django-atom Documentation

Release 0.15.2

Adam Dobrawy

Aug 04, 2017

Contents

Contents:

CHAPTER 1

django-atom

A different stuff for Django to faster make a world a better place.

Documentation

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

Quickstart

Install django-atom:

```
pip install django-atom
```

Then use it in a project:

```
import atom
```

Extensions

slugify

Example usage in `settings.py` add

```
AUTOSLUG_SLUGIFY_FUNCTION = 'atom.ext.slugify.slugifier.ascii_slugify'
```

Required `unicode-slugify`,`django-autoslug`.

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-atom
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-atom
$ pip install django-atom
```

CHAPTER 3

Usage

To use django-atom in a project:

```
import atom
```

CHAPTER 4

Django-guardian

There is some extensions for django-guardian package.

Forms

class atom.ext.guardian.forms.**PermissionsTranslationMixin**(**args*, ***kwargs*)

class atom.ext.guardian.forms.**TranslatedUserObjectPermissionsForm**(**args*, ***kwargs*)

base_fields = OrderedDict()

declared_fields = OrderedDict()

media

Tests

Example usage

```
from django.core.urlresolvers import reverse
from django.test import TestCase

from atom.ext.guardian.tests import PermissionStatusMixin
from feder.monitorings.factories import MonitoringFactory
from feder.users.factories import UserFactory

class ObjectMixin(object):
    def setUp(self):
        self.user = UserFactory(username='john')
        self.monitoring = self.permission_object = MonitoringFactory()
        self.case = CaseFactory(monitoring=self.monitoring)
```

```
        self.from_user = OutgoingLetterFactory(title='Wniosek',
                                               case__monitoring=self.monitoring)
        self.from_institution = IncomingLetterFactory(title='Odpowiedz',
                                                      case__monitoring=self.
↪monitoring)

class LetterRssFeedTestCase(ObjectMixin, PermissionStatusMixin, TestCase):
    status_anonymous = 200
    status_no_permission = 200
    permission = []

    def get_url(self):
        return reverse('letters:rss')
```

Docstrings

class atom.ext.guardian.tests.**PermissionStatusMixin**

Mixin to verify object permission status codes for different users

Require user with username='john' and password='pass'

Attributes: permission (list): Description status_anonymous (int): Status code for anonymouser status_has_permission (int): Status code for user with permission status_no_permission (403): Status code for user without permission url (string): url to test

get_permission()

Returns the permission to assign for granted permission user

Returns: list: A list of permission in format `codename.permission_name`

Raises: ImproperlyConfigured: Missing a permission to assign

get_permission_object()

Returns object of permission-carrying object for grant permission

get_url()

Get url to tests

Returns: str: url to test

Raises: ImproperlyConfigured: Missing a url to test

grant_permission()

Grant permission to user in self.user

Returns: TYPE: Description

login_permitted_user()

Login client to user with granted permissions

permission = None

setUp()

status_anonymous = 302

status_has_permission = 200

status_no_permission = 403

test_status_code_for_anonymous_user()

A test status code of response for anonymous user

test_status_code_for_privileged_user()
A test for status code of response for privileged user

Grant permission to permission-carrying object and login before test

test_status_code_for_signed_user()
A test for status code of response for signed (logged-in) user

Only login before test.

url = None

CHAPTER 5

Django-crispy-forms

There is some extensions for django-crispy-forms package.

Filters

class atom.ext.crispy_forms.filters.**CrispyFilterMixin**

form

form_class = 'form-inline'

Forms

class atom.ext.crispy_forms.forms.**BaseTableFormSet**(**args*, ***kwargs*)

class atom.ext.crispy_forms.forms.**FormHorizontalMixin**(**args*, ***kwargs*)

class atom.ext.crispy_forms.forms.**FormsetHelper**(*form=None*)

form_method = 'post'

form_tag = False

class atom.ext.crispy_forms.forms.**HelperMixin**(**args*, ***kwargs*)

form_helper_cls
alias of FormHelper

class atom.ext.crispy_forms.forms.**InlineTableFormSet**(**args*, ***kwargs*)

class atom.ext.crispy_forms.forms.**SingleButtonMixin**(**args*, ***kwargs*)
Dynamically add crispy button to form layout

Usage of mixins is obvious:

```
from django import forms
from atom.ext.crispy_forms.forms import SingleButtonMixin

class PersonModelForm(SingleButtonMixin, forms.ModelForm):

    class Meta:
        model = Person
```

HelperMixin

action_text
A text of added action submit button.

In standards it detects when forms save or update objects

string – A text used in button

class atom.ext.crispy_forms.forms.**TableFormSet**(**args*, ***kwargs*)

class atom.ext.crispy_forms.forms.**TableFormSetHelper**(*form=None*)

template = 'bootstrap/table_inline_formset.html'

class atom.ext.crispy_forms.forms.**TableFormSetMixin**(**args*, ***kwargs*)

Views

class atom.ext.crispy_forms.views.**FormSetMixin**

form_valid(*form*)

formset = None

formset_cls
alias of BaseTableFormSet

formset_invalid(*form*, *formset*)

formset_valid(*form*, *formset*)

get_context_data(***kwargs*)

get_form(**args*, ***kwargs*)

get_formset()

get_formset_kwargs()

get_formset_valid_message()

get_instance()

get_success_url()

inline_form_cls = None

inline_model = None

CHAPTER 6

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/ad-m/django-atom/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-atom could always use more documentation, whether as part of the official django-atom 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/ad-m/django-atom/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-atom* for local development.

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

