EBook Gratuito

APPRENDIMENTO
django-admin

Free unaffiliated eBook created from
Stack Overflow contributors.

#django-admin

Sommario

Di

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.

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

Capitolo 1: Inizia con django-admin

Osservazioni

Django Admin è l'interfaccia CRUD del framework web Django . Generalmente viene generato automaticamente ma può essere ampiamente personalizzato. Tuttavia, è necessario tenere presente che è progettato solo **per utenti fidati** e ha i suoi limiti. In ogni caso, non devi **mai concedere l'accesso come amministratore agli utenti non fidati** .

Django Admin offre un alto livello di personalizzazione, ma fai attenzione a non cadere troppi dettagli di personalizzazione. Se lo fai, probabilmente è il momento di creare la tua interfaccia personalizzata senza Django Admin.

risorse

- Introduzione ufficiale di Django Admin
- Esercitazione ufficiale di Django Admin
- Documentazione ufficiale di Django Admin
- Django Admin Source Code

Versioni

Versione	Data di rilascio
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

Setup Django Admin

Tutto ciò di cui hai bisogno per iniziare con l'admin di Django è già impostato nel layout di progetto predefinito di Django. Ciò comprende:

```
# 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',
                ...
            ],
        },
    },
]
```

`urls.py` attenzione a `urls.py` che è leggermente diverso in Django> = 1.9 rispetto alle versioni precedenti.

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 versione con `include` funzionerà ancora in Django 1.9 ma è deprecata e verrà rimossa in futuro.

Se non è già stato fatto, è necessario applicare le migrazioni di base:

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

Per accedere all'amministratore, devi anche creare un superutente con:

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

Una volta fatto, puoi eseguire il tuo server:

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

E visita la pagina di amministrazione all'indirizzo http://127.0.0.1:8000/admin/

Aggiungi un modello alle pagine di amministrazione

Quando hai creato i tuoi modelli in un'app, devono comunque essere *registrati* per essere disponibili nelle pagine di amministrazione.

Questo è fatto nel sottomodulo `admin` . Se la tua app è stata creata utilizzando `manage.py startapp` , un file `admin.py` dovrebbe già essere `admin.py` nel modulo dell'app. Altrimenti crearlo.

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

admin.site.register(MyModel)
```

Tutte le opzioni sono definite nella sottoclasse ModelAdmin. alcune opzioni:

```
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)
```

Un modo più sintetico per registrare un modello è utilizzare il decoratore `admin.register` :

```
@admin.register(MyModel)
class MyCustomAdmin(admin.ModelAdmin)
    ...
```

Rimuovere un modello dalle pagine di amministrazione

Django Admin viene fornito con alcuni modelli registrati per impostazione predefinita. Vi sono alcune occasioni in cui è possibile rimuovere un modello dalle pagine di amministrazione.

Questo è fatto nel sottomodulo `admin` . Se la tua app è stata creata utilizzando `manage.py startapp` , il file `admin.py` dovrebbe già essere `admin.py` nel modulo dell'app. Altrimenti crearlo.

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

admin.site.unregister(User)
```

Personalizza il modello di amministratore utente django

```
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)
```

Abbiamo bisogno di annullare la registrazione prima di registrare UserAdmin personalizzato perché in Admin modello utente di django già registrato, quindi dobbiamo prima annullare la registrazione del modello utente nel nostro admin.py, quindi possiamo registrare il modello utente con ModelAdmin personalizzato

Leggi Inizia con django-admin online: https://riptutorial.com/it/django-admin/topic/3499/inizia-con-django-admin

Capitolo 2: Azioni di amministrazione

Osservazioni

Non penso che sia necessario get_price (self) specialmente quando non si apportano modifiche alla variabile di prezzo. Vorrei rimuovere il metodo get_price dal momento che è possibile ottenere lo stesso valore dal prezzo sotto il modello del prodotto. Potrei sbagliarmi. Non vedere il valore del metodo get_price qui.

Examples

Azione di sconto sul prodotto

Un giorno ho avuto una conversazione con un mio amico che usa il framework PHP Laravel nel suo lavoro. Quando gli ho detto che Django ha il suo sistema CRUD HTML tutto incluso, per interagire con il database, chiamato Django admin , i suoi occhi sono saltati fuori! Mi ha detto: " *Mi ci sono voluti mesi per creare un'interfaccia di amministrazione per la mia attuale app web e stai dicendo che hai tutto questo senza dover scrivere una singola riga di codice?* ". Ho risposto " *Yeap!* "

L'admin di Django è una potente funzionalità di Django che fornisce molte chicche. Una di queste sono azioni .

