Pyrocumulus Documentation

Release 0.4.4

Juca Crispim

July 03, 2014

Contents

Easy restful web services with tornado and mongodb on python 3.

Write some models with mongoengine, create a new pyrocumulus RestApplication instance and that's it. You're done. You've got a free restful API for your mongoengine documents.

- Automatic restful API with optional CORS and jsonp support.
- Easy tornado application testing with `pyrocumulus.testing.TornadoApplicationTestCase`
- Little improvements on static files and templates.

CHAPTER 1

Getting started

1.1 Install

Pyrocumulus installation is straight-forward. You can install it using pip or from sources.

Note: In order to make things easy, you better create a new virtualenv (using python 3) to install pyrocumulus

1.1.1 Using pip

Simple like:

```
$ pip install pyrocumulus
```

That's all.

1.1.2 From sources

The code for Pyrocumulus is hosted on gitorious. So, clone it.

```
$ git clone git@gitorious.org:pyrocumulus/pyrocumulus.git
```

Then, install all dependencies (including tests and docs dependencies);

```
$ cd pyrocumulus
$ pip install -r requirements_env.txt
```

And then, run tests.

```
$ python setup.py test
$ python setup.py test --test-suite=functional_tests
```

1.2 Tutorial

The main feature of pyrocumulus is automatic create restful APIs for mongodb documents. Let't try it!

1.2.1 Creating a new project

First, create a new project using the *pyrocumulus* command in your command line:

```
$ pyrocumulus createproject ~/pyrotut
$ cd ~/pyrotut
```

1.2.2 Writing your models with mongoengine

In this tutorial, we'll use the same models used on mongoengine tutorial. So, in the file pyrotut/models.py, write these models:

```
from mongoengine import Document, EmbeddedDocument
from mongoengine.fields import (StringField, ReferenceField, IntField,
                                EmbeddedDocumentField, ListField)


class User(Document):
    email = StringField(required=True)
    first_name = StringField(max_length=50)
    last_name = StringField(max_length=50)


class Comment(EmbeddedDocument):
    content = StringField()
    name = StringField(max_length=120)


class Post(Document):
    title = StringField(max_length=120, required=True)
    # reverse_delete_rule=CASCADE
    authors = ListField(ReferenceField(User, reverse_delete_rule=2))
    tags = ListField(StringField(max_length=30))
    comments = ListField(EmbeddedDocumentField(Comment))

    meta = {'allow_inheritance': True}


class TextPost(Post):
    content = StringField()


class ImagePost(Post):
    image_path = StringField()


class LinkPost(Post):
    link_url = StringField()
```

1.2.3 Configuring the restful api

Using the models we created earlier, lets create a `pyrocumulus.web.applications.RestApplication` instance.

In the file pyrotut/web.py, write the following code:

```
from pyrocumulus.web.applications import RestApplication
from pyrotut.models import User, TextPost, LinkPost, ImagePost

simple_blog_api = RestApplication(User, TextPost, LinkPost, ImagePost)
```

And that's it. You have a rest API for your documents.

1.2.4 Setting up config

Before we run our application, we need to set up some config variables. So, in the file settings.py, change your database settings and the APPLICATIONS variable

```
PROJECT_NAME = 'pyrotut'
TORNADO_PORT = 8888

DATABASE = {'default': {'HOST': 'localhost',
                        'PORT': 27017,
                        'NAME': 'pyrotut'}}

# list of applications to be served by tornado
APPLICATIONS = ['pyrotut.web.simple_blog_api',]
```

And that's it. We have a new project created. Now, let's try out something.

1.2.5 Testing the API

For testing, we'll use the package requests, so, install it.

```
$ pip install requests
```

Now, is time to test our api, but first, lets see which url were mapped by our RestApplication.

```
>>> from pyrotut.web import application
>>> application.urls
[('/api/user/(.*)', ...),
 ('/api/textpost/comments/(.*)', ...),
 ('/api/textpost/(.*)', ...),
 ('/api/linkpost/comments/(.*)', ...),
 ('/api/linkpost/(.*)', ...),
 ('/api/imagepost/comments/(.*)', ...),
 ('/api/imagepost/(.*)', ...)]
>>>
```

As we can see, `pyrocumulus.web.applications.RestApplication` creates a url for each one of our Documents, and URLs for its EmbeddedDocuments too. Then, we can send HTTP requests to it and use our automatic api.

