Бесплатная электронная книга

УЧУСЬ

django-admin

Free unaffiliated eBook created from
Stack Overflow contributors.

#django-admin

Около

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

глава 1: Начало работы с django-admin

замечания

Django Admin - это интерфейс CRUD в веб-среде Django . Он в основном автоматически генерируется, но может быть широко настроен. Однако вы должны иметь в виду, что он предназначен только **для доверенных пользователей** и имеет свои пределы. В любом случае вы **никогда не** должны **предоставлять доступ администратора к ненадежным пользователям** .

Django Admin обеспечивает высокий уровень настройки, но будьте осторожны, чтобы не падать слишком много деталей настройки. Если вы это сделаете, возможно, настало время создать собственный интерфейс без Django Admin.

Ресурсы

- Официальное введение администратора Django
- Официальный учебный курс Django Admin
- Официальная документация администратора Django
- Исходный код администратора Django

Версии

Версия	Дата выхода
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

Версия	Дата выхода
1,0	2008-09-03

Examples

Настройка Django Admin

Все, что вам нужно для начала работы с администратором Django, уже настроено в макете проекта Django по умолчанию. Это включает:

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

Будьте осторожны с `urls.py` который немного отличается в Django> = 1.9, чем в более старых версиях.

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)),
]
```

Версия с `include` будет по-прежнему работать в Django 1.9, но устарела и будет удалена в будущем.

Если это еще не сделано, вы должны применить базовые миграции:

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

Чтобы получить доступ к администратору, вам также необходимо создать суперпользователя с:

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

Как только это будет сделано, вы можете запустить свой сервер:

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

Посетите страницу администратора по адресу http://127.0.0.1:8000/admin/.

Добавление модели к страницам администратора

Когда вы создавали свои собственные модели в приложении, они все равно должны быть *зарегистрированы* , чтобы стать доступными на страницах администратора.

Это делается в подмодуле `admin` . Если ваше приложение было создано с помощью `manage.py startapp` , файл `admin.py` должен уже `admin.py` в вашем модуле приложения. В противном случае создайте его.

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

admin.site.register(MyModel)
```

Все параметры определены в подклассе ModelAdmin. некоторые варианты:

```
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)
```

Более краткий способ регистрации модели - использовать декоратор `admin.register` :

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

Удаление модели из страниц администратора

Django Admin поставляется с некоторыми моделями, зарегистрированными по умолчанию. В некоторых случаях вы можете удалить модель из страниц администратора.

Это делается в подмодуле `admin` . Если ваше приложение было создано с помощью `manage.py startapp` , файл `admin.py` должен быть уже установлен в вашем модуле приложения. В противном случае создайте его.

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

admin.site.unregister(User)
```

Настроить 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)
```

Нам нужно отменить регистрацию до регистрации пользовательского UserAdmin, потому что в django User Model Admin уже зарегистрирован. Поэтому нам нужно сначала отменить регистрацию модели пользователя в нашем admin.py, тогда мы можем зарегистрировать модель пользователя с пользовательским ModelAdmin

Прочитайте Начало работы с django-admin онлайн: https://riptutorial.com/ru/django-admin/topic/3499/начало-работы-с-django-admin

глава 2: Действия администратора

замечания

Я не думаю, что вам нужно get_price (self) специально, когда вы не вносите никаких изменений в ценовую переменную. Я бы удалил метод get_price, так как вы можете получить то же значение от цены под моделью продукта. Я могу ошибаться. Просто не вижу значения метода get_price здесь.

Examples

Действие скидок на акции

Однажды я поговорил с моим другом, который использует Laravel PHP framework в своей работе. Когда я сказал ему, что у Django есть своя собственная система HTML CRUD, для взаимодействия с базой данных, называемой администратором Django , его глаза выскочили! Он сказал мне: « *Мне потребовались месяцы, чтобы создать интерфейс администратора для моего текущего веб-приложения, и вы говорите, что у вас есть все это без необходимости писать одну строку кода?* ». Я ответил « *Дай!* ».

Администратор Django - это мощная функция Django, которая предоставляет множество положительных героев. Одно из них - это действия .

Но какие « *действия* » есть?

