adminsortable2 Documentation

Release 0.2.7

Jacob Rief

Dec 04, 2017

Contents

This Django module is as a replacement for django-admin-sortable using an unintrusive approach.

It is a generic drag-and-drop ordering module for sorting objects in the list view of the Django admin interface. This plugin offers simple mixin classes which enrich the functionality of *any* existing class derived from `admin.ModelAdmin`, `admin.StackedInline` or `admin.TabularInline`. It thus makes it very easy to integrate with existing models and their model admin interfaces.

Project home: https://github.com/jrief/django-admin-sortable2

Please ask questions and report bugs on: https://github.com/jrief/django-admin-sortable2/issues

CHAPTER 1

Why another adminsortable plugin?

All available plugins which add functionality to make list views for the Django admin interface sortable, offer a base class to be used instead of `models.Model`. This abstract base class then contains a hard coded position field, additional methods, and meta directives. The problem with such an approach is twofold:

First, an *is-a* relationship is abused to enrich the functionality of a class. This is bad OOP practice. A base class shall only be used to reflect a real *is-a* relation which specializes this class. For instance: A mammal *is an* animal, a primate *is a* mammal, homo sapiens *is a* primate, etc. Here the inheritance model is appropriate, but it would be wrong to derive from homo sapiens to reflect a human which is able to hunt using bows and arrows.

Therefore, a sortable model *is not an* unsortable model. Making a Django Model sortable enriches its functionality. In OOP design this does not qualify for an *is-a* relationship.

Fortunately, Python makes it very easy to distinguish between real *is-a* relationships and interfaces enriching their functionality. The latter are handled by so named mixin classes. They offer additional methods for a class without deriving from the base class.

Also consider the case when someone wants to augment some other functionality of a model class. If he also derives from `models.Model`, he would create another abstract intermediate class. Someone who wants to use *both* functional augmentations, now is in trouble. He has to choose between one of the two, as he cannot derive from both of them, because then its model class would inherit the base class `models.Model` twice. This kind of diamond-shaped inheritance is to be avoided under all circumstances.

By using a mixin class rather than deriving from a special abstract base class, these problems can be avoided!

CHAPTER 2

Contents:

2.1 Installation

Install **django-admin-sortable2**. The latest stable release can be found on PyPI

```
pip install django-admin-sortable2
```

or the newest development version from GitHub

```
pip install -e git+https://github.com/jrief/django-admin-sortable2#egg=django-admin-
↪sortable2
```

2.1.1 Configuration

Add `'adminsortable2'` to the list of `INSTALLED_APPS` in your project's `settings.py` file

```
INSTALLED_APPS = (
    ...
    'adminsortable2',
    ...
)
```

2.2 Using Admin Sortable

This Django module offers three mixin classes to be added to the existing classes of your model admin:

- `admin.ModelAdmin`
- `admin.StackedInline`
- `admin.TabularInline`

They slightly modify the admin views of a sortable model. There is no need to derive your model class from a special base model class. You can use your existing ordered field, just as you always did, or add a new one with any name you like, if needed.

2.2.1 Integrate your models

Each database model which shall be sortable, requires a position value in its model description. Rather than defining a base class, which contains such a positional value in a hard coded field, this module lets you reuse existing sort fields or define a new field for the sort value.

Therefore this module can be applied in situations where your model is derived from an existing abstract model which already contains any kind of position value. The only requirement for this module is, that this position value be specified as the primary field used for sorting. This in Django is declared through the model's Meta class. An example `models.py`:

```
class SortableBook(models.Model):
    title = models.CharField('Title', null=True, blank=True, max_length=255)
    my_order = models.PositiveIntegerField(default=0, blank=False, null=False)

    class Meta(object):
        ordering = ['my_order']
```

Here the ordering field is named `my_order`, but you may choose any other name. There are three constraints:

- `my_order` is the first field in the `ordering` tuple of the model's Meta class.
- **`my_order`'s default value must be 0. The JavaScript which performs the sorting is 1-indexed,** so this will not interfere with the order of your items, even if you're already using 0-indexed ordering fields.
- **The `my_order` field must be editable, so make sure that you do not add `editable=False`** to it.

