django-nested-admin Documentation

Release 3.2.0

The Atlantic

Apr 03, 2019

Contents

django-nested-admin is a project that makes it possible to nest admin inlines (that is, to define inlines on InlineModelAdmin classes). It also allows these inlines to be sorted by drag-and-drop, and has optional Grappelli support.

CHAPTER 1

Compatibility

django-nested-admin is compatible with Django 1.8+ and Python versions 2.7 and 3.4+. It is also compatible with Grappelli and Django Suit. It does not require Grappelli or Django Suit for the drag-and-drop sorting features.

1.1 Contents

1.1.1 Quick start guide

Django (version 1.8+) needs to be installed to use django-nested-admin.

Installation

```
pip install django-nested-admin
```

Go to GitHub if you need to download or install from source.

Setup

Open `settings.py` and add `nested_admin` to your `INSTALLED_APPS`

```
INSTALLED_APPS = (
    # ...
    'nested_admin',
)
```

Add URL-patterns:

```
urlpatterns = patterns('',
    # ...
```

(continues on next page)

(continued from previous page)

```
    url(r'^_nested_admin/', include('nested_admin.urls')),
)
```

Collect the static files:

```
$ python manage.py collectstatic
```

Example Usage

In order to use `django-nested-admin`, use the following classes in place of their django admin equivalents:

django.contrib.admin	nested_admin
ModelAdmin	NestedModelAdmin
InlineModelAdmin	NestedInlineModelAdmin
StackedInline	NestedStackedInline
TabularInline	NestedTabularInline

There is also `nested_admin.NestedGenericStackedInline` and `nested_admin.NestedGenericTabularInline` which are the nesting-capable versions of `GenericStackedInline` and `GenericTabularInline` in `django.contrib.contenttypes.admin`.

```
# An example admin.py for a Table of Contents app

from django.contrib import admin
import nested_admin

from .models import TableOfContents, TocArticle, TocSection

class TocArticleInline(nested_admin.NestedStackedInline):
    model = TocArticle
    sortable_field_name = "position"

class TocSectionInline(nested_admin.NestedStackedInline):
    model = TocSection
    sortable_field_name = "position"
    inlines = [TocArticleInline]

class TableOfContentsAdmin(nested_admin.NestedModelAdmin):
    inlines = [TocSectionInline]

admin.site.register(TableOfContents, TableOfContentsAdmin)
```

1.1.2 Why use django-nested-admin?

	django-nested-admin	django-super-inlines	django-nested-inlines	dj-nested-inlines	django-nested-inline
Nested inlines	✓	✓	✓	✓	✓
Grappelli support	✓	✓			
Generic inlines support	✓	✓			
Drag-and-drop sorting	✓				
Selenium Tests	✓				
Django Suit support	✓				

Above, you will find a feature comparison of all other implementations of nested inlines in the django admin of which I am aware. django-nested-inline, django-nested-inlines, and dj-nested-inlines are all variations upon a patch posted to Django ticket #9025. django-nested-inlines has had no commits since 2013 and appears to be abandoned, with dj-nested-inlines being a better maintained fork. It, along with django-nested-inline, are maintained insofar as the maintainers merge in pull requests and update PyPI, but they do not appear to be working on improving these implementations themselves. django-super-inlines is a fresher take on the problem, and looks to be more promising.

The largest functional difference between django-nested-admin and these other projects is its support for drag-and-drop sorting functionality within and between inlines, similar to Grappelli's sortable inline feature (though it does not require Grappelli). It also has fairly extensive test coverage. And lastly django-nested-admin is and will remain an integral part of the CMS that powers TheAtlantic.com (and CityLab.com) where it is used by editors to curate the homepages, email newsletters, and landing pages for special projects and site sections (the way we used django-nested-admin to power the curation of TheAtlantic.com's homepage was the subject of a talk we gave at the 2015 Djangocon).

1.1.3 Customization

Sortables

The steps for enabling drag-and-drop sorting functionality is identical to Grappelli's (even with the vanilla Django admin), so the Grappelli documentation for this feature is the best reference. This is equally true of sortables in Django Suit—do not follow Django Suit's instructions for sortables, as they will not work. The other features of Grappelli have not been ported to work with vanilla Django, but in principle they should all still work with nested inlines using django-nested-admin if Grappelli is installed. If you run into any difficulty with this, please create an issue on this project's Github.

Events

Adding or removing an inline fires the `formset:added` and `formset:removed` events, respectively, that were added to Django in 1.9 (#15760). This is particularly useful for when you need to execute javascript upon an inline being added (for instance, to initialize a custom widget). The API is the same as Django 1.9, so reference the official Django docs for information about how to use these events. In addition, django-nested-admin also emits events for when the delete checkbox (or button, in Grappelli) is toggled with the `formset:deleted` and `formset:undeleted` events. Less likely to be used, but still available, are the following events:

`djnesting:mutate` Fired whenever the state of a formset changes

`djnesting:initialized` Fired after an inline has been initialized, either on load or after being added

djnesting::attrchanged Fired whenever a drag-and-drop operation or removal necessitates changing the `name`, `id`, and `for` attributes of the elements within a form or formset. Ideally custom Django widgets with javascript hooks should be written so that they do not need to be reinitialized if a form field's `id` or `name` changes, but in cases where they do, use this event to refresh or reinitialize the bindings.

