django-moderation Documentation

Release 0.3.6

Dominik Szopa

Jul 17, 2017

Contents

CHAPTER 1

Introduction

django-moderation is reusable application for Django framework, that allows to moderate any model objects.

Possible use cases:

- User creates his profile, profile is not visible on site. It will be visible on site when moderator approves it.
- User change his profile, old profile data is visible on site. New data will be visible on site when moderator approves it.

Features:

- configurable admin integration(data changed in admin can be visible on site when moderator approves it)
- moderation queue in admin
- html differences of changes between versions of objects
- configurable email notifications
- custom model form that allows to edit changed data of object
- auto approve/reject for selected user groups or user types
- support for ImageField model fields on moderate object page
- 100% PEP8 correct code
- test coverage > 80%

CHAPTER 2

Known issues

- m2m relations in models are not currently supported

CHAPTER 3

Contributors

Special thanks to all persons that contributed to this project.

- jonwd7 http://github.com/jonwd7
- treyhunner http://github.com/treyhunner

Thank you for all ideas, bug fixes, patches.

CHAPTER 4

Screenshots

CHAPTER 5

Requirements

python 2.6+, 3.3+

django >= 1.7

CHAPTER 6

Contents:

Getting started quick guide

Installation

Use easy_install:

```
$ easy_install django-moderation
```

Or download source code from http://github.com/dominno/django-moderation and run installation script:

```
$ python setup.py install
```

Updating existing projects to Django 1.7

If you are updating an existing project which uses `django-moderation` to Django 1.7 you need to follow these simple steps:

1. Remove `'south'` from your `INSTALLED_APPS` if present.
2. Run `python manage.py migrate`.

That's it!

Configuration

`django-moderation` will autodiscover moderation classes in `<app>/moderator.py` files by default. So the simplest moderation configuration is to simply add `moderation` (or `moderation.apps.ModerationConfig`) to `INSTALLED_APPS` in your `settings.py`:

```
INSTALLED_APPS = [
    # ...
    'moderation',  # or 'moderation.apps.ModerationConfig',
    # ...
]
```

Then add all of your moderation classes to a `moderator.py` file in an app and register them with moderation:

```
from moderation import moderation
from moderation.db import ModeratedModel

from yourapp.models import YourModel, AnotherModel


class AnotherModelModerator(ModelModerator):
    # Add your moderator settings for AnotherModel here


moderation.register(YourModel)  # Uses default moderation settings
moderation.register(AnotherModel, AnotherModelModerator)  # Uses custom moderation
↪settings
```

This is exactly how Django's contributed admin app registers models.

Alternative Configuration

If you don't want `django-moderation` to autodiscover your moderation classes, you will add `moderation.apps.SimpleModerationConfig` to `INSTALLED_APPS` in your `settings.py`:

```
INSTALLED_APPS = [
    # ...
    'moderation.apps.SimpleModerationConfig',
    # ...
]
```

Then you will need to subclass your models from `moderation.db.ModeratedModel` and add moderation classes to each moderated model in `models.py`:

```
from django.db import models
from moderation.db import ModeratedModel


class MyModel(ModeratedModel):
    my_field = models.TextField()

    class Moderator:
        notify_user = False
```

Admin integration

1. If you want to enable integration with Django Admin, then register admin class with your model:

```
from django.contrib import admin
from moderation.admin import ModerationAdmin
```

```
class YourModelAdmin(ModerationAdmin):
    """Admin settings go here."""

admin.site.register(YourModel, YourModelAdmin)
```

If `admin_integration_enabled` is enabled then when saving object in admin, data will not be saved in model instance but it will be stored in moderation queue. Also data in the change form will not display data from the original model instance but data from the ModeratedObject instance instead.