The field used to store the ordering position may be any kind of numeric model field offered by Django. Use one of these models fields:

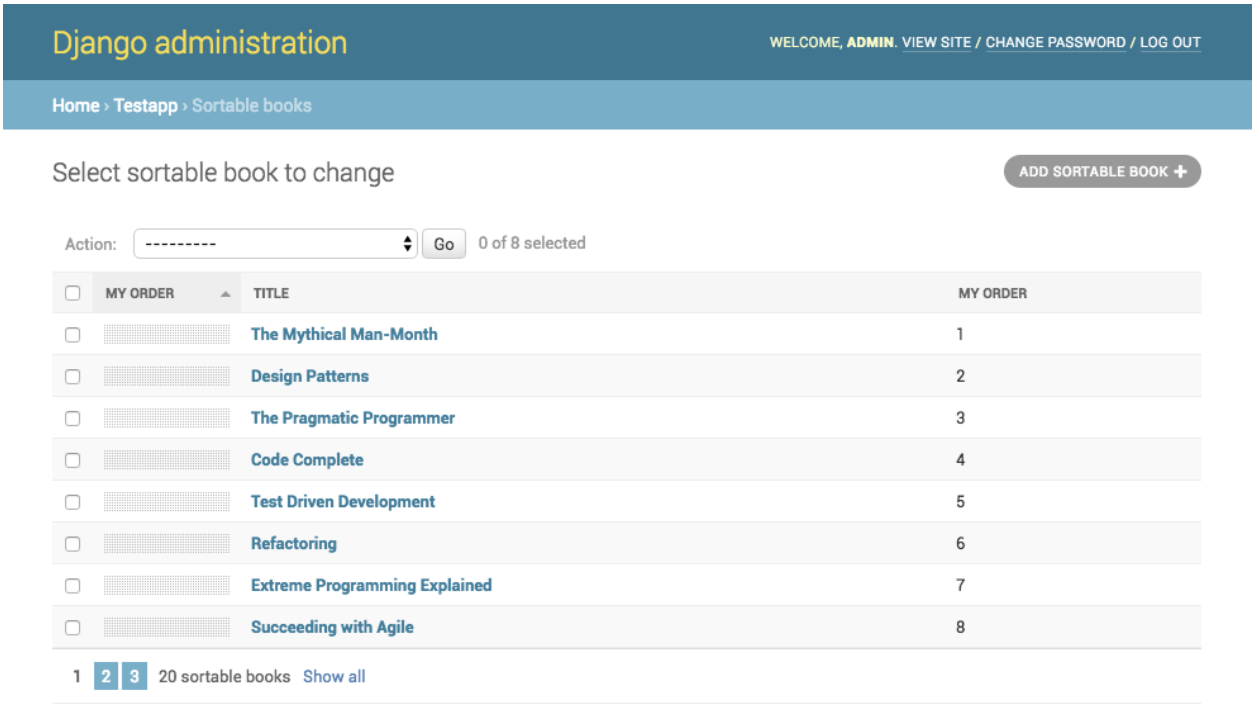
- `models.BigIntegerField`
- `models.IntegerField`
- `models.PositiveIntegerField` (recommended)
- `models.PositiveSmallIntegerField` (recommended for small sets)
- `models.SmallIntegerField`

Additionally you may use `models.DecimalField` or `models.FloatField`, but these model fields are not recommended.

> **Warning:** Do not make this field unique! See below why.

2.2.2 In Django's Admin, make the list view sortable

Next to the action checkbox, a draggable area is added to each entry line. The user than may click on any item and vertically drag that item to a new position.



Sortable List View

If one or more items shall be moved to another page, this can easily been done by selecting them though the action checkbox. Then the user shall click on a predefined action from the pull down menu on the top of the list view.

Integrate into a list view

In `admin.py`, add a mixin class to augment the functionality for sorting (be sure to put the mixin class before model.ModelAdmin):

```
from django.contrib import admin
from adminsortable2.admin import SortableAdminMixin
from models import MyModel

@admin.register(MyModel)
class MyModelAdmin(SortableAdminMixin, admin.ModelAdmin):
    pass
```

That's it! The list view of the model admin interface now adds a column with a draggable area. By clicking on that area, the user can move that row up or down. If he wants to move it to another page, he can do that as a bulk operation, using the admin actions.

By default the draggable area is positioned on the first column. If it shall be placed somewhere else, add the ordering field name explicitly to the attribute `list_display`.