1.1.4 Integrations

django-polymorphic

Quickstart

Support for django-polymorphic is currently in beta. To use polymorphic inlines with django-nested-admin follow the steps for setting up django-polymorphic admin inline integration. The instructions are identical with django-nested-admin, except that the classes you would normally import from `polymorphic.admin` should instead be imported from `nested_admin` and the classes themselves should be prefixed with `Nested` For example:

```
# Instead of these imports
from django.contrib.admin import ModelAdmin
from polymorphic.admin import (
    PolymorphicInlineSupportMixin, StackedPolymorphicInline)

# One should import:
from nested_admin import (
    NestedModelAdmin,
    NestedPolymorphicInlineSupportMixin, NestedStackedPolymorphicInline)
```

The polymorphic inlines used with django-nested-admin can have other inlines nested inside them, or even other polymorphic inlines. The only requirement is that all ModelAdmin and InlineAdmin classes inherit from the appropriate nested version.

Example

```
from django.contrib import admin
import nested_admin
from .models import (
    Store, Customer, GuestCustomer, LoggedInCustomer, OrderItem, Product,␣
↪WishListItem,
    Order, WishList, Payment, CreditCardPayment, BankPayment)

class PaymentInline(nested_admin.NestedStackedPolymorphicInline):
    model = Payment
    class CreditCardPaymentInline(nested_admin.NestedStackedPolymorphicInline.Child):
        model = CreditCardPayment
    class BankPayment(nested_admin.NestedStackedPolymorphicInline.Child):
        model = BankPayment
    child_inlines = (CreditCardPaymentInline, BankPayment)

class ProductInline(nested_admin.NestedStackedInline):
    model = Product

class WishListItemInline(nested_admin.NestedStackedInline):
    model = WishListItem
    sortable_field_name = 'position'
```

(continues on next page)

(continued from previous page)

```
class WishListInline(nested_admin.NestedStackedInline):
    model = WishList
    inlines = [WishListItemInline]

class OrderItemInline(nested_admin.NestedStackedInline):
    model = OrderItem

class OrderInline(nested_admin.NestedTabularInline):
    model = Order
    inlines = [OrderItemInline, PaymentInline]

class CustomerInline(nested_admin.NestedStackedPolymorphicInline):
    model = Customer
    inlines = [OrderInline]
    class GuestCustomerInline(nested_admin.NestedStackedPolymorphicInline.Child):
        model = GuestCustomer
    class LoggedInCustomerInline(nested_admin.NestedStackedPolymorphicInline.Child):
        model = LoggedInCustomer
        inlines = [WishListInline]
    child_inlines = (GuestCustomerInline, LoggedInCustomerInline)

@admin.register(Store)
class StoreAdmin(nested_admin.NestedPolymorphicModelAdmin):
    inlines = [CustomerInline]
```

1.1.5 Contributing

Editing javascript and css

This project uses webpack for building its javascript and css. To install the dependencies for the build process, run `npm install` from the root of the repository. You can then run `npm run build` to rebuild the static files.

Running tests

django-nested-admin has fairly extensive test coverage. The best way to run the tests is with tox, which runs the tests against all supported Django installs. To run the tests within a virtualenv run `python runtests.py` from the repository directory. The tests require a selenium webdriver to be installed. By default the tests run with phantomjs, but it is also possible to run the tests with the chrome webdriver by passing `--selenium=chrome` to runtests.py or, if running with tox, running `tox -- --selenium=chrome`. See `runtests.py --help` for a complete list of the options available.

Pull requests are automatically run through Travis CI upon submission to verify that the changes do not introduce regressions.

Writing tests

The tests for this project are patterned off of the system that the Django project itself uses for tests (including the changes that are being made in 1.10), so Django's guide on writing unit tests is a good place to start.

Selenium tests are difficult to write, particularly for complicated user interactions such as drag-and-drop. To assist in writing tests for the functionality of nested inlines, a base TestCase with methods for executing the possible user

interactions with inlines (e.g. adding inlines, removing inlines, setting field values, drag-and-drop re-ordering) is provided with `base.BaseNestedAdminTestCase`.