Ma quali " *azioni* " sono?

Supponiamo di avere un modello e di aver già aggiunto alcune voci (forse centinaia, forse migliaia). Ora, si desidera applicare un'azione della regola ad almeno uno di essi, **senza** utilizzare la console (`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'])
```

Hai notato che abbiamo applicato lo sconto a **tutti i** prodotti. E se volessimo applicare questa logica a quelli specifici? O se volessimo inserire manualmente il valore dello sconto e applicare questo valore ad alcuni prodotti? Tutto questo attraverso l'admin di Django! Sei sulla strada giusta.

Azioni Django FTW. Vediamo un esempio completo.

Le basi:

- Python 3.4.3
- Django 1.10
- SQLite (costruito in Python, nessuna necessità di installazione-installazione extra)

Il modello:

- Rappresentare un prodotto per il nostro e-shop
- Il prezzo di un prodotto sarebbe costituito da numeri interi (non decimali)

L'obiettivo. il gol:

- Per essere in grado, tramite l'interfaccia di Django Admin, di applicare uno sconto fisso a una o più voci di prodotto.

Setup (app, model e admin di Django)

Supponendo di aver già avviato un progetto , vai nella directory in cui `manage.py` vive e crea un'app con il nome `stock` , digitando:

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

Django creerà automaticamente una struttura di directory per te. Vai e modifica il file `models.py` e aggiungi:

```
# 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
```

Il nostro modello di `Product` è stato creato, ma non ancora nel database. Affinché le migrazioni funzionino, la nostra app deve essere inclusa nell'elenco `INSTALLED_APPS` .

Modifica il tuo file `settings.py` e sotto l'elenco `INSTALLED_APPS` aggiungi:

```
# settings.py
```

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

Ora esegui:

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

Dopo il comando `migrate` , il tuo database ha ora una tabella di nome `product` che contiene tre colonne, `id` , `name` e `price` . Fin qui tutto bene!

Se non hai modificato nulla nel tuo file ROOT_URLCONF , che di solito risiede nella cartella `<your_project_name>/<your_project_name>/` , allora l'URL che punta al sito admin di Django dovrebbe essere:

```
# urls.py

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

Fino ad ora, non abbiamo guardato nulla di specifico sulle azioni amministrative di Django. Fai un ultimo passaggio e aggiungili nel tuo file `stock/admin.py` :

```
# admin.py

from django.contrib import admin

from .models import Product


@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    pass
```

OK. L'installazione è terminata. Per assicurarti che tutto funzioni, esegui:

```
python manage.py runserver
```

e con il tuo browser preferito visita la pagina `127.0.0.1:8000/admin/` . Si dovrebbe vedere la-glamour-formidabile lucido pagina di amministrazione Django, che permette di **C** reate- **R** ead- **U** pdate- **D** elete vostro `Product` del modello! Se per caso, la pagina sopra ti chiede un nome utente / password e non ne hai, nessun problema. Non hai appena creato un `User` per accedere all'amministratore. Basta eseguire:

```
python manage.py createsuperuser
```

inserisci il tuo nome, email, nome utente e password (due volte) e il gioco è fatto.

Genera alcuni prodotti contraffatti

Finora abbiamo creato il modello ma non ancora voci (nessun prodotto). Abbiamo bisogno di alcune voci per far luce sulla potenza delle azioni amministrative di Django.

