django-monitor Documentation

Release 0.2 rc

Rajeesh Nair

December 20, 2015

Contents

1 Introduction to Django-monitor **3**

1.1 About 3

1.2 Installation 3

1.3 Features 4

2 How to use (for developers) **5**

2.1 Registration (Enqueue) 5

2.2 Special model-admin class 6

2.3 Related moderation 6

2.4 Data-protection 6

2.5 Creation of objects by code 7

2.6 Post-moderation hook 7

3 How to use (for admin-users) **9**

3.1 Who can moderate 9

3.2 What to moderate & from where 9

3.3 How to moderate 11

4 Indices and tables **13**

Contents:

CHAPTER 1

Introduction to Django-monitor

Note: You can skip this chapter if you have already read the README document in source path.

1.1 About

Moderation seems to be some job each one want to do in their own way. Django-monitor is a django-app to moderate model objects. It was started as a clone of the *django-gatekeeper* project but to meet a different set of business requirements.

The terms, ‘‘monitor‘‘ and ‘‘moderate‘‘ are used with same meaning everywhere in the source.

Here, the moderation process is well integrated with django-admin. That is, all moderation actvities are performed from within the changelist page itself.

1.2 Installation

1. Directly from the python package index
   (a) Using pip:

```
$ pip install django_monitor
```

   (b) Using easy_install:

```
$ easy_install django_monitor
```

2. OR Directly from the mercurial repo
   (a) Clone the repo (if you have hg installed):

```
$ hg clone http://bitbucket.org/rajeesh/django-monitor/
```

3. OR Download & install from available archives
   - Get the archive from any of the following locations:
     – http://bitbucket.org/rajeesh/django-monitor/downloads
     – http://pypi.python.org/pypi/django-monitor#downloads
   - Install using the setuptools as given below:

```
$ tar xzf django-monitor-xxx.tar.gz
$ cd django-monitor-xxx
$ python setup.py install
```

- If setuptools is not installed, you may copy the `django_monitor` directory to somewhere in your python path.

4. Then add 'django_monitor' to your project's `settings.INSTALLED_APPS`.

1.3 Features

1.3.1 Model-specific permission

Each moderated model will have an associated moderate permission. To approve or challenge any object created for a particular model, users need to have the corresponding permission.

1.3.2 Auto-moderation

Any object created by a user with add permission will have an `In Pending` status. If the user has got moderate permission also, the object created will automatically get approved (status becomes `Approved`).

1.3.3 Moderation from within admin changelist

The changelist of a moderated model displays the current moderation status of all objects. Also, you can filter the objects by their moderation status. Three actions are available for moderation. To moderate, user just need to select the objects, choose appropriate action and press `Go`.

1.3.4 Related moderation

When a manager moderates some model objects, there may be some other related model objects which also can get moderated along with the original ones. The developer can specify such related models to be moderated during registration.

1.3.5 Data protection

The developer can prevent admin-users from changing values of selected fields of approved objects. Deleting approved objects also can be prevented if your client's business requires that.

CHAPTER 2

How to use (for developers)

2.1 Registration (Enqueue)

Register the model for moderation using `django_monitor.nq`.

Example

```
import django_monitor
# Your model here
django_monitor.nq(YOUR_MODEL)
```

The full signature is...

```
django_monitor.nq(
    model, [rel_fields = [], can_delete_approved = True,
    manager_name = 'objects', status_name = 'status',
    monitor_name = 'monitor_entry', base_manager = None]
)
```

`model` is the only required argument. Other optional arguments follow:

- `rel_fields`: List of related fields to be moderated along with this. Read more details below at *Related moderation*.
- `can_delete_approved`: To prevent admin-users from deleting approved objects, set this to `False`. Default is `True`. Read more details below at *Data-protection*.
- `manager_name`: We assume that `objects` is the name of the manager instance of your model. If you want to use a different name for the instance, specify the name with `manager_name` parameter.
- `status_name`: By default, the moderation status field is named as *status*. If you prefer some other name, specify it.
- `monitor_name`: A MonitorEntry object will be created to monitor each object of moderated model. By default, it is referred as *monitor_entry*. If you prefer some other name, specify it.
- `base_manager`: Django-monitor replaces the manager of moderated model with a special manager class derived from the original. Leave this as None if you want to use the default manager class. If you have written a custom manager for the model, you may specify it here.