How django-moderation works

When you change existing object or create new one, it will not be publicly available until moderator approves it. It will be stored in ModeratedObject model.:

```
your_model = YourModel(description='test')
your_model.save()

YourModel.objects.get(pk=your_model.pk)
Traceback (most recent call last):
DoesNotExist: YourModel matching query does not exist.
```

When you will approve object, then it will be publicly available.:

```
your_model.moderated_object.approve(by=user, reason='Reason for approve')

YourModel.objects.get(pk=1)
<YourModel: YourModel object>
```

Please note that you can also access objects that are not approved by using unmoderated_objects manager, this manager will bypass the moderation system

```
YourModel.unmoderated_objects.get(pk=your_model.pk)
```

You can access changed object by calling changed_object on moderated_object:

```
your_model.moderated_object.changed_object
<YourModel: YourModel object>
```

This is deserialized version of object that was changed.

Now when you will change an object, old version of it will be available publicly, new version will be saved in moderated_object:

```
your_model.description = 'New description'
your_model.save()

your_model = YourModel.objects.get(pk=1)
your_model.__dict__
{'id': 1, 'description': 'test'}

your_model.moderated_object.changed_object.__dict__
{'id': 1, 'description': 'New description'}

your_model.moderated_object.approve(by=user, reason='Reason for approve')
```

```
your_model = YourModel.objects.get(pk=1)
your_model.__dict__
{'id': 1, 'description': 'New description'}
```

Upgrading From Previous Versions of Django ModerationAdmin

Upgrading from previous versions of django-moderation will require converting from South migrations to Django 1.7+ migrations.

To do so, you will need to perform the following steps (skip any you have already done):

1. Configure South to use the *migrations-pre17* directory for django-moderation migrations:

```
SOUTH_MIGRATION_MODULES = {
    'moderation': 'moderation.migrations-pre17',
}
```

2. Use South to migrate up to `0002` in the `migrations-pre17` directory:

```
python manage.py syncdb moderation 0001  # Skip this if already applied
python manage.py syncdb moderation 0002  # Skip this if already applied
```

3. Fake the first two Django migrations:

```
python manage.py migrate moderation 0001 --fake
python manage.py migrate moderation 0002 --fake
```

4. Use Django to migrate `0003`:

```
python manage.py migrate moderation 0003
```

5. Finally, remove the settings for South:

```
SOUTH_MIGRATION_MODULES = {
    # 'moderation': 'moderation.migrations-pre17',
}
```

Advanced options

Moderation registration options

`moderation.register` takes following parameters:

`model_class` Model class that will be registered with moderation

`moderator_class` Class that subclasses GenericModerator class. It Encapsulates moderation options for a given model. Example:

```
from moderation.moderator import GenericModerator


class UserProfileModerator(GenericModerator):
    notify_user = False
    auto_approve_for_superusers = True
```

```
moderation.register(UserProfile, UserProfileModerator)
```

GenericModerator options

visible_until_rejected By default moderation stores objects pending moderation in the `changed_object` field in the object's corresponding `ModeratedObject` instance. If `visible_until_rejected` is set to True, objects pending moderation will be stored in their original model as usual and the most recently approved version of the object will be stored in `changed_object`. Default: False

manager_names List of manager names on which moderation manager will be enabled. Default: ['objects']

moderation_manager_class Default manager class that will enabled on model class managers passed in `manager_names`. This class takes care of filtering out any objects that are not approved yet. Default: ModerationObjectsManager

visibility_column If you want a performance boost, define visibility field on your model and add option `visibility_column = 'your_field'` on moderator class. Field must by a BooleanField. The manager that decides which model objects should be excluded when it were rejected, will first use this option to properly display (or hide) objects that are registered with moderation. Use this option if you can define visibility column in your model and want to boost performance. This method benefits those who can add fields to their models. Default: None.

fields_exclude Fields to exclude from object change list. Default: []

resolve_foreignkeys Display related object's string representation instead of their primary key. Default: True

auto_approve_for_superusers Auto approve objects changed by superusers. Default: True

auto_approve_for_staff Auto approve objects changed by user that are staff. Default: True

auto_approve_for_groups List of user group names that will be auto approved. Default: None

auto_reject_for_anonymous Auto reject objects changed by users that are anonymous. Default: True

auto_reject_for_groups List of user group names that will be auto rejected. Default: None

bypass_moderation_after_approval When set to True, affected objects will be released from the model moderator's control upon initial approval. This is useful for models in which you want to avoid unnecessary repetition of potentially expensive auto-approve/reject logic upon each object edit. This cannot be used for models in which you would like to approve (auto or manually) each object edit, because changes are not tracked and the moderation logic is not run. If the object needs to be entered back into moderation you can set its status to "Pending" by unapproving it. Default: False

keep_history When set to True this will allow multiple moderations per registered model instance. Otherwise there is only one moderation per registered model instance. Default: False.

