eBook Gratuit

APPRENEZ

django-admin

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#django-admin

Table des matières

À propos

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

Chapitre 1: Commencer avec django-admin

Remarques

Django Admin est l'interface CRUD du framework web Django . Il est principalement généré automatiquement, mais peut être largement personnalisé. Cependant, vous devez garder à l'esprit qu'il est conçu **pour les utilisateurs de confiance uniquement** et qu'il a ses limites. Dans tous les cas, vous **ne** devez **jamais donner à l'administrateur un accès à des utilisateurs non approuvés** .

Django Admin fournit un haut niveau de personnalisation, mais veillez à ne pas perdre trop de détails de personnalisation. Si vous le faites, il est probablement temps de créer votre propre interface personnalisée sans Django Admin.

Ressources

- Django Admin Présentation officielle
- Didacticiel officiel de Django Admin
- Documentation officielle de Django
- Code source de l'administrateur Django

Versions

Version	Date de sortie
1.10	2106-08-01
1,9	2015-12-01
1.8	2015-04-02
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

Configuration de l'administrateur Django

Tout ce dont vous avez besoin pour démarrer avec Django admin est déjà configuré dans la structure de projet par défaut de Django. Ceci comprend:

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

Attention à `urls.py` qui est légèrement différent dans Django> = 1.9 que dans les anciennes versions.

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 version avec `include` fonctionnera toujours dans Django 1.9 mais elle est obsolète et sera supprimée dans le futur.

Si ce n'est pas déjà fait, vous devez appliquer les migrations de base:

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

Pour accéder à l'administrateur, vous devez également créer un superutilisateur avec:

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

Une fois cela fait, vous pouvez exécuter votre serveur:

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

Et visitez la page d'administration à http://127.0.0.1:8000/admin/

Ajouter un modèle aux pages d'administration

Lorsque vous créez vos propres modèles dans une application, ils doivent toujours être *enregistrés* pour être disponibles dans les pages d'administration.

Ceci est fait dans le sous-module `admin` . Si votre application a été créée à l'aide de `manage.py startapp` , un fichier `admin.py` devrait déjà `admin.py` dans votre module d'application. Sinon, créez-le.

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

admin.site.register(MyModel)
```

Toutes les options sont définies dans la sous-classe ModelAdmin. quelques options:

```
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 moyen plus concis pour enregistrer un modèle consiste à utiliser le décorateur `admin.register` :

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

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

Supprimer un modèle des pages d'administration

Django Admin est livré avec certains modèles par défaut. Il peut arriver que vous souhaitiez supprimer un modèle des pages d'administration.

Ceci est fait dans le sous-module `admin` . Si votre application a été créée avec `manage.py startapp` , le fichier `admin.py` doit déjà se `admin.py` dans le module de votre application. Sinon, créez-le.

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

admin.site.unregister(User)
```

Personnaliser le modèle d'administration utilisateur 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)
```

Nous devons nous désinscrire avant d'enregistrer UserAdmin personnalisé car dans django User Model Admin déjà enregistré, nous devons donc d'abord désenregistrer le modèle utilisateur dans notre admin.py puis nous pouvons enregistrer le modèle utilisateur avec ModelAdmin personnalisé

Lire Commencer avec django-admin en ligne: https://riptutorial.com/fr/django-admin/topic/3499/commencer-avec-django-admin

Chapitre 2: Actions d'administration

Remarques

Je ne pense pas que vous ayez besoin de get_price (self) spécialement quand vous ne modifiez pas la variable de prix. Je supprimerais la méthode get_price car vous pouvez obtenir la même valeur à partir du prix dans le modèle de produit. Je peux me tromper. Il suffit de ne pas voir la valeur de la méthode get_price ici.

Examples

Action de remise de produit

Un jour, j'ai eu une conversation avec un ami qui utilise le framework PHP Laravel dans son travail. Quand je lui ai dit que Django avait son propre système HTML CRUD tout inclus, pour interagir avec la base de données, appelé Django admin , ses yeux se sont envolés! Il m'a dit: " *Il m'a fallu des mois pour créer une interface d'administration pour mon application Web actuelle et vous dites que vous avez tout cela sans avoir à écrire une seule ligne de code?* ". J'ai répondu " *Yeap!* "

Django admin est une fonctionnalité puissante de Django qui fournit de nombreux avantages. L'une d'entre elles est des actions .

Mais quelles sont les " *actions* "?