Overriding change list page

To add for example a custom tool to the change list view, copy `contrib/admin/templates/admin/change_list.html` to either `templates/admin/my_app/` or `templates/admin/my_app/page/` di-

rectory of your project and make sure you are extending from the right template:

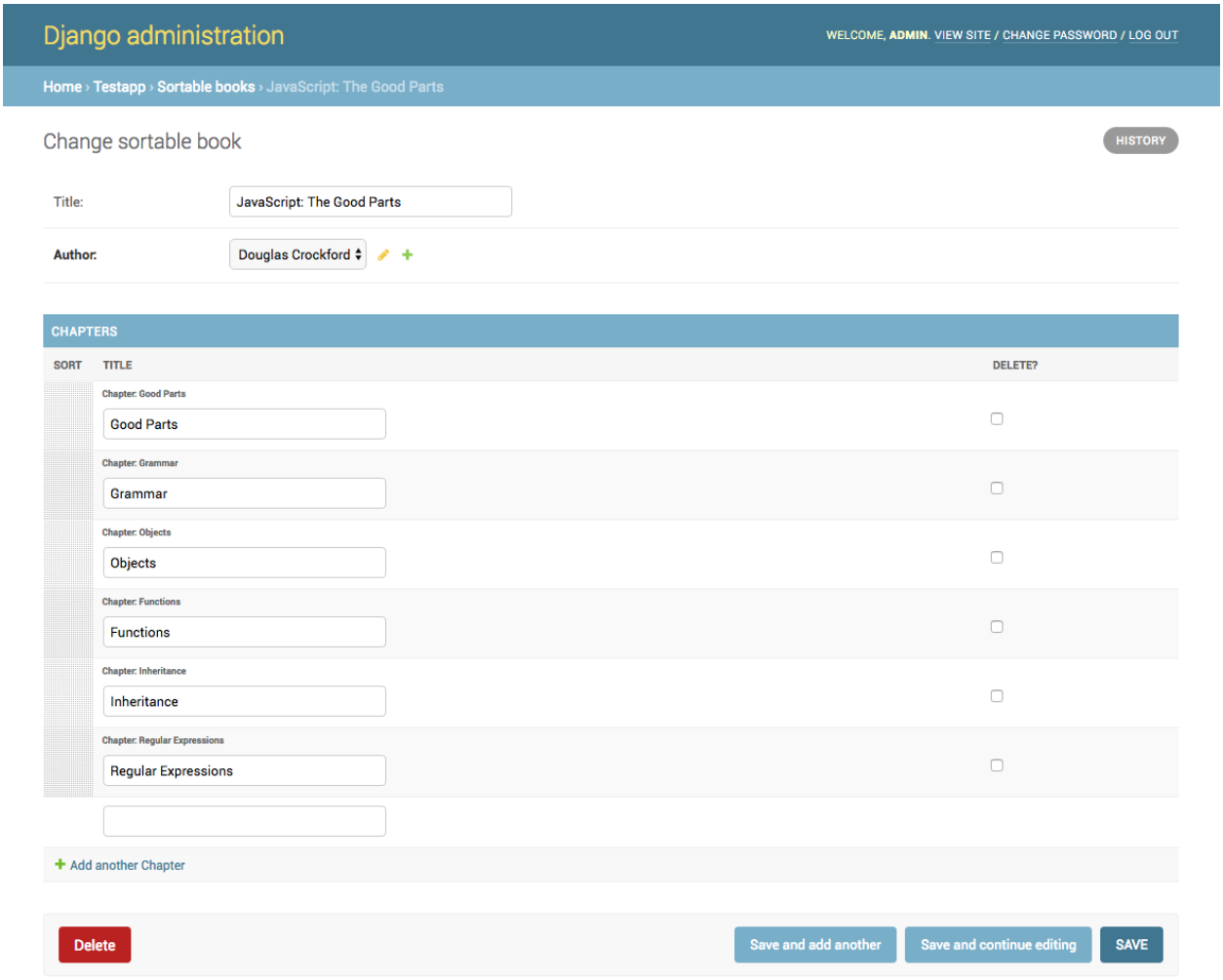
```
{% extends "adminsortable2/change_list.html" %}

{% block object-tools-items %}
    {{ block.super }}
    <li>
        <a href="mylink/">My Link</a>
    </li>
{% endblock %}
```

2.2.3 Make a stacked or tabular inline view sortable

The interface for a sortable stacked inline view looks exactly the same. If you click on an stacked inline's field title, this whole inline form can be moved up and down.