notify_moderator Defines if notification e-mails will be send to moderator. By default when user change object that is under moderation, e-mail notification is send to moderator. It will inform him that object was changed and need to be moderated. Default: True

notify_user Defines if notification e-mails will be send to user. When moderator approves or reject object changes then e-mail notification is send to user that changed this object. It will inform user if his changes were accepted or rejected and inform him why it was rejected or approved. Default: True

subject_template_moderator Subject template that will be used when sending notifications to moderators. Default: moderation/notification_subject_moderator.txt

message_template_moderator Message template that will be used when sending notifications to moderator. Default: moderation/notification_message_moderator.txt

subject_template_user Subject template that will be used when sending notifications to users. Default: moderation/notification_subject_user.txt

message_template_user Message template that will be used when sending notifications to users. Default: moderation/notification_message_user.txt

Notes on auto moderation If you want to use auto moderation in your views, then you need to save user object that has changed the object in ModeratedObject instance. You can use following helper. Example:

```
moderation.register(UserProfile)

new_profile = UserProfile()

new_profile.save()

from moderation.helpers import automoderate

automoderate(new_profile, user)
```

Custom auto moderation If you want to define your custom logic in auto moderation, you can overwrite methods: `is_auto_reject` or `is_auto_approve` of GenericModerator class

Example:

```
class MyModelModerator(GenericModerator):

    def is_auto_reject(self, obj, user):
        # Auto reject spam
        if akismet_spam_check(obj.body):  # Check body of object for spam
            # Body of object is spam, moderate
            return self.reason('My custom reason: SPAM')
        super(MyModelModerator, self).is_auto_reject(obj, user)

moderation.register(MyModel, MyModelModerator)
```

Default context of notification templates

Default context:

content_type content type object of moderated object

moderated_object ModeratedObject instance

site current Site instance

How to pass extra context to email notification templates

Subclass GenericModerator class and overwrite `inform_moderator` and `inform_user` methods.:

```
class UserProfileModerator(GenericModerator):

    def inform_moderator(self,
                         content_object,
                         extra_context=None):
        '''Send notification to moderator'''
```

```
        extra_context={'test':'test'}
        super(UserProfileModerator, self).inform_moderator(content_object,
                                                           extra_context)

    def inform_user(self, content_object, user, extra_context=None)
        '''Send notification to user when object is approved or rejected'''
        extra_context={'test':'test'}
        super(CustomModerationNotification, self).inform_user(content_object,
                                                              user,
                                                              extra_context)

moderation.register(UserProfile, UserProfileModerator)
```

ModerationAdmin

If you have defined your own `save_model` method in your ModelAdmin then you must:

```
# Custom save_model in MyModelAdmin
def save_model(self, request, obj, form, change):
    # Your custom stuff
    from moderation.helpers import automoderate
    automoderate(obj, request.user)
```

Otherwise what you save in the admin will get moderated and automoderation will not work.

Message backend

By default the message backend used for sending notifications is *moderation.message_backends.EmailMessageBackend*, which is trigger a synchronous task on the main thread and call the *django.core.mail.send_mail* method.

You can write your own message backend class by subclassing *moderation.message_backends.BaseMessageBackend*, in order to use another api to send your notifications (Celery, RabbitMQ, ...).

