Kostenloses eBook

LERNEN

django-admin

Free unaffiliated eBook created from
Stack Overflow contributors.

#django-admin

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit django-admin

Bemerkungen

Django Admin ist die CRUD-Schnittstelle des Django- Web-Frameworks. Es wird meist automatisch generiert, kann aber weitgehend angepasst werden. Sie müssen jedoch beachten, dass es nur **für vertrauenswürdige Benutzer** gedacht ist und Grenzen hat. In keinem Fall dürfen Sie **nicht vertrauenswürdigen Benutzern Administratorzugriff gewähren** .

Django Admin bietet ein hohes Maß an Anpassungsmöglichkeiten. Achten Sie jedoch darauf, nicht zu viele Anpassungsdetails zu verlieren. Wenn Sie dies tun, ist es wahrscheinlich an der Zeit, eine eigene Benutzeroberfläche ohne Django Admin zu erstellen.

Ressourcen

- Offizielle Django Admin-Einführung
- Offizielles Tutorial für Django Admin
- Offizielle Dokumentation für Django Admin
- Django-Admin-Quellcode

Versionen

Ausführung	Veröffentlichungsdatum
1.10	2106-08-01
1,9	2015-12-01
1.8	01.04.2015
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

Django-Administrator einrichten

Alles, was Sie benötigen, um mit dem Django-Administrator zu beginnen, ist bereits in Djangos Standardprojektlayout eingerichtet. Das beinhaltet:

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

Seien Sie vorsichtig mit `urls.py` , das in Django> = 1.9 etwas anders ist als in älteren Versionen.

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)),
]
```

Die Version mit `include` wird in Django 1.9 weiterhin funktionieren, ist aber veraltet und wird in der Zukunft entfernt.

Falls noch nicht geschehen, müssen Sie die Basismigrationen anwenden:

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

Um auf den Admin zuzugreifen, müssen Sie auch einen Superuser erstellen mit:

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

Sobald dies erledigt ist, können Sie Ihren Server ausführen:

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

Und besuchen Sie die Admin-Seite unter http://127.0.0.1:8000/admin/.

Fügen Sie den Admin-Seiten ein Modell hinzu

Wenn Sie Ihre eigenen Modelle in einer App erstellt haben, müssen diese noch *registriert* werden, um auf den Verwaltungsseiten verfügbar zu werden.

Dies geschieht im `admin` Modul. Wenn Ihre App mit `manage.py startapp` , sollte sich bereits eine `admin.py` Datei in `admin.py` App-Modul befinden. Ansonsten erstellen Sie es.

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

admin.site.register(MyModel)
```

Alle Optionen sind in der ModelAdmin-Unterklasse definiert. einige Optionen:

```
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)
```

Eine `admin.register` Methode zum Registrieren eines Modells ist die Verwendung des Dekorators `admin.register` :

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

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

Entfernen Sie ein Modell aus den Admin-Seiten

Django Admin wird standardmäßig mit einigen registrierten Modellen geliefert. Es gibt einige Situationen, in denen Sie ein Modell möglicherweise von den Verwaltungsseiten entfernen möchten.

Dies geschieht im `admin` Modul. Wenn Ihre App mit `manage.py startapp` , sollte die Datei `admin.py` bereits in Ihrem App-Modul liegen. Ansonsten erstellen Sie es.