Предположим, у вас есть модель, и вы уже добавили в нее несколько записей (возможно, сотни, а может быть тысячи). Теперь вы хотите применить правило-действие хотя бы к одному из них, **не** используя консоль (`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'])
```

Вы заметили, что мы применили скидку на **все** продукты. Что, если мы хотим применить эту

логику к конкретным? Или, если мы хотим вручную ввести значение скидки, а затем применить это значение к некоторым продуктам? Все это через администратора Django! Ты на правильном пути. Действия Django FTW. Давайте посмотрим на полный пример.

Основы:

- Python 3.4.3
- Django 1.10
- SQLite (встроенный Python, нет необходимости в дополнительной установке)

Модель:

- Представление продукта для нашего интернет-магазина
- Цена продукта будет целым числом (не десятичным)

Цель:

- Чтобы иметь возможность через интерфейс Django Admin применять фиксированную скидку на одну или несколько записей продукта.

Настройка (приложение, модель и администратор Django)

Предполагая, что вы уже начали проект , перейдите в каталог, где `manage.py` живет и создайте приложение с `stock` имен, набрав:

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

Django автоматически создаст для вас структуру каталогов. Перейдите и отредактируйте файл `models.py` и добавьте:

```
# 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
```

Наша модель `Product` создана, но ничего в базе данных пока нет. Чтобы миграция работала, наше приложение должно быть включено в список `INSTALLED_APPS` .

Измените файл `settings.py` и в списке `INSTALLED_APPS` добавьте:

```
# settings.py

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

Теперь запустите:

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

После команды `migrate` ваша база данных теперь имеет таблицу с именем `product` которая содержит три столбца, `id` , `name` и `price` . Все идет нормально!

Если вы не изменили ничего в вашем файле ROOT_URLCONF , который обычно живет внутри папки `<your_project_name>/<your_project_name>/` , тогда URL-адрес, указывающий на сайт администратора Django, должен быть:

```
# urls.py

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

До сих пор мы не видели ничего конкретного о действиях администратора Django. Сделайте последний шаг и добавьте их в свой файл `stock/admin.py` :

```
# admin.py

from django.contrib import admin

from .models import Product


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

ХОРОШО. Настройка выполнена. Просто чтобы убедиться, что все работает, запустите:

```
python manage.py runserver
```

и с вашим любимым браузером зайдите на страницу `127.0.0.1:8000/admin/` . Вы должны увидеть блестящую-гламурную-потрясающую страницу администратора Django, которая

позволяет **C** reate- **R** ead- **U** pdate- **D** алить ваш `Product` модель! Если случайно, приведенная выше страница запрашивает у вас имя пользователя / пароль, и у вас его нет, не проблема. Вы просто не создали `User` для входа в админ. Просто запустите:

```
python manage.py createsuperuser
```

введите свое имя, адрес электронной почты, имя пользователя и пароль (дважды), и все готово.

Создание некоторых поддельных продуктов

Пока мы создали модель, но нет записей (без продуктов). Нам нужны некоторые записи, чтобы пролить свет на действия администратора Django.