Example of a custom message backend

```
class CustomMessageBackend(object):

    def send(self, **kwargs):
        subject = kwargs.get('subject', None)
        message = kwargs.get('message', None)
        recipient_list = kwargs.get('recipient_list', None)

        trigger_custom_message(subject, message, recipient_list)
```

Then specify the custom class in the moderator

```
from moderation.moderator import GenericModerator
from myproject.message_backends import CustomMessageBackend


class UserProfileModerator(GenericModerator):
    message_backend_class = CustomMessageBackend

moderation.register(UserProfile, UserProfileModerator)
```

Signals

`moderation.signals.pre_moderation` - signal send before object is approved or rejected

Arguments sent with this signal:

sender The model class.

instance Instance of model class that is moderated

status Moderation status, 0 - rejected, 1 - approved

`moderation.signals.post_moderation` - signal send after object is approved or rejected

Arguments sent with this signal:

sender The model class.

instance Instance of model class that is moderated

status Moderation status, 0 - rejected, 1 - approved

Forms

When creating ModelForms for models that are under moderation use BaseModeratedObjectForm class as ModelForm class. Thanks to that form will initialized with data from changed_object.:

```
from moderation.forms import BaseModeratedObjectForm


class ModeratedObjectForm(BaseModeratedObjectForm):

    class Meta:
        model = MyModel
```

Settings

DJANGO_MODERATION_MODERATORS This option is deprecated in favor of `MODERATION_MODERATORS`.

MODERATION_MODERATORS Tuple of moderators' email addresses to which notifications will be sent.

Contributing guide

Setup

Fork on GitHub

Before you do anything else, login/signup on GitHub and fork **django-moderation** from the **'GitHub project'_**.

Clone your fork locally

If you have git-scm installed, you now clone your git repo using the following command-line argument where <my-github-name> is your account name on GitHub:

```
git clone git@github.com:<my-github-name>/django-moderation.git
```

Local Installation

1. Create a virtualenv (or use virtualenvwrapper). Activate it.
2. cd into django-moderation
3. type `$ python setup.py develop`

Try the example projects

1. cd into example_project/
2. create the database: `$ python manage.py syncdb`
3. run the dev server: `$ python manage.py runserver`

How to get your pull request accepted

We want your submission. But we also want to provide a stable experience for our users and the community. Follow these rules and you should succeed without a problem!

Run the tests!

Before you submit a pull request, please run the entire django-moderation test suite via:

```
python setup.py test
```

If you add code/views you need to add tests!

We've learned the hard way that code without tests is undependable. If your pull request reduces our test coverage because it lacks tests then it will be **rejected**.

For now, we use the Django Test framework (based on unittest).

Keep your pull requests limited to a single issue

django-moderation pull requests should be as small/atomic as possible. Large, wide-sweeping changes in a pull request will be **rejected**, with comments to isolate the specific code in your pull request

Code style

Please follow PEP8 rules for code style.

Code structure

- moderation/admin.py - Django admin classes for moderation queue
- moderation/diff.py - used for generation of differences between model fields
- moderation/fields.py - SerializedObjectField code
- moderation/filterspecs.py - filters definitions used in Django admin queue
- moderation/forms.py - custom ModelForm class that uses unmoderated data instead of public one.
- moderation/helpers - moderation helpers functions
- moderation/managers - Managers used by moderation
- moderation/models - ModeratedObject class code
- moderation/moderator - base class for moderation options used during model registration with moderation
- moderation/register - code responsible for model registration with moderation
- moderation/signals - signals used by moderation

Test are located in directory tests/tests.

Each file is used for tests of different part of the moderation module.

Example: tests/unit/register - tests all things related with model registration with moderation system.

How to run django-moderation tests

1. Download source from http://github.com/dominno/django-moderation
2. Run: python setup.py test

Continuous Integration system

Continuous Integration system for django-moderation is available at:

https://travis-ci.org/dominno/django-moderation

Changelog

0.1 alpha (2010-03-11)

- Initial release

Added features

- configurable admin integration(data changed in admin can be visible on site when moderator approves it)
- moderation queue in admin
- html differences of changes between versions of objects
- configurable email notifications
- custom model form that allows to edit changed data of object

0.2 (2010-05-19)