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

admin.site.unregister(User)
```

Passen Sie das django User Admin-Modell an

```
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)
```

Wir müssen die Registrierung des benutzerdefinierten UserAdmin aufheben, da in django User Model Admin bereits registriert ist. Daher müssen Sie zuerst das User Model in unserer admin.py abmelden. Anschließend können Sie das User Model mit dem benutzerdefinierten ModelAdmin registrieren

Erste Schritte mit django-admin online lesen: https://riptutorial.com/de/django-admin/topic/3499/erste-schritte-mit-django-admin

Kapitel 2: Admin-Aktionen

Bemerkungen

Ich glaube nicht, dass Sie get_price (self) benötigen, insbesondere wenn Sie keine Änderungen an der Preisvariable vornehmen. Ich würde die Methode get_price entfernen, da Sie unter dem Produktmodell den gleichen Wert vom Preis erhalten können. Ich könnte falsch liegen. Der Wert der get_price-Methode wird hier nicht angezeigt.

Examples

Aktion für Produktrabatte

Eines Tages hatte ich ein Gespräch mit einem Freund von mir, der in seinem Job das Laravel PHP-Framework verwendet. Als ich ihm erzählte, dass Django ein eigenes, vollständig integriertes HTML-CRUD-System für die Interaktion mit der Datenbank namens Django-Administrator hat , blickten seine Augen auf! Er sagte mir: " *Ich habe Monate gebraucht, um eine Admin-Oberfläche für meine aktuelle Web-App zu erstellen, und Sie sagen, dass Sie alle haben, ohne eine einzige Zeile Code schreiben zu müssen?* " Ich antwortete " *Yeap!* "

Django-Admin ist eine leistungsstarke Funktion von Django, die viele Goodies bietet. Eine davon sind Aktionen .

Aber was sind " *Aktionen* "?

Angenommen, Sie haben ein Modell und Sie haben bereits einige Einträge (vielleicht Hunderte, vielleicht Tausende) hinzugefügt. Jetzt möchten Sie eine `python manage.py shell` auf mindestens eine davon anwenden, **ohne** die Konsole zu verwenden (`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'])
```

Haben Sie bemerkt, dass wir den Rabatt auf **alle** Produkte angewendet haben. Was wäre, wenn wir diese Logik auf bestimmte anwenden wollten? Oder wenn wir den Rabattwert manuell eingeben und diesen Wert dann auf einige Produkte anwenden wollten? All dies durch den

Django-Admin! Du bist auf dem richtigen Weg. Django-Aktionen FTW. Schauen wir uns ein vollständiges Beispiel an.

Die Grundlagen:

- Python 3.4.3
- Django 1.10
- SQLite (in Python integriert, keine zusätzliche Installation erforderlich)

Das Model:

- Darstellung eines Produkts für unseren E-Shop
- Preis eines Produkts wäre ganze Zahlen (keine Dezimalzahlen)

Das Ziel:

- Um über die Django Admin-Oberfläche einen festen Rabatt auf einen oder mehrere Produkteinträge anwenden zu können.

Setup (App, Modell und Django Admin)

Angenommen , Sie haben bereits begonnen , `manage.py stock` ein Projekt , in das Verzeichnis , in dem `manage.py` Leben und schafft eine App mit dem Namen `stock` , indem Sie eingeben:

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

Django erstellt automatisch eine Verzeichnisstruktur für Sie. Gehen `models.py` Datei `models.py` und bearbeiten `models.py` :

```
# 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
```

Unser `Product` wurde erstellt, aber noch nichts in der Datenbank. Damit die Migration funktionieren kann, muss unsere App in der Liste `INSTALLED_APPS` .

Bearbeiten Sie Ihre Datei `settings.py` und fügen Sie unter der Liste `INSTALLED_APPS` hinzu:

```
# settings.py

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

Jetzt ausführen:

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

Nach dem `migrate` verfügt Ihre Datenbank jetzt über eine Tabelle mit dem Namen `product` die die drei Spalten `id` , `name` und `price` . So weit, ist es gut!

Wenn Sie nichts an Ihrer ROOT_URLCONF- Datei geändert haben, die sich normalerweise im Ordner `<your_project_name>/<your_project_name>/` , sollte die URL, die auf die Django-Administrationssite verweist, folgendermaßen lauten:

```
# urls.py

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

Bis jetzt haben wir uns bei den Django-Admin-Aktionen nichts genauer angeschaut. `stock/admin.py` Sie einen letzten Schritt und fügen Sie diese in Ihre `stock/admin.py` Datei ein:

```
# admin.py

from django.contrib import admin

from .models import Product


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

OK. Das Setup ist abgeschlossen. Um sicherzustellen, dass alles funktioniert, führen Sie Folgendes aus:

```
python manage.py runserver
```

und besuchen Sie mit Ihrem bevorzugten Browser die Seite `127.0.0.1:8000/admin/` . Sie sollten die glänzend-glamourös-grandios Django Admin - Seite sehen, die Sie reate- **R** ead- **U** pdate- **D** ÉLETE Ihre **C** ermöglicht `Product` Wenn Sie zufällig auf der obigen Seite nach einem Benutzernamen und einem Kennwort gefragt werden, haben Sie kein Problem. Sie haben gerade keinen `User` , um sich beim Admin anzumelden. Einfach ausführen:

```
python manage.py createsuperuser
```

Geben Sie Ihren Namen, Ihre E-Mail-Adresse, Ihren Benutzernamen und Ihr Passwort (zweimal) ein und fertig.

Erzeugen Sie einige gefälschte Produkte

Bisher haben wir das Modell erstellt, aber noch keine Einträge (keine Produkte). Wir brauchen einige Einträge, um die Macht der Django-Verwaltungsaktionen zu beleuchten.

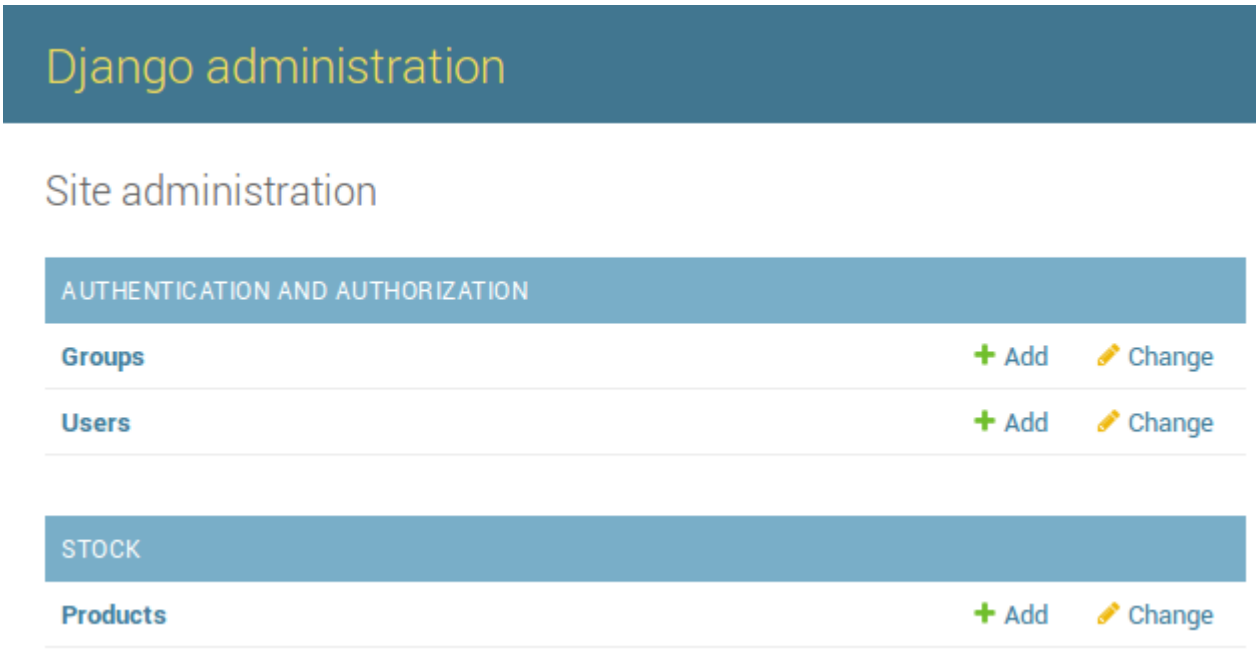
Wir werden 100 Produkte kreieren und damit arbeiten. Anstatt den ADD-Button manuell zu drücken und einen `name` und einen `price` einzugeben, schreiben wir ein Skript, das die Arbeit für uns erledigt.

Führen `python manage.py shell` und geben Sie Folgendes ein:

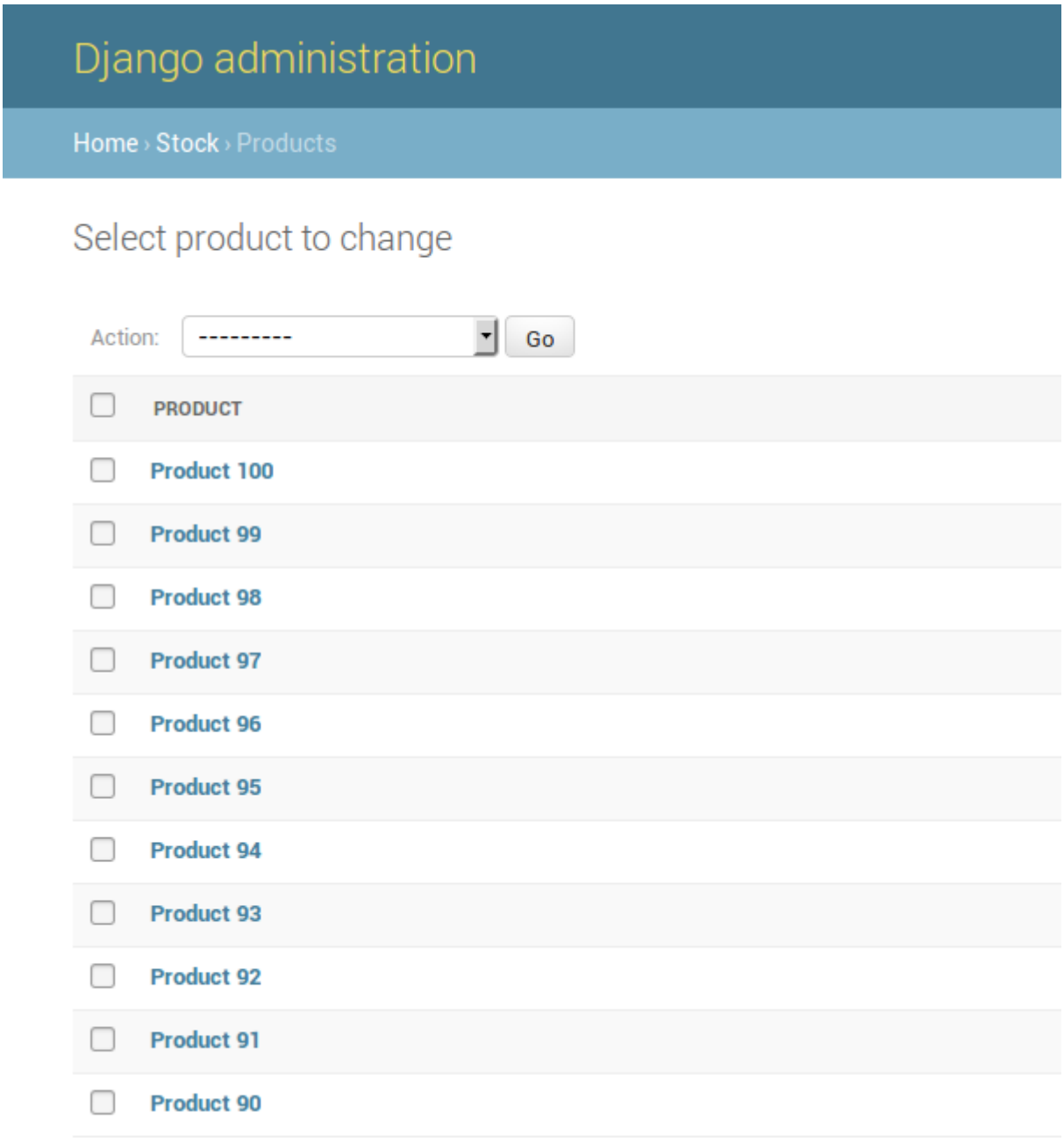
```
# python manage.py shell

from stock.models import Product


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

Die obige `for` Schleife erstellt (dh Daten werden in der Datenbank gespeichert) 100 Produkte (Einträge) mit den Namen `Product 1` , `Product 2` , ... `Product 100` und Preise `1` , `2` , ..., `100` .