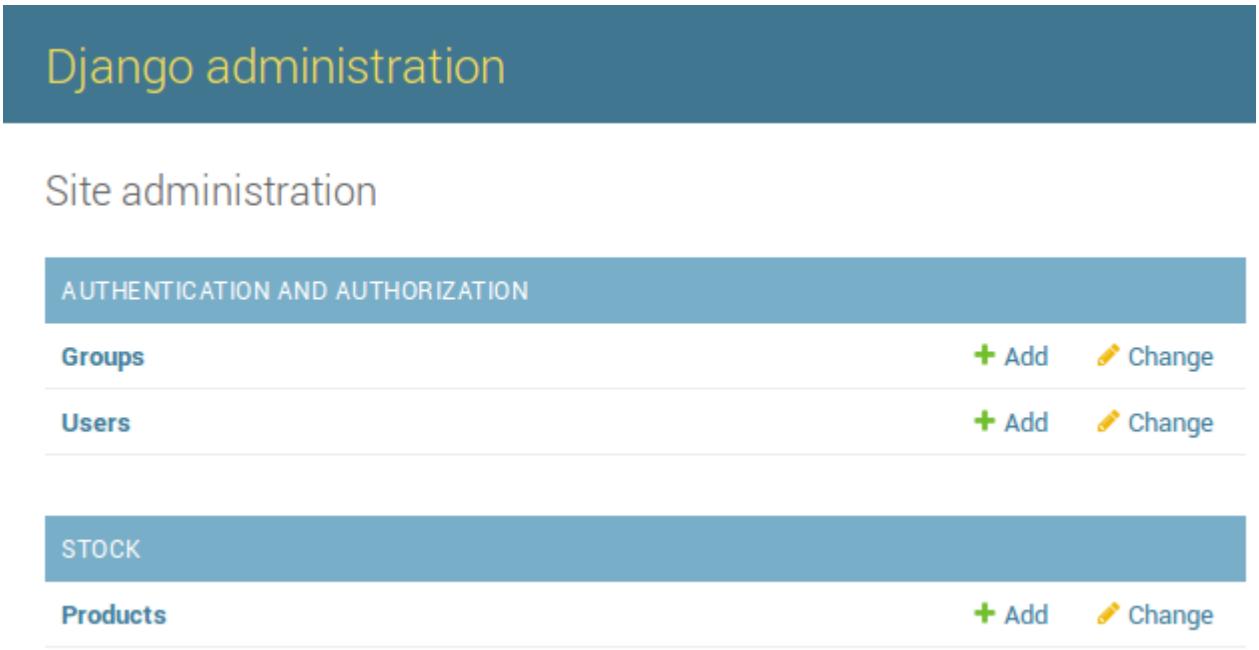
Creeremo 100 prodotti e lavoreremo con questi. Ma, invece di premere manualmente il pulsante ADD e inserire un `name` e un `price` , scriveremo una sceneggiatura per fare il lavoro per noi.

Esegui `python manage.py shell` e inserisci quanto segue:

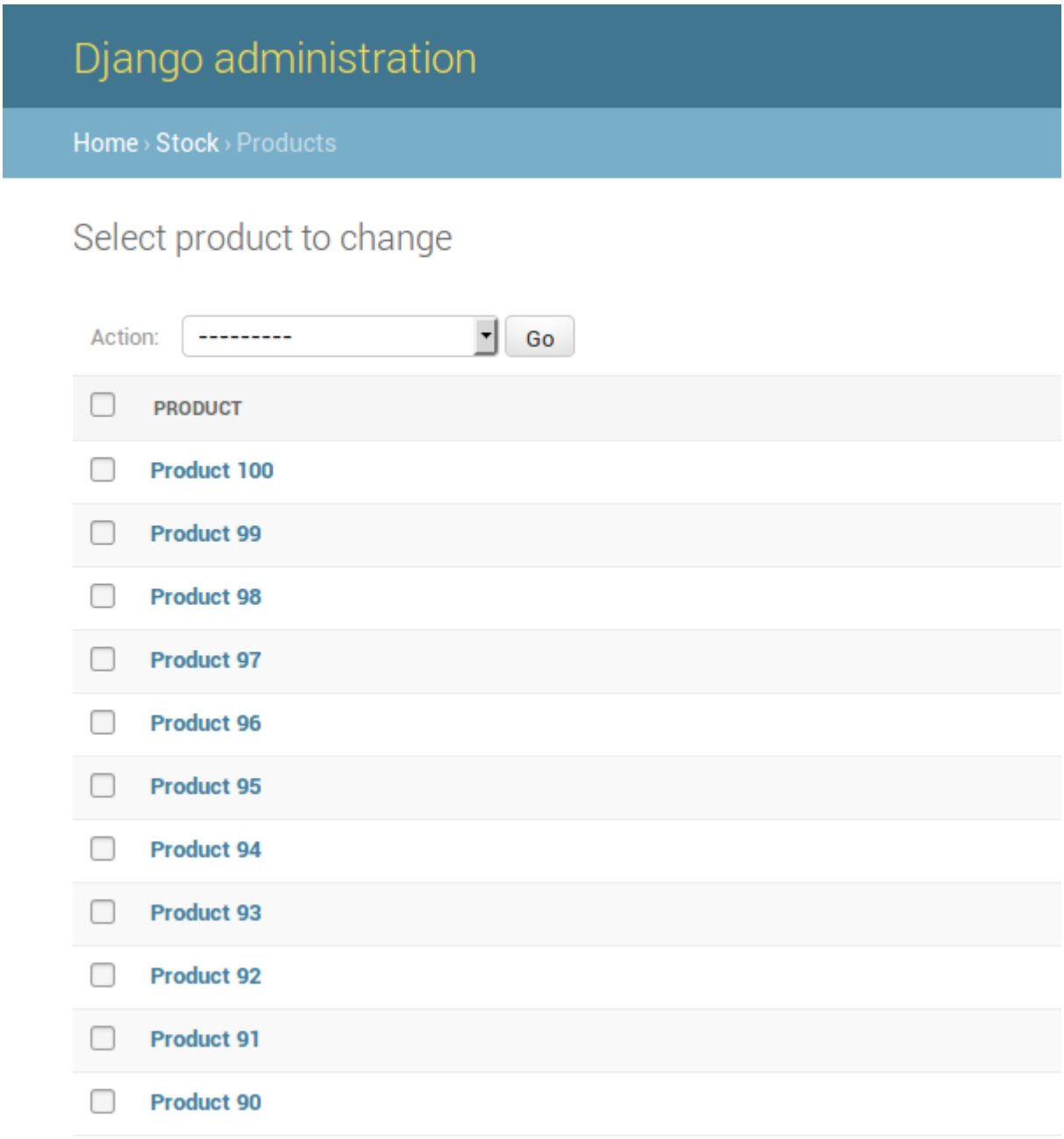
```
# python manage.py shell

from stock.models import Product


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

Quanto sopra `for` ciclo crea (il che significa che i dati vengono salvati nel database) 100 prodotti (voci) con i nomi di `Product 1` , `Product 2` , ... `Product 100` ed i prezzi `1` , `2` , ..., `100` .

Per visualizzare questi prodotti attraverso la pagina di amministrazione di Django, visita di nuovo `127.0.0.1:8000/admin/` e fai clic sul collegamento `Products` :



Goditi i tuoi 100 prodotti generati automaticamente:



Questa è conosciuta come la pagina di `change list` di Django. Ora, per fare in modo che questo sia molto più `stock/admin.py` , modifica il file `stock/admin.py` e inserisci:

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

Ora, premi Aggiorna e dovresti vedere una seconda colonna che mostra i prezzi.

Accidenti! Finale le azioni

Per ricapitolare, abbiamo il modello e abbiamo le voci. Successivamente, vogliamo creare un'azione che, una volta selezionata, farà uno sconto del 30% sui prodotti selezionati.

Hai notato che esiste già una casella di selezione nella parte superiore della pagina `change list` , con l'etichetta `Action` ? Django, aggiunge automaticamente un'azione predefinita su ogni voce che esegue l'azione **D** elete.

Le azioni di amministrazione di Django sono scritte come semplici funzioni. `stock/admin.py` . Modifica il file `stock/admin.py` e aggiungi quanto segue:

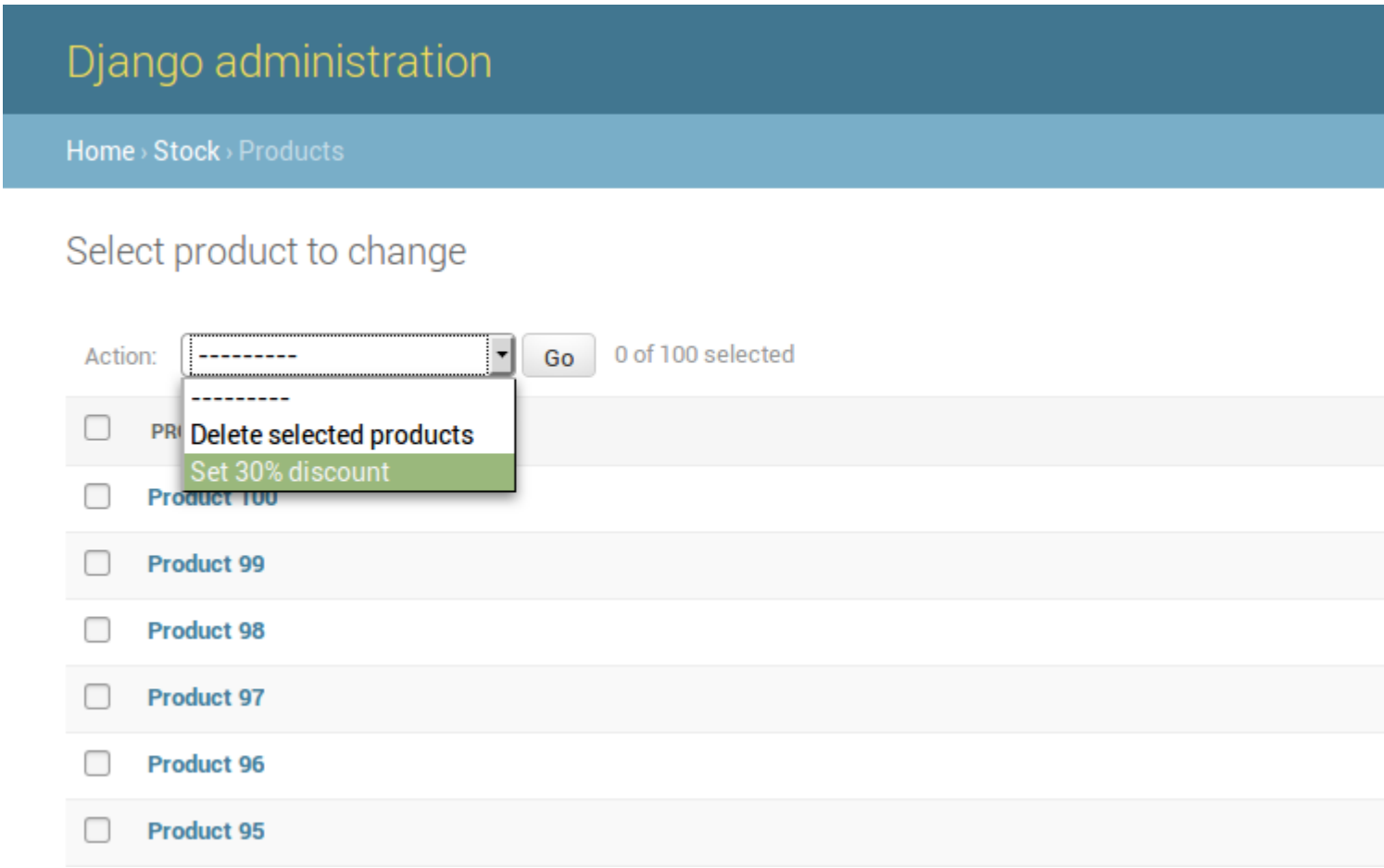
```
# admin.py