Depending on what you are trying to test, it might make sense to add the test to one of the existing "test apps" in django-nested-admin. These include:

admin_widgets Integration tests for built-in Django widgets that use javascript, e.g. `prepopulated_fields` or ManyToManyFields with `filter_horizontal`.

gfk Tests for `nested_admin.NestedGenericStackedInline` and `nested_admin.NestedGenericTabularInline`

one_deep Tests that compare screenshots of standard, non-nested inlines using the standard inline classes and their nested equivalents (but without any actual nesting), to ensure that the custom templates, css, and javascript used by django-nested-admin looks the same as their non-nested Django or Grappelli equivalents.

two_deep Tests for `nested_admin.NestedStackedInline` and `nested_admin.NestedTabularInline` where there is only one "nested" inline (using two levels of inlines, hence "two deep").

three_deep Tests the same things as `two_deep`, except with inlines nested "three deep."

If your test requires the creation of new test models, then it may make sense to write a new test app. Create a new subdirectory under `nested_admin/tests` with an `__init__.py`, `admin.py`, `models.py`, and `tests.py`. The test runner will automatically add this to the list of `INSTALLED_APPS` and execute the tests defined in `tests.py`. The tests in `two_deep` are the most complete, and can serve as a reference for how to use the helper methods for simulating user interactions.

1.1.6 Changelog

3.2.0 (Apr 3, 2019)

- Feature: Added beta support for django-polymorphic admin (#86)
- Feature: Made compatible with Django 2.2 and 3.0. Django 3.0 is still in alpha, so the django-nested-admin compatibility is likewise not yet stable
- Fixed: django-nested-admin now respects permissions for inline model admins in Django 2.1+, including the new 'view' permission.
- Fixed: (grappelli) Collapsing inline groups now works for stacked inlines (thanks @maldn) (#121)
- Fixed: FileFields in deeply nested inlines now work in Django 2.1+ (thanks @btknu) (#111, #127)
- Fixed: Use correct translation for 'Delete?' text in templates (thanks @kigawas) (#116)

3.1.3 (Dec 15, 2018)

- Fixed: Use `jQuery.fn.length`, not `.size`, for compatibility with jQuery 3, the version bundled with Django 2.1+ (#109)

3.1.2 (Sep 6, 2018)

- Fixed: Bug with grappelli that prevented deeply nested datepicker and timepicker widgets from working.

3.1.0 (Aug 13, 2018)

- Fixed: NestedTabularInline support in Django 2.0 (#97)
- Fixed: Ensure correct relative order of js media. (#71)
- Switch js build process to use webpack, without gulp
- Add test coverage reporting for both python and js code

3.0.21 (Nov 1, 2017)

- Fixed: Bug when saving child models that use django-polymorphic
- Feature: Made compatible with django-autocomplete-light (#84)

3.0.20 (Aug 2, 2017)

- Fixed: Correctly show inline label number in django admin 1.9+ (#79)

3.0.16 (Mar 10, 2017)

- Support Django 2.0

3.0.15 (Feb 27, 2017)

- Fixed: bug caused when `TEMPLATE['OPTIONS']['string_if_invalid']` is set (#70)

3.0.13 (Feb 13, 2017)

- Fixed: grappelli autocomplete widget support (#57)
- Improvement: enforce admin `min_num` setting in javascript

3.0.11 (Oct 18, 2016)

- Fixed: bug when multiple inlines share the same prefix (#60)

3.0.10 (Sep 13, 2016)

- Fixed: bug if `django.contrib.admin` precedes `nested_admin` in `INSTALLED_APPS` (#56)
- Fixed: don't show add inline link when `max_num = 0` (#54)
- Improvement: Added `'djnesting:beforeadded` javascript event to ease integration with third-party admin widgets. (#47)
- Feature: support Django 1.10 inline classes (for collapsing) (#32, #52)

3.0.8 (Jun 13, 2016)

- Fixed: `max_num` off-by-one error (#44)
- Fixed: saving with a blank intermediate inline now works (#46)

3.0.5 (Jun 7, 2016)

- Fixed: ForeignKey widget on added inline (#45)

3.0.4 (June 3, 2016)

- Fixed: Support `prepopulated_fields` in grappelli (#43)

3.0.3 (May 26, 2016)

- Fixed: Bug with grappelli ForeignKey related lookup widget (thanks @sbussetti)

3.0.2 (April 17, 2016)

- Feature: django-suit support
- Fixed: javascript syntax error
- Fixed: bug where tabular inlines without sortables could not be updated

3.0.0 (April 13, 2016)

- Added documentation
- Fixed visual discrepancies between the appearance of nested and their usual appearance in Django and Grappelli. Added screenshot comparison tests to prevent future regressions.

- Support nesting of generic inlines (fixes #30)
- Support for `show_change_link` (fixes #22)
- Support for Django 1.10dev
- Dropped support for version of Django prior to 1.8, which greatly simplified the Python code.
- Use gulp for building static assets, rewritten with scss and ES6

1.1.7 API reference

`nested_admin` Package

`nested_admin.tests` Module

1.2 License

1.3 Indices and tables

- genindex
- modindex
- search

Index

N