barbeque Documentation

Release 1.8.2

Moccu GmbH & Co. KG

Dec 04, 2018

Contents

Barbeque is a collection of custom extensions and helpers, mostly related to the Django Web framework.

These include a commands framework, logging helpers, django-anylink, filer extensions and much more.

CHAPTER 1

Getting it

You can get `barbeque` by using pip:

```
$ pip install barbeque
```

If you want to install it from source, grab the git repository and run setup.py:

```
$ git clone git://github.com/moccu/barbeque.git
$ cd barbeque
$ python setup.py install
```

CHAPTER 2

Compatibility with versions of Python and Django

At this time we test and thrive to support valid combinations of Python 2.7, 3.4, 3.5 and pypy with Django versions 1.8+. The most recent django master usually works too.

CHAPTER 3

Contents

3.1 Testing helpers

barbeque.testing.**get_messages_from_cookie**(*cookies*)

Get `messages` from `cookies`

Example:

```
from django.test import TestCase, Client
from barbeque.testing import get_messages_from_cookie


class MyTest(TestCase):

    def test_error_message_on_access_denied(self):
        client = Client()

        response = client.get('/restricted/')

        assert response.status_code == 302

        messages = list(get_messages_from_cookie(response.cookies))

        assert len(messages) == 1
        assert messages[0].tags == 'error'
        assert response['Location'] == 'http://testserver/login/?next=/restricted/
↪'
```

3.2 Shortcuts

get_object_or_none(klass, *args, **kwargs):

Uses `get()` to return an object, or returns `None` if the object doesn't exist.

`klass` may be a `Model`, `Manager`, or `QuerySet` object. All other passed arguments and keyword arguments are used in the `get()` query.

Note: Like with `get()`, a `MultipleObjectsReturned` will be raised if more than one object is found.

3.3 Logging helpers

barbeque.logging.**logged**(*obj*)
Decorator to mark a object as logged.

This injects a `logger` instance into `obj` to make log statements local and correctly named.

Example:

```
>>> @logged
... class MyClass(object):
...     def foo(self):
...         self.logger.warning('logged')
...
>>> import logging
>>> logging.basicConfig()
>>> obj = MyClass()
>>> obj.foo()
WARNING:__main__.MyClass:logged
```

Supported objects:

- Functions
- Methods
- Classes
- Raw names (e.g, user at module level)

barbeque.logging.**get_logger**(*obj*)
Return a logger object with the proper module path for `obj`.

3.4 Form mixins

class barbeque.forms.**PlaceholderFormMixin**(**args*, ***kwargs*)
Adds placeholder attributes to input all visible fields. The placeholder will be named after the field label.

class barbeque.forms.**ItemLimitInlineMixin**
Mixin that validates the min/max number of forms in an admin inline.

Usage:

```
class MyInlineAdmin(ItemLimitInlineMixin, admin.StackedInline):
    min_items = 1
    max_items = 4
```

Please note that both options are optional and default to `None` thus no validation takes place.

3.5 File helpers

barbeque.files.**upload_to_path**(*base_path*, *attr=None*, *uuid_filename=False*)
Returns a callback suitable as `upload_to` argument for Django's `FileField`.

Parameters

- **base_path** – Base folder structure for uploaded files. Note that you cannot use *strftime* placeholders here.
- **attr** – Attribute to use for base path generation.
- **uuid_filename** – Render file names as UUIDs.

Usage:

```
class Picture(models.Model):
    image = models.ImageField(_('Image'), upload_to=upload_to_path('uploads/
↪images/'))
```

Use `attr` for base path generation:

```
class Picture(models.Model):
    image = models.ImageField(
        _('Image'),
        upload_to=upload_to_path('uploads/%s/images/', attr='category.name')
    )
    category = models.ForeignKey(Category)
```

class barbeque.files.**MoveableNamedTemporaryFile**(*name*)
Wraps `NamedTemporaryFile()` and implements *chunks*, *close* and *temporary_file_path* on top of it.

Suitable for the django storage system to allow files to simply get moved to it's final location instead of copying the file-contents.

3.6 django-filer extensions

class barbeque.filer.**AdminFileFormField**(*rel*, *queryset*, *to_field_name*, **args*, ***kwargs*)
Add additional validation capabilities for django-filer.

Parameters

- **extensions** – A whitelist of extensions.
- **alt_text_required** – Validate that the `default_alt_text` is set.

class barbeque.filer.**FilerFileField**(*verbose_name*, **args*, ***kwargs*)
FilerFileField implementation that forwards `extensions` and `alt_text_required` to *`AdminFileFormField`*.

3.7 Excel/CSV Exporter

An exporter framework that allows you to export a queryset to Excel XLSX or CSV directly from the django `ModelAdmin` changelist view.

Example:

```
from barbeque.exporter import action_export_factory
from django.contrib import admin

from .models import User


class UserAdmin(admin.ModelAdmin):
    actions = (
        action_export_factory('xlsx')
    )

    export_fields = ('id', 'first_name', 'last_name', 'street', 'city')

    actions = [
        action_export_factory('csv', 'Export as CSV', export_fields),
        action_export_factory('xlsx', 'Export as XLSX', export_fields)
    ]
```

3.8 JSON Encoder

A json encoder extension based on `DjangoJSONEncoder` that supports manualize serialization of objects.

Given the following model:

```
from django.forms.models import model_to_dict

class Recipe(models.Model):
    url = models.URLField(max_length=2048, blank=True)
    title = models.CharField(max_length=80)
    author = models.ForeignKey('accounts.User')
    description = models.TextField(blank=True)

    def serialize(self):
        data = model_to_dict(self, fields=('url', 'title', 'description'))
        data['author'] = self.author.pk
        return data
```

Now we're able to serialize all instances of `Recipe` to json.

```
>>> import json
>>> from barbeque.encoders import SerializableModelEncoder

>>> recipes = Recipe.objects.all()
>>> print(json.dumps(recipes, cls=SerializableModelEncoder))
'[{"url": "", "title": "Chocolate Cookies", "author": 28}]'
```

3.9 Context Processors

3.9.1 settings

Expose specific settings directly into the template context.

Can be configured via `BARBEQUE_EXPOSED_SETTINGS` and defaults to `['DEBUG']`

3.10 Compressor support

Adds support for new filters and compressors to add support for UglifyJS.

3.10.1 Filters

barbeque.compressor.UglifyJSFilter

A filter that uses UglifyJS to compress JavaScript code.

Integration:

```
COMPRESSOR_JS_FILTERS = [
    'barbeque.compressor.UglifyJSFilter',
]
```

3.10.2 Compressors

barbeque.compressor.UglifyJSSourcemapCompressor

A compressor that compiles javascript code using UglifyJS and generates a proper sourcemap.

Integration:

```
COMPRESS_JS_COMPRESSOR = 'barbeque.compressor.UglifyJSSourcemapCompressor'
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index