Um diese Produkte über die Django-Administrationsseite anzuzeigen, besuchen Sie einfach erneut `127.0.0.1:8000/admin/` und klicken Sie auf den Link `Products` :



Genießen Sie Ihre automatisch generierten 100 Produkte:



Dies wird als `change list` des Django bezeichnet. Damit dies nun viel `stock/admin.py` aussieht, bearbeiten Sie Ihre `stock/admin.py` Datei und geben Sie `stock/admin.py` ein:

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

Klicken Sie nun auf "Aktualisieren". In einer zweiten Spalte werden die Preise angezeigt.

Puh! Beenden Sie die Aktionen

Zusammenfassend haben wir das Modell und die Einträge. Als Nächstes möchten wir eine Aktion erstellen, die nach der Auswahl einen Rabatt von 30% auf die ausgewählten Produkte gewährt.

Haben Sie festgestellt, dass sich oben auf der `change list` bereits ein Auswahlfeld mit der Bezeichnung `Action` ? Django fügt automatisch für jeden Eintrag eine Standardaktion hinzu, die die Aktion **D** elete ausführt.

Django-Admin-Aktionen werden als einfache Funktionen geschrieben. Lasst `stock/admin.py` eintauchen. Bearbeiten Sie die Datei `stock/admin.py` und fügen Sie Folgendes hinzu:

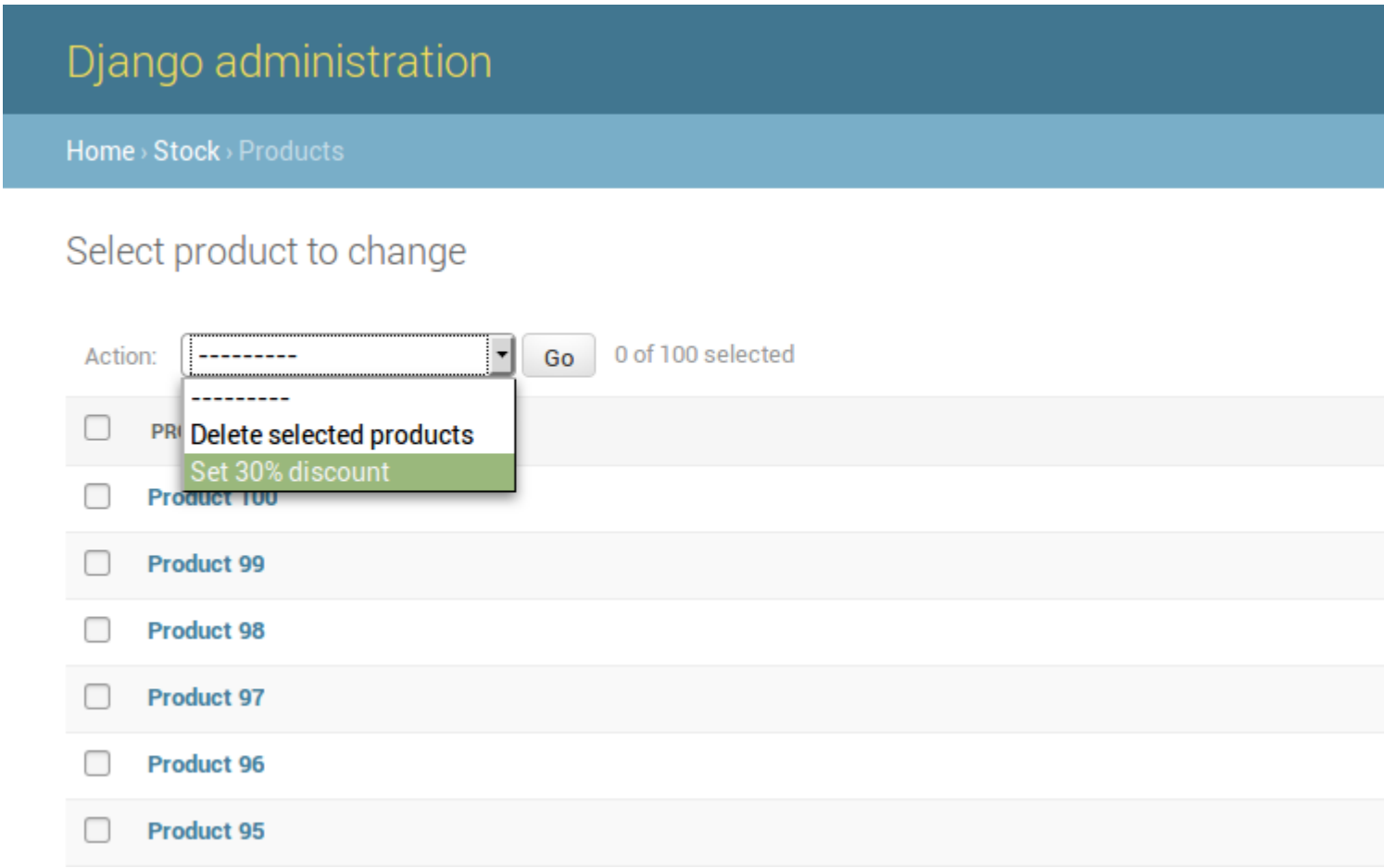
```
# admin.py

@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'
```

