django-model-publisher Documentation

Release 0.1.4

JP74

Sep 21, 2017

Contents

Contents:

CHAPTER 1

django-model-publisher

A handy mixin/abstract set of classes for provising a publisher workflow (draft, publish) to arbitrary Django models

Documentation

The full documentation is at https://django-model-publisher.readthedocs.org.

Quickstart

Install django-model-publisher:

```
pip install django-model-publisher
```

Features

- Django CMS placeholders support.
- Hvad/Parler support.
- Restrict user access to publish functions with user permissions.

CHAPTER 2

Installation

Install django-model-publisher (using pip):

```
pip install django-model-publisher
```

Add it to installed apps in your settings:

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

Add the middleware:

```
MIDDLEWARE_CLASSES = (
    ...
    'publisher.middleware.PublisherMiddleware',
)
```

Making models publishable

1. Have your model inherit from PublisherModel:

```
from publisher.models import PublisherModel


class Article(PublisherModel):
    pass
```

2. Update your database schema. If using South, you can run:

```
python manage.py schemamigration --auto <app_name>
python manage.py migrate <app_name>
```

The model records in the database will now be duplicated in a draft and a published version.

Django admin

If using Django admin, you can use the provided PublisherAdmin. You can also import PublisherPublishedFilter to use in `list_filter`:

```
from django.contrib import admin

from publisher.admin import PublisherAdmin

from .models import Article


class ArticleAdmin(PublisherAdmin):
    pass


admin.site.register(Article, ArticleAdmin)
```

Setting up the views

Use the provided views to serve either the draft or published version of your model. PublisherDetailView and PublisherListView are based on Django's DetailView and ListView respectively, but you can also use the PublisherViewMixin mixing in your custom views:

```
from .models import Article

from publisher.views import PublisherDetailView, PublisherListView


class ArticleListView(PublisherListView):
    model = Article


class ArticleView(PublisherDetailView):
    model = Article
```

Those views will only display the published version by default. To view the draft version, either follow the preview link from the admin, or append `?edit` at the end of the URL (note that you will need to be logged in).

CHAPTER 3

Usage

Viewing/previewing

The views provided will only display the published version by default. To preview the draft version, either follow the preview link from the admin, or append `?edit` at the end of the URL (note that you will need to be logged in).

Publishing/unpublishing

Press the “publish draft” button from an edit page to make your model public. If the model has been published, there will also be an “unpublished” button.

By default, the listing page displays a checkbox to quickly publish/unpublished models. The “Last changes” column highlights wether or not there’s unpublished changes. Clicking on the button in that column will publish the changes.

Discarding changes

If you are unhappy with the new draft, use the revert button to discard your draft changes.

CHAPTER 4

Permissions

To restrict publish access to your model, add ‘PublisherModel.Meta’ to your models Meta class:

```
class Article(PublisherModel):

    class Meta(PublisherModel.Meta):
            pass
```

Then run the following management command:

```
python manage.py update_permissions
```

You should now have the “Can publish” permission available for your model.

CHAPTER 5

Handling relations

django-model-publisher does not provide any mechanism to publish related models, as it can be quite specific. Implementing classes wishing to do so will have to override the `clone_relations()` method from `PublisherModelBase`, which takes two arguments: `src_obj` (the draft instance), and `dst_obj` (the published instance).

Here's a simple example which maintains the relations with a many to many model:

```
def clone_relations(self, src_obj, dst_obj):
    dst_obj.sites.add(*src_obj.sites.all())
```

CHAPTER 6

Signals

`publisher_pre_publish` - Sent when a model about to be published (the draft is sent).

`publisher_publish_pre_save_draft` - Sent when a model is being published, before the draft is saved (the draft is sent).

`publisher_post_publish` - Sent when a model is published (the draft is sent).

`publisher_pre_unpublish` - Sent when a model about to be unpublished (the draft is sent).

`publisher_post_unpublish` - Sent when a model is unpublished (the draft is sent).

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/jp74/django-model-publisher/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

django-model-publisher could always use more documentation, whether as part of the official django-model-publisher docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jp74/django-model-publisher/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-model-publisher* for local development.

1. Fork the *django-model-publisher* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-model-publisher.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-model-publisher
$ cd django-model-publisher/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 publisher tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/jp74/django-model-publisher/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_publisher
```