- Added GenericModerator class that encapsulates moderation options for a given model.Changed register method, it will get only two parameters: model class and settings class.
- Added option to register models with multiple managers.
- Added options to GenericModerator class: auto_approve_for_superusers, auto_approve_for_staff, auto_approve_for_groups, auto_reject_for_anonymous, auto_reject_for_groups. Added methods for checking auto moderation.
- Added automoderate helper function.
- Changed moderated_object property in ModerationManager class, moderated object is get only once from database, next is cached in _moderated_object, fixed issue with not setting user object on changed_by attribute of ModeratedObject model.
- Fixed issue when loading object from fixture for model class that is registered with moderation. Now moderated objects will not be created when objects are loaded from fixture.
- Fixed issue with TypeError when generating differences of changes between model instances that have field with non unicode value ex. DateField.
- Fixed issue with accessing objects that existed before installation of django-moderation on model class.
- Fixed issue when more then one model is registered with moderation and multiple model instances have the same pk.
- Fixed issue with multiple model save when automoderate was used. Auto moderation in save method of ModeratedObject has been moved to separate method.
- Added admin filter that will show only content types registered with moderation in admin queue.
- Fixed issue when creating model forms for objects that doesn't have moderated object created.
- Added possibility of passing changed object in to is_auto- methods of GenericModerator class. This will allow more useful custom auto-moderation. Ex. auto reject if akismet spam check returns True.
- Added ability to provide custom auto reject/approve reason.
- Added option bypass_moderation_after_approval in to GenericModerator class that will release object from moderation system after initial approval of object.
- Other bug fixes and code refactoring.

0.3.2 (2012-02-15)

- Added visibility_column option in to GenericModerator class. Boost performance of database queries when excluding objects that should not be available publicly. Field must by a BooleanField. The manager that decides which model objects should be excluded when it were rejected, will first use this option to properly display (or hide) objects that are registered with moderation. Use this option if you can define visibility column in your model and want to boost performance. By default when accessing model objects that are under moderation, one extra query is executed per object in query set to determine if object should be excluded from query set. This method benefit those who do not want to add any fields to their Models. Default: None. Closes #19
- Added support for ImageField model fields on moderate object page.
- Made moderation work with south and grappelli
- Added possibility of excluding fields from moderation change list. Closes #23
- Moved ModerationManager class to moderation.register module, moved GenericModerator class to moderation.moderator module.

- Added auto_discover function that discover all modules that contain moderator.py module and auto registers all models it contain with moderartion.
- Efficiency improvement: get all info needed to filter a queryset in two SQL requests, rather than one for each object in the queryset.
- Added south migrations
- Added suport for foreignkey changes
- Add support for multi-table inheritance
- Add visible_until_rejected functionality
- Added specific initials in BaseModeratedObjectForm
- Added posibility to specify list of moderated fields
- Fixed SMTPRecipientsRefused when user has no email, when sending messages by moderation. Closes #48
- Added sorting of content types list on admin moderation queue

0.3.3 (2013-10-14)

- Tests refactor
- Added Travis CI
- Added CONTRIBUTING GUIDE

0.3.4 (2013-10-18)

- Dropped support for django 1.2.X

0.3.5 (2014-06-02)

- Added message backends
- Added support for custom user model
- Added support for django 1.6.X

0.3.6 (2014-06-09)

- Added support for python 3.X
- Dropped support for python 2.5
- Dropped support for django 1.3
- Added support for ForeignKey relations

0.4.0 (2016-08-25)

- Updated to support Django 1.7 - 1.9
- Added instructions for switching from South migrations to Django 1.7+ migrations
- Improved filter logic for Django 1.8+ to only create one additional query per queryset, instead of N additional queries (eg: one additional query per object in the querset)
- Renamed model fields to be shorter, less redundant, and more semantically correct
- Modified registry to add a `moderation_status` shortcut to registered models
- Added support for moderating multiple objects at once
- Changed model choice fields to use `Choices` from django-model-utils
- Deprecated the `DJANGO_MODERATION_MODERATORS` setting in favor of `MODERATION_MODERATORS`, which does the same thing
- Improved default email template formatting
- PEP8 and Flake Fixups
- Internal code and documentation typo fixes
- Bug fixes (specifically, closes #87)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search