Supposons que vous ayez un modèle et que vous y ayez déjà ajouté des entrées (peut-être des centaines, voire des milliers). Maintenant, vous voulez appliquer une action de règle à au moins l'un d'entre eux, **sans** utiliser 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'])
```

Avez-vous remarqué que nous avons appliqué la remise à **tous les** produits. Et si nous voulions appliquer cette logique à des logiques spécifiques? Ou si nous voulions saisir manuellement la valeur de la remise, puis appliquer cette valeur à certains produits? Tout cela via l'administrateur

Django! Tu es sur la bonne piste. Actions Django FTW. Regardons un exemple complet.

Les bases:

- Python 3.4.3
- Django 1.10
- SQLite (construit en Python, pas besoin d'une installation supplémentaire)

Le modèle:

- Représenter un produit pour notre e-shop
- Le prix d'un produit serait des entiers (non décimaux)

Le but:

- Pour pouvoir, via l'interface d'administration de Django, appliquer une remise fixe à une ou plusieurs entrées de produit.

Configuration (application, modèle et administrateur Django)

En supposant que vous ayez déjà démarré un projet , allez dans le répertoire où `manage.py` vit et créez une application avec le nom du `stock` , en tapant:

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

Django créera automatiquement une structure de répertoires pour vous. Allez modifier le fichier `models.py` et ajoutez:

```
# 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
```

Notre modèle de `Product` a été créé, mais rien dans la base de données pour le moment. Pour que les migrations fonctionnent, notre application doit être incluse dans la liste `INSTALLED_APPS` .

Modifiez votre fichier `settings.py` et, sous la liste `INSTALLED_APPS` ajoutez:

```
# settings.py

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

Maintenant, lancez:

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

Après la commande `migrate` , votre base de données a maintenant une table nommée `product` qui contient trois colonnes, `id` , `name` et `price` . Jusqu'ici tout va bien!

Si vous n'avez rien changé dans votre fichier ROOT_URLCONF , qui réside généralement dans le dossier `<your_project_name>/<your_project_name>/` , l'URL pointant vers le site d'administration Django doit être:

```
# urls.py

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

Jusqu'à présent, les actions d'administration de Django n'avaient rien de spécifique. Faites une dernière étape et ajoutez-les dans votre fichier `stock/admin.py` :

```
# admin.py

from django.contrib import admin

from .models import Product


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

D'ACCORD. La configuration est terminée. Juste pour s'assurer que tout fonctionne, lancez:

```
python manage.py runserver
```

et avec votre navigateur préféré, visitez la page `127.0.0.1:8000/admin/` . Vous devriez voir la page d'administration Django-fantastique brillant glamour qui vous permet de **C** reate- **R** ead- **U** pdate- **D** élete votre `Product` modèle! Si par hasard, la page ci-dessus vous demande un nom d'utilisateur / mot de passe et que vous n'en avez pas, pas de problème. Vous n'avez simplement pas créé d' `User` pour vous connecter à l'administrateur. Exécutez simplement:

```
python manage.py createsuperuser
```

entrez votre nom, email, nom d'utilisateur et mot de passe (deux fois) et vous avez terminé.

Générer des produits contrefaits

Jusqu'à présent, nous avons créé le modèle mais pas encore d'entrées (pas de produits). Nous avons besoin de quelques entrées pour mettre en lumière la puissance des actions d'administration de Django.