```
$ git clone git@github.com:your_name_here/django-atom.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-atom
$ cd django-atom/
$ 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 atom 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/ad-m/django-atom/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_atom
```

CHAPTER 7

Credits

Development Lead

- Adam Dobrawy <naczelnik@jawnosc.tk>

Contributors

None yet. Why not be the first?

CHAPTER 8

History

0.1.0 (2015-08-01)

- First release on PyPI.

0.2.0 (2015-08-04)

- Added AutocompleteChoiceFilter
- Add CSS-class settings for CrispyFormFilter
- Drop strict requirements for django-tinycontent

0.3.0 (2015-08-04)

- Split class to independent package to resolve dependencies issues

0.6.0 (2015-10-25)

- A lot

0.10.0 (2015-11-21)

- Add asci_slugifier
- Add absolute_import to crispy_forms

0.11.0 (2015-12-10)

- Fix support of dj19 by autocomplete_light API changes

0.12.0 (2016-12-4)

- Add `atom.ext.guardian.tests.PermissionStatusMixin`

0.12.8 (2016-12-16)

- Mark code as Python 3 compatible

0.14.0 (2017-06-1)

- Add `atom.mixins.AdminTestCaseMixin`

0.14.1 (2017-06-1)

- Add base class validation in `atom.mixins.AdminTestCaseMixin`

0.14.2 (2017-06-1)

- Fix username and message in `atom.mixins.AdminTestCaseMixin`

0.14.3 (2017-06-1)

- Fix django 1.8-1.9 compatibility of `atom.mixins`

0.15.0 (2017-07-21)

- Add management command `generate_factory`, `generate_routers`, `generate_serializers`, `generate_viewsets`, `generate_viewsets_tests`
- Add serializer `atom.ext.contenttypes.ContentTypeSerializer`, `atom.ext.sites.serializer.SiteSerializer`
- Add mixin `atom.ext.rest_framework.ViewSetTestCaseMixin`

0.16.0 (2017-08-4)

- Fix management commands import

CHAPTER 9

Docstrings

Filters

Forms

class atom.forms.**AuthorMixin**

save(**args*, ***kwargs*)

class atom.forms.**PartialMixin**

classmethod partial(**args*, ***kwargs*)

Models

class atom.models.**AttachmentBase**(**args*, ***kwargs*)

class Meta

abstract = False

verbose_name = u'Attachment'

verbose_name_plural = u'Attachments'

AttachmentBase.**attachment**
The descriptor for the file attribute on the model instance. Returns a FieldFile when accessed so you can do stuff like:

```
>>> from myapp.models import MyModel
>>> instance = MyModel.objects.get(pk=1)
>>> instance.file.size
```

Assigns a file object on assignment so you can do:

```
>>> with open('/path/to/hello.world', 'r') as f:
...     instance.file = File(f)
```

AttachmentBase.**filename**

AttachmentBase.**get_absolute_url**()

Views

class atom.views.**ActionMessageMixin**

post(*request*, **args*, ***kwargs*)

class atom.views.**ActionMixin**

action()

get_success_url()

post(*request*, **args*, ***kwargs*)

success_url = None

class atom.views.**ActionView**(***kwargs*)

template_name_suffix = '_action'

class atom.views.**BaseActionView**(***kwargs*)
Base view for action on an object. Using this base class requires subclassing to provide a response mixin.

class atom.views.**CreateMessageMixin**

get_form_valid_message()

class atom.views.**DeleteMessageMixin**

delete(*request*, **args*, ***kwargs*)

get_success_message()

hide_field = None

class atom.views.**FormInitialMixin**

get_initial(**args*, ***kwargs*)

class atom.views.**MessageMixin**

get_success_message()

success_message = None

class atom.views.**UpdateMessageMixin**

get_form_valid_message()

Mixins

class atom.mixins.**AdminTestCaseMixin**

QUERY_LIMIT = 30

change_viewname = None

changelist_viewname = None

delete_viewname = None

factory_cls = None

get_change_viewname()

get_changelist_viewname()

get_delete_viewname()

get_factory_cls()

get_history_viewname()

history_viewname = None

model = None

setUp()

test_change_view_queries_limit()

test_changelist_queries_limit()

test_status_change_view()

test_status_changelist()

test_status_delete_view()

test_status_history_view()

user_factory_cls = None

Python Module Index

a

Index

T

U

V