Ein paar Dinge, die Sie hier beachten sollten:

- Die `ProductAdmin` Klasse verfügt über ein zusätzliches Attribut (`actions`), das eine Liste von Zeichenfolgen ist (jede Zeichenfolge ist der Name der Funktion, die die Aktion darstellt).
- Die Aktionsfunktion, die eine Methode der `ProductAdmin` Klasse ist. Als Argumente werden die `ModelAdmin` Instanz (`self` da dies eine Methode ist), das `HTTP request` und das `queryset` (eine Liste der ausgewählten Objekte-Einträge-Produkte) als Argumente verwendet.
- Die letzte Zeile ist ein Funktionsattribut (`short_description`), das den angezeigten Namen innerhalb des `short_description` festlegt (dort ist ein doppeltes%, um das einzelne% zu `short_description`).

Innerhalb der Funktions-Aktion wird jedes Produkt (das ausgewählt wurde) iteriert, und der Wert wurde um 30% verringert. Dann rufen wir die `save()` -Methode mit dem Argument `update_fields` auf, um aus Performance-Gründen (statt eines UPDATE für alle Felder des Modells) in der Datenbank ein UPDATE für die Felder zu erzwingen, die in der `update_fields` Liste enthalten sind (nicht Ein Leistungsgewinn in diesem Beispiel mit nur 2 Spalten, aber Sie erhalten den Punkt).

Klicken Sie nun in der `change list` auf Aktualisieren, und Ihre Aktion sollte unter `delete` . Gehen Sie weiter und wählen Sie einige Produkte (mit der Option auf der linken Seite jeder oder alle Produkte die linke oberste Checkbox), wählen Sie das `Set 30% discount` Aktion und klicken Sie auf die `Go` - Taste. **Das ist es!**



In den meisten Situationen ist dies natürlich nicht sehr praktisch, da Sie mit dieser Aktion keinen anderen Rabattbetrag eingeben können. Sie müssen die Datei `admin.py` jedes Mal bearbeiten, `admin.py` Sie einen anderen Rabatt anwenden möchten. Im nächsten Beispiel werden wir sehen, wie das geht.

Admin-Aktionen online lesen: https://riptutorial.com/de/django-admin/topic/7747/admin-aktionen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit django-admin	Antoine Pinsard, Community, lxer, NeErAj KuMaR, Roald Nefs, Udi
2	Admin-Aktionen	ira, nik_m, Stryker