acción, iteramos en cada producto (que fue seleccionado) y establecemos su valor disminuido en un 30%. Luego llamamos al método `save()` con el argumento `update_fields` para forzar un `UPDATE` en los campos que se incluyen en la lista de `update_fields` (en lugar de un `UPDATE` en todos los campos del modelo) en la base de datos, por razones de rendimiento (no una ganancia de rendimiento en este ejemplo, con solo 2 columnas, pero se obtiene el punto).

Ahora, pulse actualizar en la página de la `change list` y debería ver su acción debajo de la `delete`. Continúe y seleccione algunos productos (con la casilla de verificación a la izquierda de cada uno o todos los productos con la casilla de verificación situada en la parte superior izquierda), seleccione la acción `Set 30% discount` y haga clic en el botón `Go`. **¡Eso es!**

Select product to change

Action: 0 of 100 selected

<input type="checkbox"/>	PR	Delete selected products
<input type="checkbox"/>	Product 100	Set 30% discount
<input type="checkbox"/>	Product 99	
<input type="checkbox"/>	Product 98	
<input type="checkbox"/>	Product 97	
<input type="checkbox"/>	Product 96	
<input type="checkbox"/>	Product 95	

Por supuesto, esto no es muy útil en la mayoría de las situaciones, ya que esta acción no le permite ingresar una cantidad diferente de descuento. Debe editar el archivo `admin.py` cada vez que desee aplicar un descuento diferente. En el próximo ejemplo veremos cómo hacerlo.

Lea Acciones de administrador en línea: <https://riptutorial.com/es/django-admin/topic/7747/acciones-de-administrador>

Creditos

S. No	Capítulos	Contributors
1	Empezando con django-admin	Antoine Pinsard , Community , Iker , NeErAj KuMaR , Roald Nefs , Udi
2	Acciones de administrador	ira , nik_m , Stryker