2.2 Special model-admin class

We build admin classes for moderated models using MonitorAdmin instead of django's built-in ModelAdmin. Always remember to inherit from MonitorAdmin when you define model-admin class for your moderated model.

```
# in your admin.py
from django_monitor.admin import MonitorAdmin
class YourModelAdmin(MonitorAdmin):
    pass
```

2.3 Related moderation

This is useful when your model is foreign key to some other model and those model objects are added inline to the former. So when you check and approve the former model object, you might have verified all those inline objects too and so they too can be approved along with it. See the **example**:

```
# In admin.py
class BookAdmin(MonitorAdmin):
    inlines = ['SupplementInline',]

# In models.py
class Book(model.Model):
    name = models.CharField(max_length = 100)

class Supplement(models.Model):
    name = models.CharField(max_length = 100)
    book = models.ForeignKey(Book, related_name = 'supplements')

django_monitor.nq(Book, rel_fields = ['supplements'])
django_monitor.nq(Supplement)
```

Remember that both models should be put in moderation queue.

2.4 Data-protection

Business organizations may require their applications to prevent admin users from modifying or deleting approved objects. We allow developers to enable that using two parameters, protected_fields and can_delete_approved.

MonitorAdmin.protected_fields can be used to prevent users from changing values of certain fields in approved objects. Specify the field names as you would do with readonly_fields. See the **example** below:

```
# in your admin.py
class YourModelAdmin(MonitorAdmin):
    protected_fields = ['field1', 'field2']
```

can_delete_approved is an optional parameter you pass to django_monitor.nq. Its default value is True which allows users to delete all objects. If this is set to False, admin-user can not delete an object once it is approved. Deleting either un-moderated or pending or challenged objects can be done as usual. You still can delete approved objects by code or from the django-shell.

2.5 Creation of objects by code

The above sections shared tips on how to prepare your application for moderation by admin-users. What about the objects you create by code? All objects created by code will be in pending status by default. You can moderate them by code using the following public methods of the moderated model:

Note: `user` is an optional parameter in all those methods described below. Please pass the current user to the methods in all possible cases. `request.user` can be used for this whenever `request` is available. Otherwise, use the function, `django_monitor.middleware.get_current_user`.

1. **approve:**

```
        approve([user = None, notes = ''])
```

2. **challenge:**

```
        challenge([user = None, notes = ''])
```

3. **reset_to_pending:**

```
        reset_to_pending([user = None, notes = ''])
```

4. **moderate (to use when status is available during runtime only):**

```
        moderate(status, [user = None, notes = ''])
```

An example usage

```
>>> my_inst = MyModel.objects.create(arg1 = 1)
>>> my_inst.approve()
```

In addition, there are 3 public boolean properties also to let you know which moderation status a particular object is in.

1. `is_approved`
2. `is_challenged`
3. `is_pending`

An example usage

```
>>> my_inst = MyModel.objects.create()
>>> # Will be in pending status by default.
>>> my_inst.is_approved
... False
>>> my_inst.is_pending
... True
>>> my_inst.approve()
>>> my_inst.is_approved
... True
```

2.6 Post-moderation hook

If you want to perform something after an object is moderated, you can make use of the `post_moderation` signal as in the below **example**:

```
from django_monitor import post_moderation

# handler_func: function to handle your post moderation activities.
def handler_func(sender, instance, **kwargs):
    # sender: MyModel
    # instance: my_model instance that was just moderated
    pass

# MyModel: The model whose moderation you are watching.
class MyModel(models.Model):
    pass

post_moderation.connect(handler_func, sender = MyModel)
```

Note that the moderated object will be passed as the `instance` and its model as the `sender`. This will help you to write separate handlers for each model.

CHAPTER 3

How to use (for admin-users)

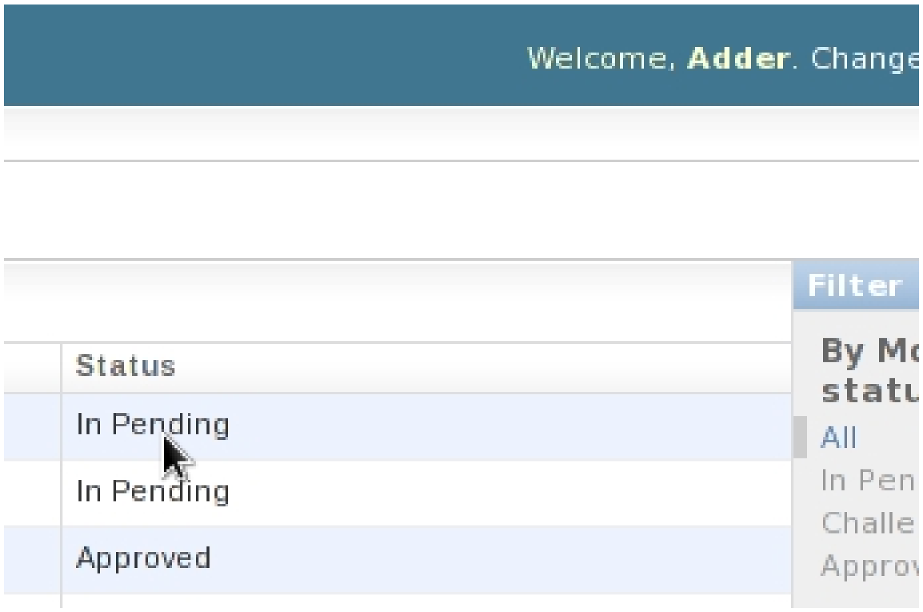
All moderation activities can be performed from within the admin interface itself. Following sections describe the process in detail. The figures are based on an example app, `TestApp` with models like `Author` and `Book`.