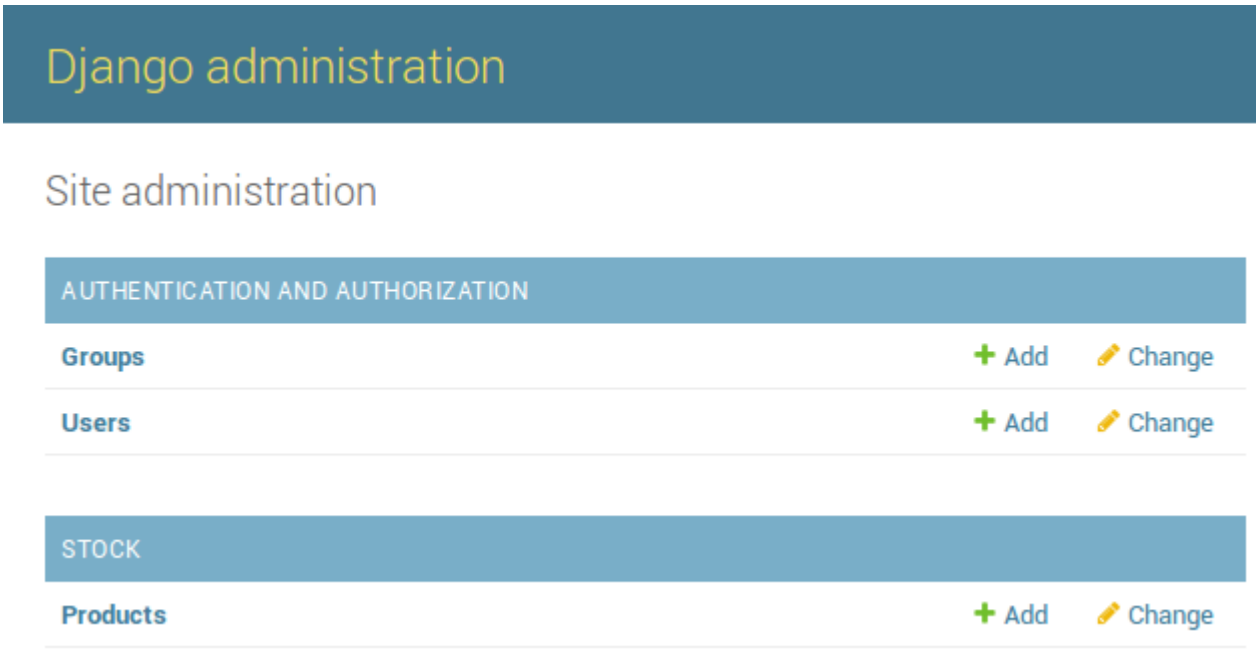
Мы собираемся создать 100 продуктов и работать с ними. Но вместо того, чтобы вручную нажимать кнопку ADD и вводить `name` и `price` , мы напишем сценарий для выполнения этой работы для нас.

Запустите `python manage.py shell` и введите следующее:

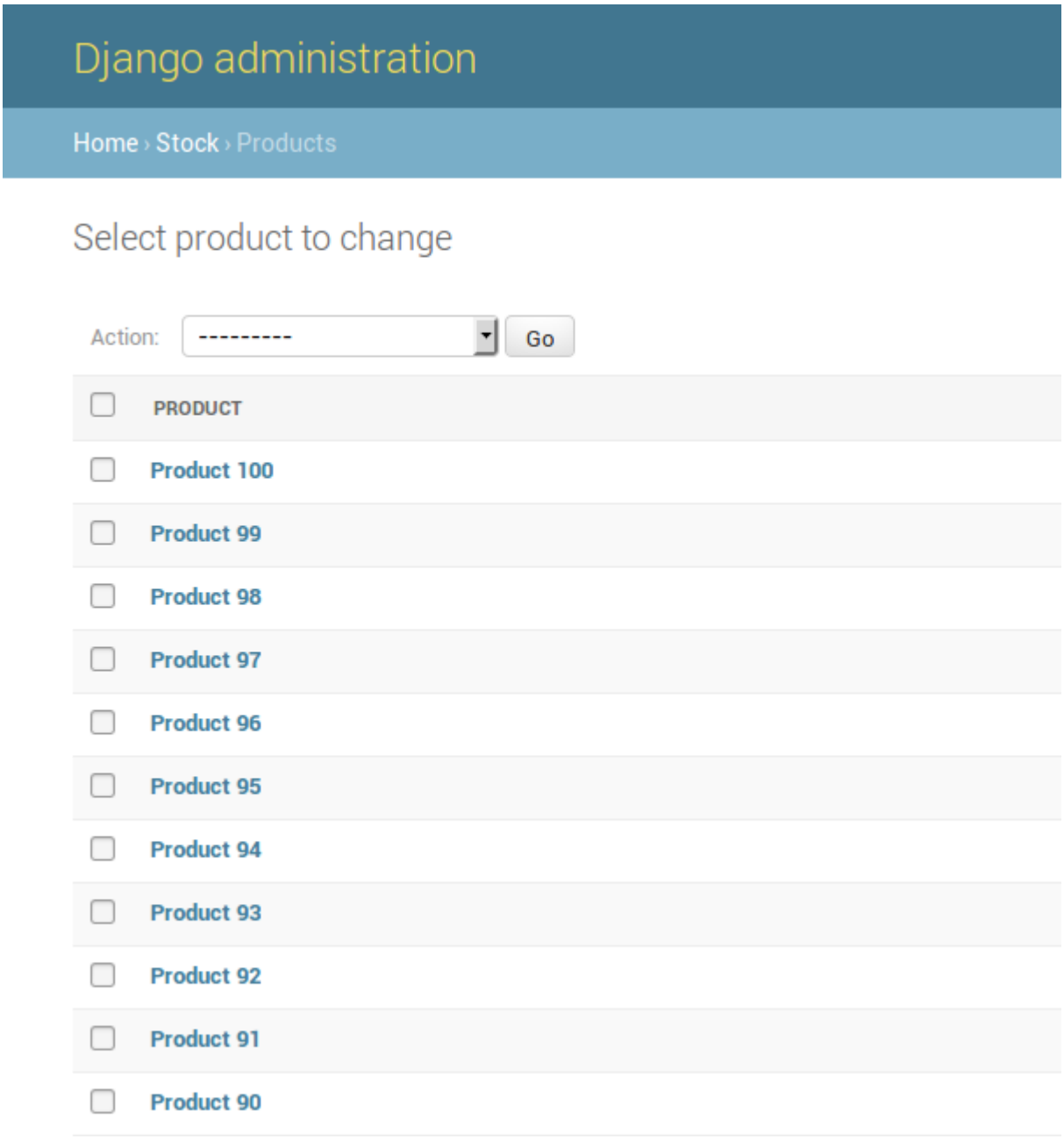
```
# python manage.py shell

from stock.models import Product


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

Вышеизложенное `for` цикла создает (что означает, что данные сохраняются в базе данных) 100 продуктов (записей) с именами `Product 1` , `Product 2` , ... `Product 100` и цены `1` , `2` , ..., `100` .

Чтобы просмотреть эти продукты с помощью страницы администрирования Django, просто зайдите еще раз `127.0.0.1:8000/admin/` и нажмите ссылку « `Products` :



Наслаждайтесь автогенерируемыми 100 продуктами:



Это известно как страница `change list` Django. Теперь, чтобы это выглядело намного красивее, отредактируйте файл `stock/admin.py` и введите:

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

Теперь нажмите «Обновить», и вы увидите второй столбец, отображающий цены.

Уф! Завершите действия

Напомним, у нас есть модель, и у нас есть записи. Затем мы хотим создать действие,

которое после его выбора сделает скидку 30% на выбранный продукт (ы).

Вы заметили, что уже есть поле выбора в верхней части страницы `change list` с надписью `Action` ? Django автоматически добавляет действие по умолчанию для каждой записи, которая выполняет действие **D** elete.

Действия администратора Django записываются как простые функции. Давайте `stock/admin.py` файл `stock/admin.py` и добавьте следующее:

```
# 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'
```

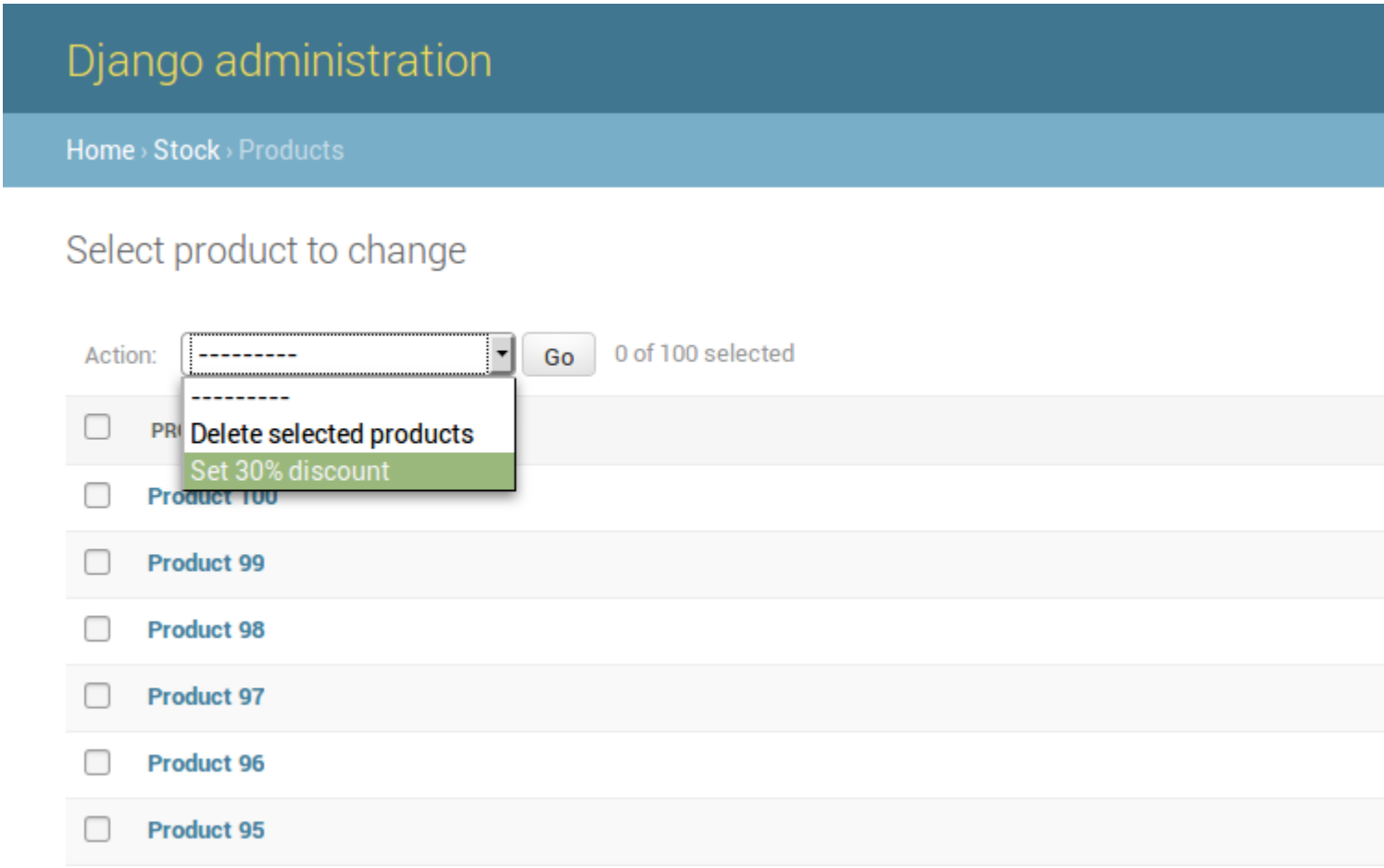
Несколько вещей, чтобы отметить здесь:

- Класс `ProductAdmin` имеет один дополнительный атрибут (`actions`), который представляет собой список строк (каждая строка - это имя функции, которая представляет действие).
- Действие-функция, которая является методом класса `ProductAdmin` . Он принимает в качестве аргументов `ModelAdmin` экземпляр (`self` , так как это метод), то `HTTP request` объект и `queryset` (список выбранных объектов-записей продуктов).
- Последняя строка - это атрибут функции (`short_description`), который устанавливает отображаемое имя внутри поля выбора действий (для выхода из единственного% есть двойной%).

Внутри функции-действия мы перебираем каждый продукт (который был выбран), и мы устанавливаем его значение на 30%. Затем мы вызываем метод `save()` с аргументом `update_fields` , чтобы заставить UPDATE в полях, которые включены в список `update_fields` (вместо UPDATE во всех полях модели) в базе данных, по причинам производительности (не прирост производительности в этом примере, всего 2 столбца, но вы получите точку).

Теперь нажмите обновление на странице `change list` и вы увидите свое действие под `delete` . Вперед и выберите некоторые продукты (используя флажок слева от каждого или всех продуктов, используя флажок слева сверху), выберите действие « `Set 30% discount` и

нажмите кнопку « `Go` . **Это оно!**



Конечно, это не очень удобно в большинстве ситуаций, так как это действие не позволяет вам вводить другую скидку. Вы должны отредактировать файл `admin.py` каждый раз, когда хотите применить другую скидку. В следующем примере мы увидим, как это сделать.

Прочитайте Действия администратора онлайн: https://riptutorial.com/ru/django-admin/topic/7747/действия-администратора

кредиты

S. No	Главы	Contributors
1	Начало работы с django-admin	Antoine Pinsard, Community, lxer, NeErAj KuMaR, Roald Nefs, Udi
2	Действия администратора	ira, nik_m, Stryker