Django-Admin-Object-Actions Documentation

Release 0.1.8

Nine More Minutes, Inc.

Aug 17, 2020

CONTENTS

1 Installation 3

2 Usage 5

Django-Admin-Object-Actions provides object-level actions in the Django admin interface.

In contrast to the built-in admin actions, which are only applied to a list of objects from the change list, object actions apply to a single object and can be accessed from either the change list view or the change form view.

Each action is defined declaratively on the `ModelAdmin` instance and may provide a form for additional input or validation before executing the action.

Inspired by This Article.

It is tested against:

- Django 1.11 (Python 3.5 and 3.6)
- Django 2.0 (Python 3.5, 3.6 and 3.7)
- Django 2.1 (Python 3.5, 3.6 and 3.7)
- Django 2.2 (Python 3.5, 3.6, 3.7 and 3.8)
- Django 3.0 (Python 3.6, 3.7 and 3.8)
- Django 3.1 (Python 3.6, 3.7 and 3.8)
- Django master/3.2 (Python 3.6, 3.7 and 3.8)

CHAPTER

ONE

INSTALLATION

Install the application from PYPI:

```
pip install django-admin-object-actions
```

Add `admin_object_actions` to your `INSTALLED_APPS`:

```
INSTALLED_APPS += ('admin_object_actions',)
```

Django-Admin-Object-Actions requires `django-crum` and will install it as a dependency. To enable it, add `CurrentRequestUserMiddleware` to your `MIDDLEWARE` setting:

```
MIDDLEWARE += ('crum.CurrentRequestUserMiddleware',)
```

CHAPTER

TWO

USAGE

Ensure your `ModelAdmin` class inherits from the `ModelAdminObjectActionsMixin`. It should also include `display_object_actions_list` in the `list_display` attribute and `display_object_actions_detail` in the `fields` and `readonly_fields` attributes to ensure the object action buttons show up in the change list and change views, respectively:

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

@admin.register(MyModel)
class MyModelAdmin(ModelAdminObjectActionsMixin, admin.ModelAdmin):

    list_display = (
        ...,  # your existing list display fields
        'display_object_actions_list',
    )

    fields = (
        ...,  # your existing detail display fields
        'display_object_actions_detail',
    )

    readonly_fields = (
        ...,  # your existing read-only fields
        'display_object_actions_detail',
    )
```

If using `fieldsets` instead of `fields`, add `display_object_actions_detail` to the desired fieldset as you would with any other field.

Create custom subclasses of `AdminObjectActionForm` for actions requiring additional input or confirmation and implement the `do_object_action` method:

```
# myapp/forms.py
from django import forms
from admin_object_actions.forms import AdminObjectActionForm
from .models import MyModel

class MyActionForm(AdminObjectActionForm):

    confirm = forms.BooleanField()

    class Meta:
```

(continues on next page)

(continued from previous page)

```
        model = MyModel
        fields = ()

    def do_object_action(self):
        self.instance.action()
```

Define an `object_actions` dictionary on your `ModelAdmin` class similar to the following:

```
from .forms import MyActionForm

class MyModelAdmin(ModelAdminObjectActionsMixin, admin.ModelAdmin):

    object_actions = [
        {
            'slug': 'myaction',
            'verbose_name': _('my action'),
            'verbose_name_past': _('acted upon'),
            'form_class': MyActionForm,
            'fields': ('id', 'confirm'),
            'readonly_fields': ('id',),
            'permission': 'change',
        },
        {
            'slug': 'myotheraction',
            'verbose_name': _('my other action'),
            'verbose_name_past': _('acted upon'),
            'form_method': 'GET',
            'function': 'do_other_action',
            'permission': 'otheraction',
        },
    ]

    def has_otheraction_permission(self, request, obj=None):
        return True

    def do_other_action(self, obj, form):
        obj.other_action(form.cleaned_data)
```

Each object action listed on the `ModelAdmin` must define a `slug` field and may define additional fields customize the action's behavior:

`slug` The internal name of this action; will be used to create the custom URL used by the action.

`verbose_name` The translatable name of this action displayed on the action buttons in the admin interface. Defaults to `slug.title()`.

`verbose_name_title` The translatable name shown on the object action form page. Defaults to `verbose_name`.

`verbose_name_past` The translatable past tense version of the action displayed to users in messages and admin log entries.

`form_class` The form class used to implement validation/confirmation of this action, should be a subclass of `admin_object_actions.forms.AdminObjectActionForm`.

`function` Function called to execute the given object action. The default is to call `do_object_action` on the given `form_class`. This option may be a string, in which case the method with the same name from the `ModelAdmin` class or the `Model` class will be used.

readonly_field List of readonly fields to display in the custom admin form.

fields List of fields to display in the custom action form.

fieldsets Custom fieldsets to display for the object action form. Defaults to a single fieldset with `fields`.

permission Custom permission required to display or execute this object action. Default is `change`. If defined, a `has_<permission>_permission` method on the `ModelAdmin` class will be called to check whether the action is allowed.

form_template Custom form template used to render the object action form. Default is `admin/object_action_form.html`.

list_only If `True`, this object action will only be shown in the changelist view.

detail_only If `True`, this object action will only be shown in the change form view.

view Overrides the default view function called for this action. The default is the `object_action_view` method defined on the mixin class.

Additional methods of the `ModelAdminObjectActionsMixin` class may be overridden to further customize the behavior of object actions.

To customize the column header in the change list view or the field label in the change view, override the `display_object_actions_list` or `display_object_actions_detail` methods as shown below:

```
class MyModelAdmin(ModelAdminObjectActionsMixin, admin.ModelAdmin):

    def display_object_actions_list(self, obj=None):
        return self.display_object_actions(obj, list_only=True)
    display_object_actions_list.short_description = _('My Actions')

    def display_object_actions_detail(self, obj=None):
        return self.display_object_actions(obj, detail_only=True)
    display_object_actions_detail.short_description = _('My Actions')
```

See `test_project/test_app/admin.py` in the project repository for additional usage examples.