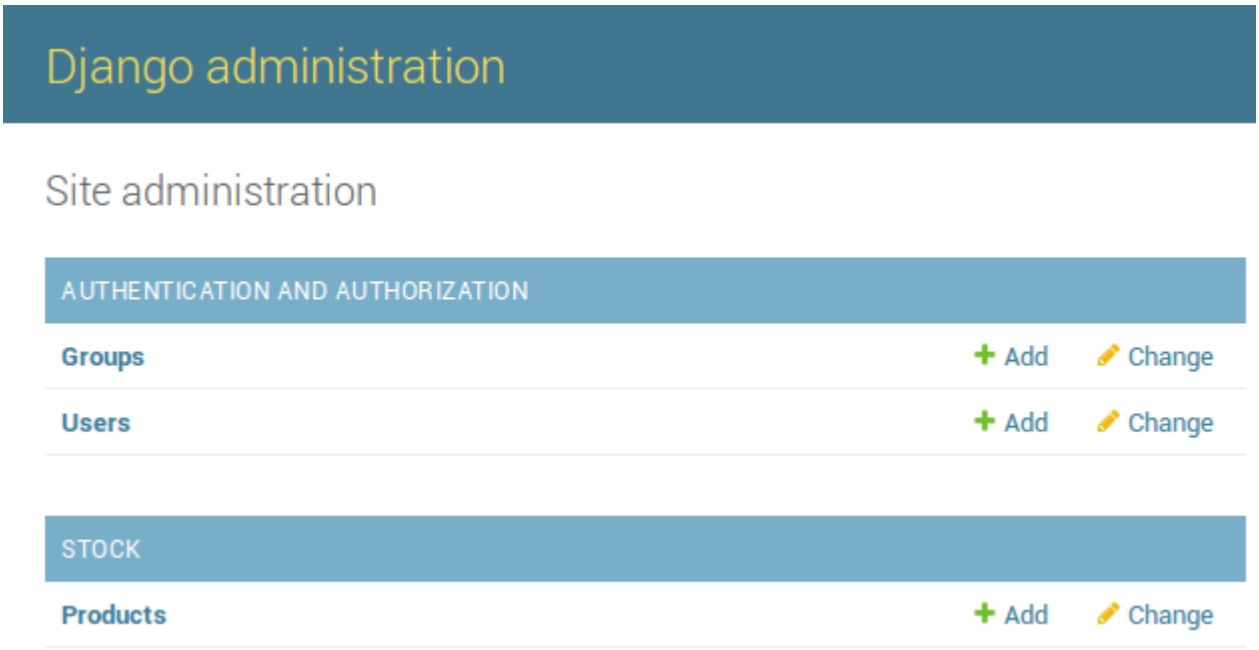
Nous allons créer 100 produits et travailler avec ceux-ci. Mais, au lieu d'appuyer manuellement sur le bouton AJOUTER et d'entrer un `name` et un `price` , nous écrirons un script pour faire le travail pour nous.

Exécutez le `python manage.py shell` et entrez les éléments suivants:

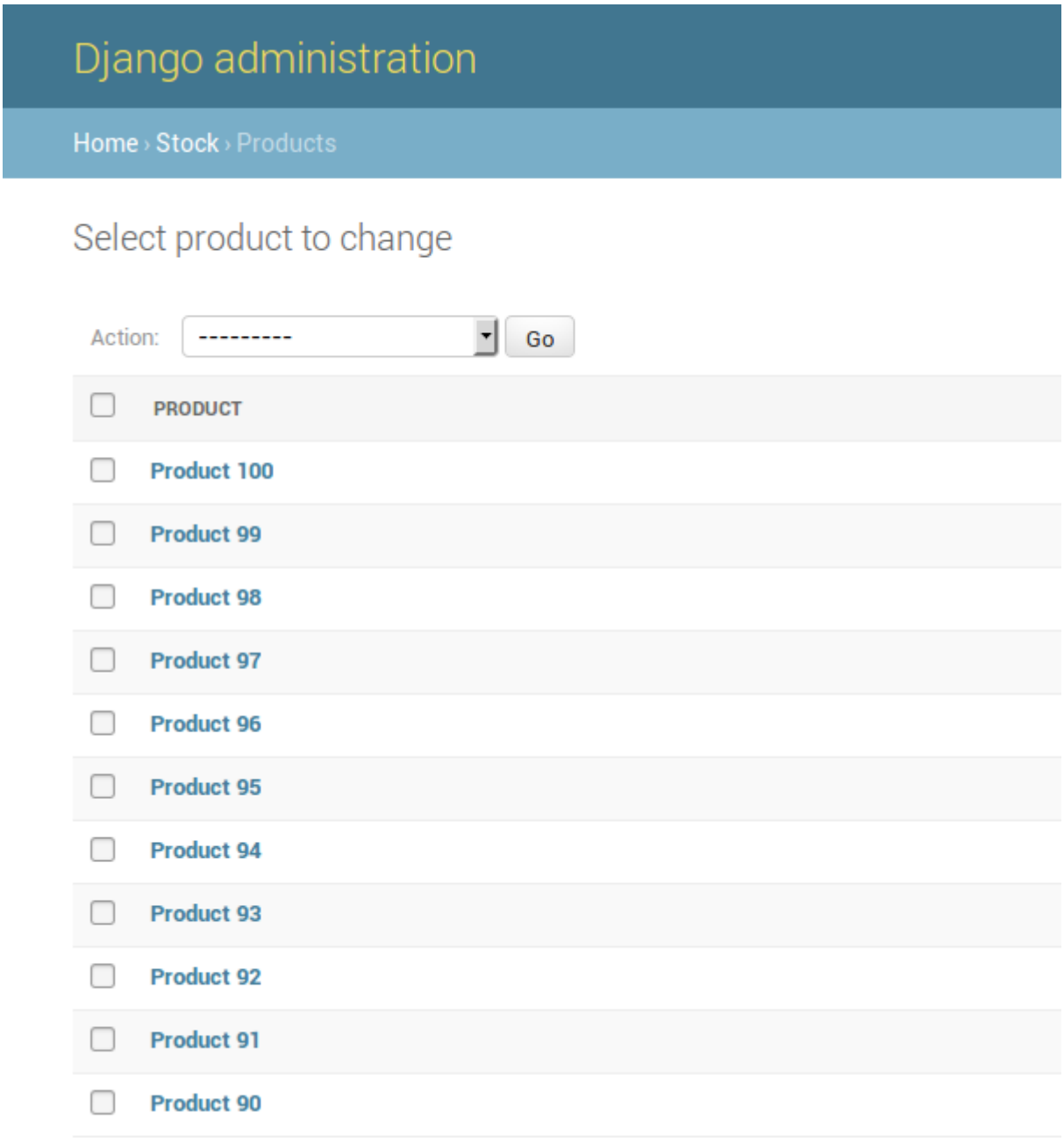
```
# python manage.py shell

from stock.models import Product


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

Ci - dessus `for` la boucle crée (ce qui signifie que les données sont enregistrées dans la base de données) 100 produits (entrées) avec des noms de `Product 1` , `Product 2` , ... `Product 100` et les prix `1` , `2` , ..., `100` .

Pour voir ces produits via la page d'administration de Django, rendez-vous simplement sur le site `127.0.0.1:8000/admin/` et cliquez sur le lien `Products` :



Profitez de vos 100 produits générés automatiquement:



Ceci est connu comme la page de `change list` de Django. Maintenant, pour que cela soit plus joli, éditez votre fichier `stock/admin.py` et entrez:

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

Maintenant, appuyez sur Actualiser et vous devriez voir une deuxième colonne affichant les prix.

Phew! Final des actions

Pour récapituler, nous avons le modèle et nous avons les entrées. Ensuite, nous voulons créer une action qui, une fois sélectionnée, fera une remise de 30% sur le ou les produits sélectionnés.

Avez-vous remarqué qu'il existe déjà une boîte de sélection en haut de la page de `change list` , avec l'étiquette `Action` ? Django ajoute automatiquement une action par défaut sur chaque entrée qui exécute l'action **D** elete.

Les actions d'administration de Django sont écrites en tant que fonctions simples. Permet de plonger. Éditez le fichier `stock/admin.py` et ajoutez ce qui suit:

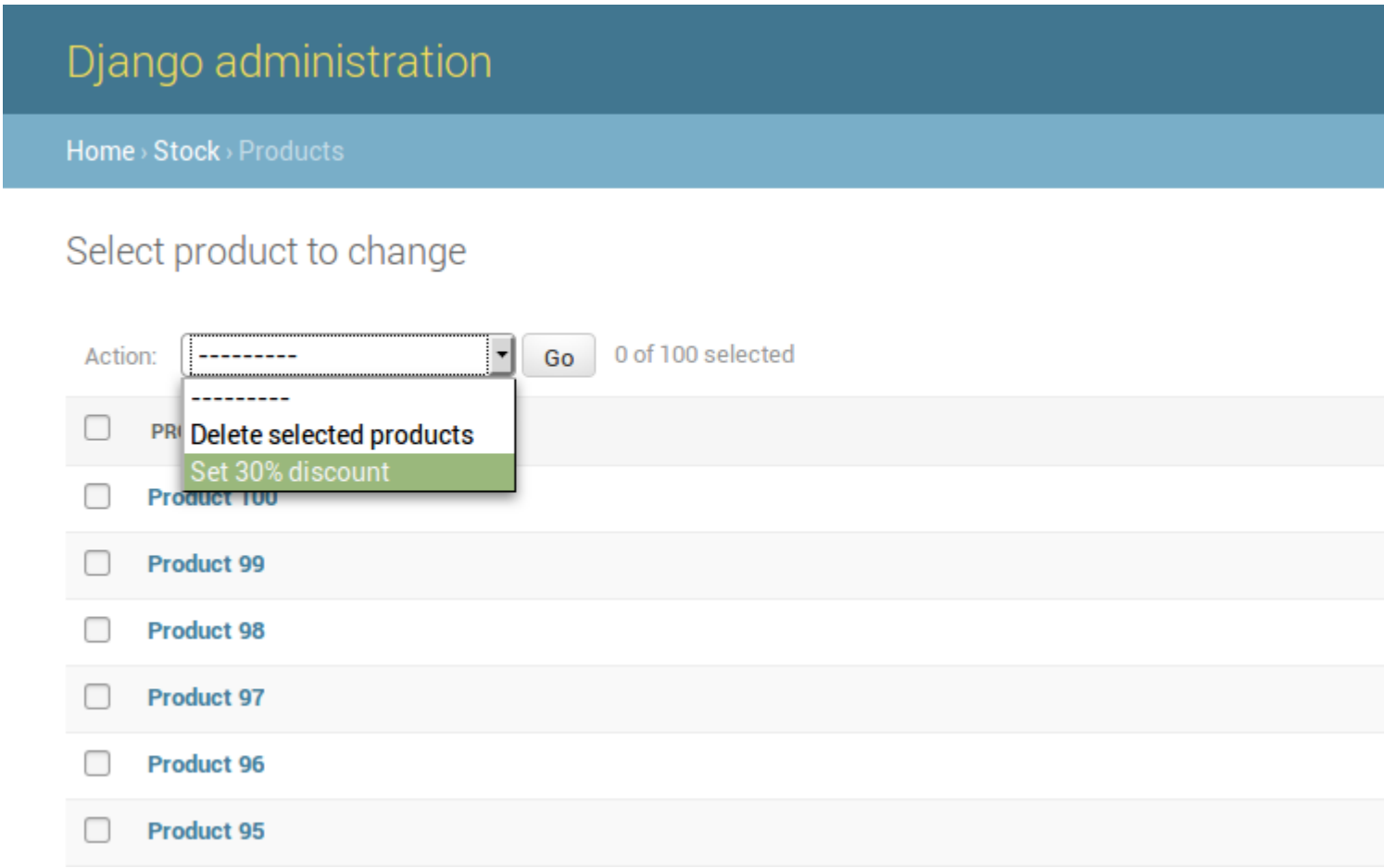
```
# admin.py