The interface for a sortable tabular inline view adds a sensitive area to each draggable row. These rows then can be moved up and down.



Sortable Tabular Inlines

After moving a tabular or stacked inline, save the model form to persist its sorting order.

Integrate into a detail view

```
from django.contrib import admin
from adminsortable2.admin import SortableInlineAdminMixin
from models import MySubModel, MyModel

class MySubModelInline(SortableInlineAdminMixin, admin.TabularInline):  # or admin.
↪StackedInline
    model = MySubModel

@admin.register(MyModel)
class MyModelAdmin(admin.ModelAdmin):
    inlines = (MySubModelInline,)
```

Note: Remember to also set the list `ordering` in the `Meta` class of `MySubModel`.

Warning: Tabular inlines with help text expect `admin/img/icon_unknown.svg` to be in the staticfiles path. Prior to Django 1.9, you'll need to manually add this icon or patch the template to remove it.

2.2.4 Sortable Many to Many Relations with Sortable Tabular Inlines

Sortable many to many relations can be achieved by creating a model to act as a juction table and adding an ordering field. This model can be specified on the `models.ManyToManyField through` parameter that tells the Django ORM to use your juction table instead of creating a default one. Otherwise, the process is conceptually similar to the above examples.

For example if you wished to have buttons added to control panel able to be sorted into order via the Django Admin interface you could do the following. A key feature of this approach is the ability for the same button to be used on more than one panel.

`models.py`

`admin.py`

2.2.5 Initial data

In case you just changed your model to contain an additional sorting field (e.g. `my_order`), which does not yet contain any values, then you **must** set initial ordering values.

django-admin-sortable2 is shipping with a management command which can be used to prepopulate the ordering field:

```
shell> ./manage.py reorder my_app.ModelOne [my_app.ModelTwo ...]
```

If you prefer to do a one-time database migration, just after having added the ordering field to the model, then create a datamigration.

..code:: python

shell> ./manage.py makemigrations myapp

this creates **non** empty migration named something like `migrations/0123_auto_20160208_054.py`.

Edit the file and change it into a data migration:

```
def reorder(apps, schema_editor):
    MyModel = apps.get_model("myapp", "MyModel")
    order = 0
    for item in MyModel.objects.all():
        order += 1
        item.my_order = order
        item.save()
```

then add to operations list, after migrations. Add `migrations.RunPython(reorder)` to the list of operations:

```
class Migration(migrations.Migration):
    operations = [
        ....
        migrations.RunPython(reorder),
    ]
```

then apply the changes to the database using:

```
shell> ./manage.py migrate myapp
```

Note: If you omit to prepopulate the ordering field with unique values, after adding this field to an existing model, then attempting to reorder field manually will fail.

2.2.6 Note on unique indices on the position field

From a design consideration, one might be tempted to add a unique index on the ordering field. But in practice this has serious drawbacks:

MySQL has a feature (or bug?) which requires to use the `ORDER BY` clause in bulk updates on unique fields.

SQLite has the same bug which is even worse, because it does neither update all the fields in one transaction, nor does it allow to use the `ORDER BY` clause in bulk updates.

Only PostgreSQL does it "right" in the sense, that it updates all fields in one transaction and afterwards rebuilds the unique index. Here one can not use the `ORDER BY` clause during updates, which from the point of view for SQL semantics, is senseless anyway.

See https://code.djangoproject.com/ticket/20708 for details.

Therefore I strongly advise against setting `unique=True` on the position field, unless you want unportable code, which only works with Postgres databases.

2.3 Run Example Code

To get a quick first impression of this plugin, clone this repositoty from GitHub and run an example webserver:

```
git clone https://github.com/jrief/django-admin-sortable2.git
cd django-admin-sortable2/example/
./manage.py migrate
./manage.py createsuperuser --username admin --email admin@example.org
./manage.py loaddata testapp/fixtures/data.json
./manage.py runserver
```

Point a browser onto http://localhost:8000/admin/, sign in as `admin` and go to **Testapp > Sortable books**. There you can test the behavior of **django-admin-sortable2**.

2.4 Release history

2.4.1 0.6.17

- Fixes #171: Adhere to Content Security Policy best practices by removing inline scripts.
- Adopted to Django-2.0 keeping downwards compatibility until Django-1.9.

2.4.2 0.6.16

- Fixes #137: Allow standard collapsible tabular inline.