@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'
```

Alcune cose da notare qui:

- La classe `ProductAdmin` ha un attributo extra (`actions`) che è un elenco di stringhe (ogni stringa è il nome della funzione che rappresenta l'azione).
- La funzione-azione che è un metodo della classe `ProductAdmin` . Prende come argomenti l'istanza `ModelAdmin` (`self` poiché questo è un metodo), l'oggetto `HTTP request` e il `queryset` (un elenco degli oggetti-voci-prodotti selezionati).
- L'ultima riga è un attributo di funzione (`short_description`) che imposta il nome visualizzato all'interno della casella di selezione delle azioni (c'è un doppio% per sfuggire al singolo%).

All'interno della funzione-azione, iteriamo su ciascun prodotto (che è stato selezionato) e impostiamo il suo valore diminuito del 30%. Quindi chiamiamo il metodo `save()` con l'argomento `update_fields` per forzare un UPDATE sui campi che sono inclusi nell'elenco `update_fields` (invece di un UPDATE su tutti i campi del modello) nel database, per motivi di prestazioni (non un guadagno in termini di prestazioni in questo esempio, con solo 2 colonne, ma ottieni il punto).

Ora, premi Aggiorna nella pagina `change list` e dovresti vedere la tua azione sotto quella di `delete` . Vai avanti e seleziona alcuni prodotti (utilizzando la casella di controllo a sinistra di tutti o tutti i prodotti utilizzando la casella di controllo in alto a sinistra), seleziona l'opzione `Set 30% discount` e fai clic sul pulsante `Go` . **Questo è tutto!**



Naturalmente, questo non è molto utile nella maggior parte delle situazioni, dal momento che questa azione non ti consente di inserire un diverso importo di sconto. È necessario modificare il file `admin.py` ogni volta che si desidera applicare uno sconto diverso. Nel prossimo esempio vedremo come farlo.

Leggi Azioni di amministrazione online: https://riptutorial.com/it/django-admin/topic/7747/azioni-di-amministrazione

Titoli di coda

S. No	Capitoli	Contributors
1	Inizia con django-admin	Antoine Pinsard, Community, lxer, NeErAj KuMaR, Roald Nefs, Udi
2	Azioni di amministrazione	ira, nik_m, Stryker