@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'
```

Quelques points à noter ici:

- La classe `ProductAdmin` a un attribut supplémentaire (`actions`) qui est une liste de chaînes (chaque chaîne est le nom de la fonction qui représente l'action).
- La fonction-action qui est une méthode de la classe `ProductAdmin` . Il prend comme arguments l'instance `ModelAdmin` (`self` puisqu'il s'agit d'une méthode), l'objet de `HTTP request` et le `queryset` (une liste des objets-entrées-produits sélectionnés).
- La dernière ligne est un attribut de fonction (`short_description`) qui définit le nom affiché dans la boîte de sélection des actions (il y a un double% pour échapper au% unique).

Dans la fonction-action, nous effectuons une itération sur chaque produit (qui a été sélectionné) et nous avons défini sa valeur réduite de 30%. Ensuite, nous appelons la méthode `save()` avec l'argument `update_fields` afin de forcer une mise à jour sur les champs inclus dans la liste `update_fields` (au lieu d'une UPDATE sur tous les champs du modèle) dans la base de données, pour des raisons de performances (pas un gain de performance dans cet exemple, avec seulement 2 colonnes, mais vous obtenez le point).

Maintenant, appuyez sur Actualiser dans la page de `change list` et vous devriez voir votre action sous celle de `delete` . Allez-y et sélectionnez certains produits (en utilisant la case à cocher située à gauche de chaque produit ou en utilisant la case à cocher située en haut à gauche), sélectionnez l'action `Set 30% discount` et cliquez sur le bouton `Go` . **C'est tout!**



Bien sûr, ce n'est pas très pratique dans la plupart des situations, car cette action ne vous permet pas d'entrer un montant de remise différent. Vous devez modifier le fichier `admin.py` chaque fois que vous souhaitez appliquer une remise différente. Dans l'exemple à venir, nous verrons comment procéder.

Lire Actions d'administration en ligne: https://riptutorial.com/fr/django-admin/topic/7747/actions-d-administration

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec django-admin	Antoine Pinsard, Community, lxer, NeErAj KuMaR, Roald Nefs, Udi
2	Actions d'administration	ira, nik_m, Stryker