2.4.3 0.6.15

- Fixes #164: TypeError when `display_list` in admin contains a callable.

Fixes #160: Updated ordering values not getting saved in Tabluar/StackedInlineAdmin.

2.4.4 0.6.14

- Fixes #162: In model admin, setting `actions` to `None` or `[]` breaks the sortable functionality.

2.4.5 0.6.13

- Fixes #159: Make stacked inline's header more clear that it is sortable.

2.4.6 0.6.12

- Fixes #155: Sortable column not the first field by default.

2.4.7 0.6.11

- Fixes #147: Use current admin site name instead of 'admin'.
- Fixes #122: Columns expand continuously with each sort.

2.4.8 0.6.9 and 0.6.10

- Fixes Issue #139: better call of post_save signal.

2.4.9 0.6.8

- Fixes Issue #135: better call of pre_save signal.
- On `./manage.py reorder ...`, name the model using `app_label.model_name` rather than requiring the fully qualified path.
- In `adminsortable2.admin.SortableAdminMixin` renamed method `update` to `update_order`, to prevent potential naming conflicts.

2.4.10 0.6.7

- Added class `PolymorphicSortableAdminMixin` so that method `get_max_order` references the ordering field from the base model.

2.4.11 0.6.6

- Fixed: Drag'n Drop reordering now send [pre|post]_save signals for all updated instances.

2.4.12 0.6.5

- Fixed: Reorder management command now accepts args.

2.4.13 0.6.4

- Drop support for Django-1.5.
- change_list_template now is extendible.
- Fixed concatenation if `exclude` is tuple.
- Support reverse sorting in CustomInlineFormSet.

2.4.14 0.6.3

- setup.py ready for Python 3.

2.4.15 0.6.2

- Fixed regression from 0.6.0: Multiple sortable inlines are now possible again.

2.4.16 0.6.1

- Removed global variables from Javascript namespace.

2.4.17 0.6.0

- Compatible with Django 1.9.
- In the list view, it now is possible to move items to any arbitrary page.

2.4.18 0.5.0

- Changed the namespace from adminsortable to adminsortable2 to allow both this

project and django-admin-sortable to co-exist in the same project. This is helpful for projects to transition from one to the other library. It also allows existing projects's migrations which previously relied on django-admin-sortable to continue to work.

2.4.19 0.3.2

- Fixed #42: Sorting does not work when ordering is descending.

2.4.20 0.3.2

- Using property method `media()` instead of hard coded `Media` class.
- Using the `verbose_name` from the column used to keep the order of fields instead of a hard coded "Sort".
- When updating order in change_list_view, use the CSRF protection token.

2.4.21 0.3.1

- Fixed issue #25: admin.TabularInline problem in django 1.5.x
- Fixed problem when adding new Inline Form Fields.
- PEP8 cleanup.

2.4.22 0.3.0

- Support for Python-3.3.
- Fixed: Add list-sortable.js on changelist only. Issue #31.

2.4.23 0.2.9

- Fixed: StackedInlines do not add an empty field after saving the model.
- Added management command to preset initial ordering.

2.4.24 0.2.8

- Refactored documentation for Read-The-Docs

2.4.25 0.2.7

- Fixed: MethodType takes only two attributes

2.4.26 0.2.6

- Fixed: Unsortable inline models become draggable when there is a sortable inline model

2.4.27 0.2.5

- Bulk actions are added only when they make sense.
- Fixed bug when clicking on table header for ordering field.

2.4.28 0.2.4

- Fix CustomInlineFormSet to allow customization. Thanks **yakky**.

2.4.29 0.2.2

- Distinction between different versions of jQuery in case django-cms is installed side by side.

2.4.30 0.2.0

- Added sortable stacked and tabular inlines.

2.4.31 0.1.2

- Fixed: All field names other than "order" are now allowed.

2.4.32 0.1.1

- Fixed compatibility issue when used together with django-cms.

2.4.33 0.1.0

- First version published on PyPI.

2.4.34 0.0.1

First working release.

2.5 Indices and tables

- genindex
- modindex
- search

CHAPTER 3

License

CHAPTER 4

Some Related projects

- https://github.com/iambrandontaylor/django-admin-sortable
- https://github.com/mtigas/django-orderable
- http://djangosnippets.org/snippets/2057/
- http://djangosnippets.org/snippets/2306/
- http://catherinetenajeros.blogspot.co.at/2013/03/sort-using-drag-and-drop.html