So, first, lets run a instance of the tornado web server and then open a Python shell and test our application.

```
$ python pyromanager.py runtorando --daemonize
$ python
```

Now, inside the python shell, we can start sending requests to our application

```
>>> import requests
>>> # base url for api
>>> base_url = 'http://localhost:8888/api/'
```

```
>>> user_url = base_url + 'user/'
>>> post_url = base_url + 'textpost/'
>>> comments_url = post_url + 'comments/'
```

First, create a user to be the posts' author

```
>>> user_email = 'user@email.com'
>>> first_name = 'user'
>>> last_name = 'niceGuy'
>>> params = {'email': user_email,
...           'first_name': first_name,
...           'last_name': last_name}
...
>>> response = requests.put(user_url, params)
>>> response.connection.close()
>>> user_id = response.json()['id']
>>> user_id
'52ba237a7c1c8452919d03f8'
```

Now, create a post using the user created earlier

```
>>> post_title = 'My First Post'
>>> post_content = """Mussum ipsum cacilds, vidis litro abertis. """
>>> params = {'title': post_title,
...           'content': post_content,
...           'tags': ['pyrotut', 'pyrocumulus', 'mussum'],
...           'authors': [user_id]}
...
>>> response = requests.put(post_url, params)
>>> response.connection.close()
>>> post_id = response.json()['id']
>>> post_id
'52ba23be7c1c8452919d03f9'
```

And then, make a comment!

```
>>> comment_name = 'john doe'
>>> comment_content = 'bla bla bla...'
>>> params = {'name': comment_name,
...           'content': comment_content,
...           'parent_id': post_id}
...
>>> response = requests.put(comments_url, params)
>>> response.connection.close()
>>> response = requests.get(post_url, params={'id': post_id})
>>> response.connection.close()
post_content = response.json()['content']
>>> print(post_content)
Mussum ipsum cacilds, vidis litro abertis.
>>> post_comments = response.json()['comments']
>>> post_comments
[{'name': 'john doe', 'content': 'bla bla bla...'}]
```

That's it. We created a user, a post and then sent a comment to that post. When retrieving the post, we get the comments list together, because our handler has as default depth the value 1. If you want only the comments list, to paginate them, you can use:

```
>>> requests.get(comments_url + 'list', params={'parent_id': post_id,
...                                             'page': 1, 'max': 10})
```

1.3 Serving static files

Tornado already has a static file handler, which has a little improvement in pyrocumulus. Using the `pyrocumulus.web.handlers.StaticFileHandler` you can set multiple directories to be used as root dir for static files in your settings file.

To use it is quite simple. First you need to set the variables STATIC_URL and STATIC_DIRS in your settings file.

```
STATIC_URL = '/static/'
STATIC_DIRS = ['/home/me/somedir',]
```

Now, instantiate StaticApplication

```
from pyrocumulus.web.applications import StaticApplication

staticapp = StaticApplication()
```

And, finally, put your staticapp in the APPLICATIONS settings variable

```
APPLICATIONS = ['myapp.web.staticapp', ..., ..., ...]
```

1.4 Using templates

Pyrocumulus has an simple request handler that uses tornado's template system with a little improvement on how to look for templates.

To use that, simply subclass `pyrocumulus.web.handlers.TemplateHandler`.

```
from pyrocumulus.web.handlers import TemplateHandler


class MyHandler(TemplateHandler):
    def get(self):
        template = 'myfile.html'
        extra_context = {'some': 'thing'}
        self.render_template(template, extra_context)
```

`pyrocumulus.web.handlers.TemplateHandler` makes use of the tornado template system, so you can use all features of that in your templates [1]. It takes root dir from the TEMPLATE_DIRS settings variable.

