dj-contentmodel Documentation

Release 0.1.5

David S.

June 13, 2016

Contents

Contents:

CHAPTER 1

dj-contentmodel

Boilerplate for taxonomies, content collections, and content models using django-mptt and django-taggit.

1.1 Features

- Drag and drop construction of taxonomies.
- Arbitrary relationships among groups, taxonomies, collections, content, and attachments.
- Templates for displaying taxonomies with and without collections' contents.

1.2 Documentation

dj-content model is just boilerplate for django-mptt and django-taggit. It defines a hierarchical structure with arbitrary relationships via collections. The Quickstart example is a sufficient starting point for many projects. Each abstract class provided by dj-contentmodel is a descendant of django.db.models, so extend them as you would a Django model. The Group and Taxonomy classes are also descendants of django-mptt's MPTTModel class and can be extended accordingly. Additional examples dj-contentmodel's abstract classes are available at https://dj-contentmodel.readthedocs.org.

1.3 Quickstart

First, install dj-contentmodel:

```
pip install dj-contentmodel
```

Then, import the abstract base classes.:

```
from dj_contentmodel.models import Taxonomy, Collection, Content, Attachment
```

Next, subclass the imported classes to create taxonomies and content models as needed. The following example is of a minimum configuration. The names of defined classes are arbitrary, but the relationships among classes are not.

```
class Sitemap(Taxonomy):
    """Main navigation"""
    collections = models.ManyToManyField('Bucket', blank=True)
    class Meta:
        verbose_name = "Category"
        verbose_name_plural = "Categories"
```

```
    ...

class Bucket(Collection):
    """Arbitrary collections to group content."""
    contents = models.ManyToManyField('Page', blank=True)
    ...

class Page(Content):
    ...

class Report(Attachment):
    parents = models.ManyToManyField('Page', blank=True)
    ...
```

Finally, register your models with the admin.

```
from django.contrib import admin
from mptt.admin import DraggableMPTTAdmin

admin.site.register(
    Sitemap,
    DraggableMPTTAdmin,
    list_display=(
        'tree_actions',
        'indented_title',),
    list_display_links=(
        'indented_title',),)
admin.site.register(Bucket)
admin.site.register(Page)
admin.site.register(Report)
```

Without a migration, it may be necessary to create the tables.:

```
python manage.py migrate --run-syncdb
```

1.4 Running Tests

Does the code actually work? For now I have my fingers crossed. Later iterations will be test driven and include integration and performance testing.

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements_test.txt
(myenv) $ python runtests.py
```

1.5 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage
- django-mptt
- django-taggit
- pip-tools

CHAPTER 2

Installation

At the command line.

```
$ easy_install dj-contentmodel
```

If you have virtualenvwrapper installed.

```
$ mkvirtualenv dj-contentmodel
$ pip install dj-contentmodel
```

If you are using conda.

```
$ conda create --name dj-contentmodel python=3.5
$ source activate dj-contentmodel
$ pip install dj-contentmodel
```

CHAPTER 3

Usage

First, install dj-contentmodel:

```
pip install dj-contentmodel
```

Add django-mptt, django-taggit, and dj-contentmodel in INSTALLED_APPS.

```
INSTALLED_APPS = [
    ...
    'mptt',
    'taggit',
    'dj_contentmodel',
    ...
]
```

Then, import the abstract base classes.:

```
from dj_contentmodel.models import Taxonomy, Collection, Content, Attachment
```

Next, subclass the imported classes to create taxonomies and content models as needed. The following example is of a minimum configuration. The names of defined classes are arbitrary, but the relationships among classes are not.

```
class Sitemap(AbstractTaxonomy):
    """Main navigation"""
    collections = models.ManyToManyField('Collection', blank=True)

class Collection(AbstractCollection):
    """Arbitrary collections to group content."""
    contents = models.ManyToManyField('Page', blank=True)

class Page(AbstractContent):
    body = models.TextField(blank=True, null=True)

class Attachment(AbstractAttachment):
    parents = models.ManyToManyField('Page', blank=True)
```

After creating your models register the subclass of Taxonomy using DraggableMPTTAdmin. Collection, Content, and Attachment subclasses do not require customization.

```
from django.contrib import admin
from mptt.admin import DraggableMPTTAdmin
from .models import Sitemap, Collection, Page, Attachment
admin.site.register(
    Sitemap,
    DraggableMPTTAdmin,
```

```
    list_display=(
        'tree_actions',
        'indented_title',),
    list_display_links=(
        'indented_title',),)
admin.site.register(Collection)
admin.site.register(Page)
admin.site.register(Attachment)
```

Without a migration, it may be necessary to create the tables using syncdb.

```
python manage.py migrate --run-syncdb
python manage.py migrate dj-contentmodel
```

Once a taxonomy is defined and registered in the admin, create your views. For simplicity, funtional views are shown here but class based views are preferred. Until authentication and authorization are implemented, you may filter taxonomies by group to feign authorization but this is not secure.

```
from django.shortcuts import render_to_response, RequestContext
from .models import Sitemap


def get_content_tree(request, group=None):
    if group == None:
        nodes = Sitemap.objects.all()
    else:
        nodes = Sitemap.objects.filter(groups=group)
    templatePath = "dj_contentmodel/taxonomy.html"    #use navigation.html to omit content nodes
    return render_to_response(templatePath, {'nodes': nodes}, context_instance=RequestContext(request
```

Then, create your urls.

```
urlpatterns = [
    url(r'^navigation/(?P<group>[0-9]+)?$', views.get_navigation_tree),
    url(r'^sitemap/(?P<group>[0-9]+)?$', views.get_content_tree),
]
```

Finally, create a template from the templates provided.

Sitemap.

```
{% load mptt_tags %}
{% load staticfiles %}
<ul>
    {% recursetree nodes %}
        <li>
            <a href="{% url 'app:url' %}{{ node.id }}">{{ node.name }}</a>
                {% if not node.is_leaf_node %}
                    <ul class="children">
                        {{ children }}
                    </ul>
                {% endif %}
        </li>
    {% endrecursetree %}
</ul>
```

Sitemap and associated content. Additional tags are provided by django-mptt and django-taggit.Note that content assigned to multiple collections associated with the same taxa will display duplicate content.

```
{% load mptt_tags %}
<ul>
    {% recursetree nodes %}
        <li>
            <a href="{% url 'app:url' %}{{ node.id }}">{{ node.name }}</a>
            {% if not node.is_leaf_node %}
                <ul class="children">
                    {{ children }}
                </ul>
            {% endif %}
            {% if node.collections.contents.count > 0 %}
                <ul>
                    {% for content in node.collections.contents.prefetch_related %}
                        <li><a href="{% url 'app:url' %}{{ content.id }}">{{ content.name }}</a></li>
                    {% endfor %}
                </ul>
            {% endif %}
        </li>
    {% endrecursetree %}
</ul>
```

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/infosmith/dj-contentmodel/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/infosmith/dj-contentmodel/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 :)

4.2 Get Started!

Ready to contribute? Here's how to set up *dj-contentmodel* for local development.

1. Fork the *dj-contentmodel* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dj-contentmodel.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 dj-contentmodel
$ cd dj-contentmodel/
$ 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 dj_contentmodel 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.

4.3 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/infosmith/dj-contentmodel/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

CHAPTER 5

Credits

5.1 Development Lead

- David S. <infosmith@protonmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2016-06-07)

- First release on PyPI.