3.1 Who can moderate

Django-monitor creates a moderate permission for each moderated model. To moderate any object of a model, user need to have permission for that particular model. The superuser must assign those permissions to the appropriate users as they would do with other permissions.

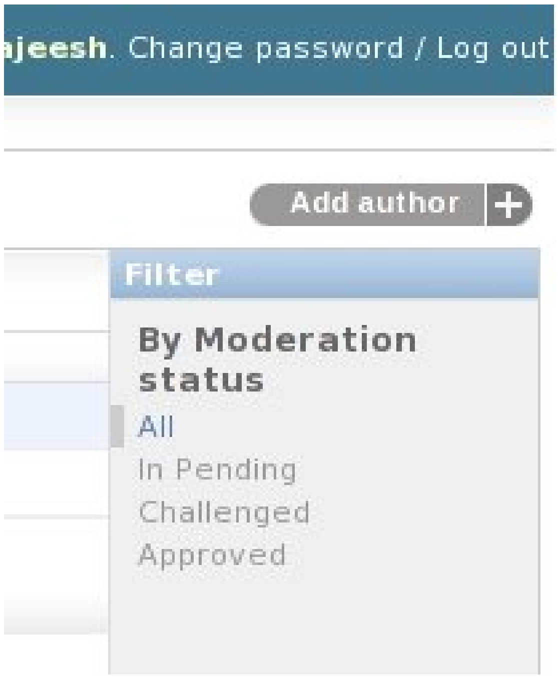
3.2 What to moderate & from where

To see and moderate the pending/challenged objects of a particualr model, visit the change-list page of that model. By default, all existing objects appear there. For moderated models, we add one more column, `status` to the right of each row. That column, as its name indicates, displays the current moderation status of each object you see in the list. This helps you to identify the pending as well as challenged objects. See the figure:

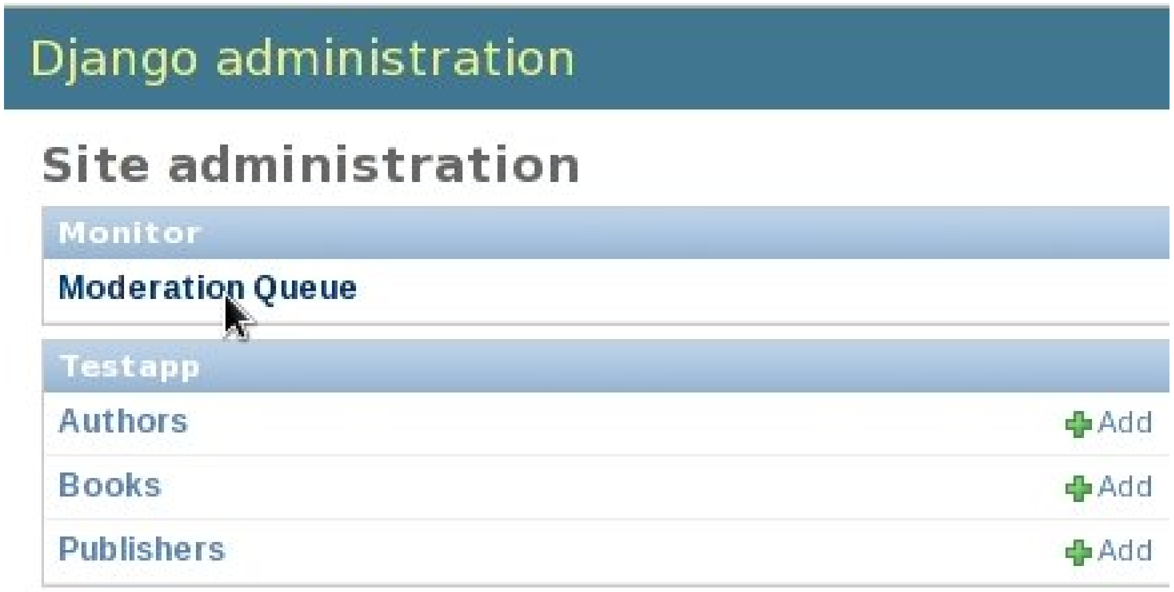


Status
In Pending
In Pending
Approved

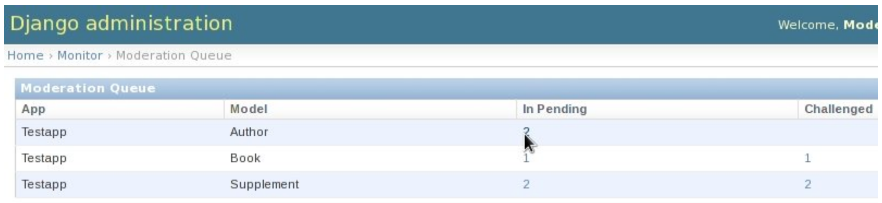
Also, you can filter the objects by `moderation status` using the options provided in the box to the right of change-list. Refer to the figure below:



You need not regularly visit change-lists of all models to know whether there are any objects to be moderated. `Moderation Queue` is the shortcut for this. It will summarize the moderation status for all models in one page. In your admin home page there is a `Moderation Queue` change-link under `Monitor` app. See the figure:



Clicking on it will lead you to a moderation queue page from where you can see a nice table listing out the number of pending and challenged objects for each moderated model. An example figure is given below:



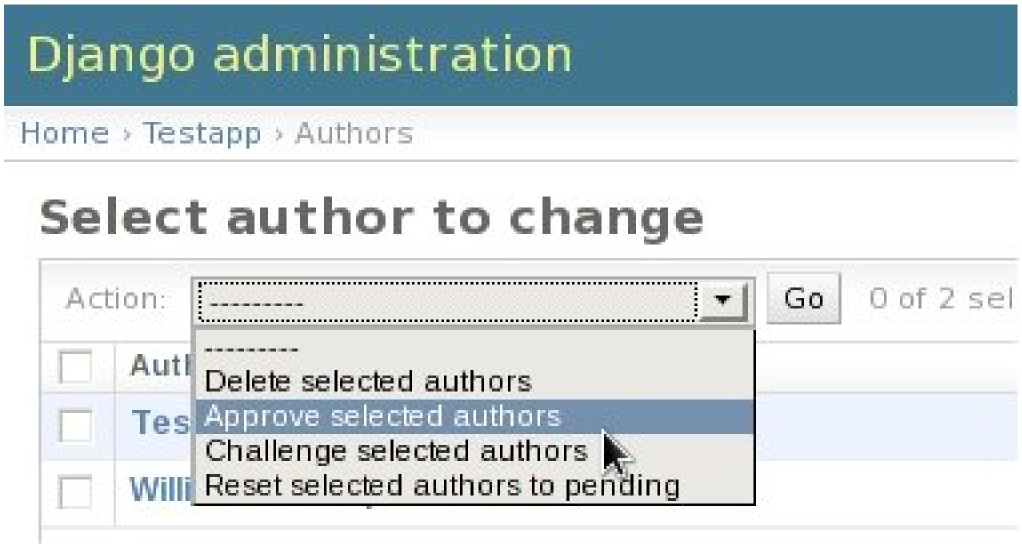
Moderation Queue

App	Model	In Pending	Challenged
Testapp	Author	2	
Testapp	Book	1	1
Testapp	Supplement	2	2

Note that the pointer points to the number *2* in the figure. This number indicates that there are 2 pending `Author` objects now. Click on that number and you will be lead to the `Author` change-list where you can see both of the pending objects. Similarly, you can find pending/challenged objects of any model from here.

3.3 How to moderate

Moderation is performed through 3 special change-list actions. They are, `Approve selected`, `Challenge selected` and `Reset selected to pending`. The figure below shows the actions found in `Author` change-list:



If the manager selects few objects, chooses the action `Approve selected` and presses `Go`, those objects will get approved. Similarly, one can challenge objects too. Once some objects get challenged, the non-managers may check them again and make required corrections. After that, they can reset the status to `In pending` using the action, `Reset selected to pending` so that their manager gets to verify the entries again.

Indices and tables

- genindex
- modindex
- search