[1] Look the tornado.template docs here

CHAPTER 2

Guides

2.1 Understandig the rest API

The main functionality of pyrocumulus is easily create rest apis based on your mongoengine documents. In this section lets take closer look on how it works [1].

2.1.1 How does it work?

This is quite simple. Pyrocumulus has a special request handler, `pyrocumulus.web.handlers.RestHandler`. It get a mongoengine.Document as a parameter and creates a restful api for it with optional CORS support.

To use that, you need to map an url to `pyrocumulus.web.handlers.RestHandler` and associate it with a document. If your document has embedded documents, they also need to be mapped to urls, but using the `pyrocumulus.web.handlers.EmbeddedDocumentHandler` class. To easily url configuration, pyrocumulus has the `pyrocumulus.web.urlmappers.DocumentURLMapper` class, wich creates all urls needed, including urls to EmbeddedDocuments.

After your urls are done, you need to create an *tornado.web.Application* to use your urls. To package all that together, you can use the `pyrocumulus.web.applications.RestApplication` class. You just pass your documents as paramenters to it and everything else is created for you.

2.1.2 RestHandler

Lets start understading how `pyrocumulus.web.handlers.RestHandler` works. In order to use that, you need to associate a mongoengine.Document to that handler and map a URL to. Its done using the URLSpec object. Something like this:

```
>>> from pyrocumulus.web.handlers import RestHandler
>>> from myapp.models import MyDocument
>>> from tornado.web import URLSpec
>>> urlspec = URLSpec('/api/mydocument/(.*)', RestHandler, model=MyDocument)
```

When you map a URL to `pyrocumulus.web.handlers.RestHandler`, you get a rest api for your documet, supporting the following operations:

[1] For a better understanding on how RequestHadler works, you should read the tornado documentation on RequestHadler here

- GET request to *YOUR_DOCUMENT_URL* or *YOUR_DOCUMENT_URL* + *'get'* to retrieve a specific object from database. Like this:

```
>>> params = {'id': '1232ua7d330akd0'}
>>> requests.get('/api/mydocument/', params=params)
```

- GET request to *YOUR_DOCUMENT_URL* + *'list'* to list objects. You can use the `page` and `max` params to paginate the result and the order_by param for ordering that.

```
>>> params = {'page': 1, 'max': 10, 'order_by': 'some_field'}
>>> requests.get('/api/mydocument/list', params=params)
```

- PUT request to *YOUR_DOCUMENT_URL* or POST request to *YOUR_DOCUMENT_URL* + *'put'* to insert a new object into the database.

```
>>> params = {'name': 'Some name', 'location': 'Somewhere'}
>>> request.put('/api/mydocument/', params=params)
```

- DELETE request to *YOUR_DOCUMENT_URL* or POST request to *YOUR_DOCUMENT_URL* + *'delete'* to delete an object from database

```
>>> params = {'id': 'a9982lsdfk34kd'}
>>> requests.delete('/api/mydocument/', params=params)
```

If your document has embedded documents, they also need to be mapped, but using the `pyrocumulus.web.handlers.EmbeddedDocumentHandler` class.

```
>>> from pyrocumulus.web.handlers import EmbbeddeDocumentRestHandler
>>> from myapp.models import MyDocument, MyEmbeddedDocument
>>> from tornado.web import URLSpec
>>> urlspec = URLSpec('/api/mydocument/embeddedfield/(.*)',
                      EmbeddedDocumentRestHandler,
                      model=MyEmbeddedDocument, parent_doc=MyDocument)
```

In order to facilitate that, pyrocumulus has a `pyrocumulus.web.urlmappers.DocumentURLMapper`, which creates all URLSpecs to a Document, including its EmbeddedDocuments.

2.1.3 DocumentURLMapper

The `pyrocumulus.web.urlmappers.DocumentURLMapper` class gets a Document and a RequestHandler in its constructor. The instance of `pyrocumulus.web.urlmappers.DocumentURLMapper` then creates all needed urls to your api works.

```
>>> from pyrocumulus.web.urlmappers import DocumentURLMapper
>>> from pyrocumulus.web.handlers import RestHandler
>>> from myapp.models import MyDocument
>>> mapper = DocumentURLMapper(MyDocument, RestHandler)
>>> urls = mapper.urls
```

You can use the optional arguments `url_prefix` and `url_name_prefix` to control how urls and urls names are created.

Once we are done with our urls, we need to create a `tornado.web.Application` instance to be served by our tornado server. Like this:

```
>>> from tornado.web import Application
>>> app = Application(urls)
```

2.1.4 RestApplication

In the pyrocumulus tutorial we used the `pyrocumulus.web.application.RestApplication` class to create our main application. Its a subclass from *tornado.web.Application* that receives a list of models and creates url mappings for it. It receives an optional argument `prefix` to be used as `url_prefix` and `url_name_prefix` on DocumentURLMapper.

```
>>> from pyrocumulus.web.applications import RestApplication
>>> app = RestApplication(MyDocument, prefix='/myapp')
```

This is how the restapi was created automaticlly. But one thing still needs to be explanined. How the tornado server runs our application. This is done by the `pyrocumulus.commands.runtornado.RunTornadoCommand` command.

2.1.5 runtornado command

The *runtornado* command is invoked in the command line as:

```
$ python pyromanager.py runtornado
```

Its default behavrior is to take the application to run from `pyrocumulus.web.applications.get_main_application()`, but you can use the parameter –application to specify an application to be used.

That's it.

2.2 Mongoengine-motor integration

Motor is a mongodb async driver that runs on the tornado main loop. I'm still working on making mongoengine works with that.

You can use this functionality using the code from gitorious.

This integration is not fully tested yiet and no documented at all. Take a look in the tests under tests/mongoengine to have an idea on how it works.

CHAPTER 3

Development

3.1 What's comming?

- refactor on pyrocumulus.web.applications and pyrocumulus.web.handlers - it stinks
- improve EmbeddedDocument inserts and updates
- implement auth on rest api
- better commands help
- refactor functional tests
- improve url configuration
- make mongoengine work with motor (can it be done?)
- refactor converters

and ALWAYS improve docs. :)

3.2 Changelog

- 0.4.5
  - adding fake join - something like ref__field
- 0.4.4
  - Correcting RestHandler behavior when using converters
- 0.4.3
  - Correcting file descriptors handling on runtornado.
  - Module pyrocumulus.web.request_handlers now is called pyrocumulus.web.handlers
  - Adding StaticApplication
  - Adding TemplateHandler
- 0.4.2.1
  - Corrected bug with tornado server daemonized on mac os x

- 0.4.2
  - Correcting bug with objects ordering
- 0.4.1
  - Adding ordering on objects list
- 0.4
  - Refactor on pyrocumulus.web.applications interface
  - Added test command
- 0.3.2.1
  - Correcting import that was causing errors on unittest
- 0.3.2
  - basic template support based on settings variable
- 0.3.1
  - basic cors support
- 0.3
  - docs updated
  - better commands help
  - added get_allowed_options to RestHandler
  - createproject command improved
- 0.3-b3
  - Change on DocumentURLMapper to receive a request handler as a arg to its constructor
- 0.3-b2
  - pyrocumulus.tornadoweb module now is pyrocumulus.web package
- 0.3-b1
  - added RestApplication to easy url configuration
  - Added support to ListField and EmbeddedDocument on RestHandler
  - change to make runtornado command get tornado port from settings
  - change PYROCUMULUS_SETTINGS_MODULE env var on base command
  - minor fixes
- 0.2
  - File descriptors close corrected on RunTornadoCommand
  - Changing commands from setupcommands module to commands package with a new options configuration scheme
  - Added pyromanager.py to run the commands from pyrocumulus.commands package
  - TornadoApplicationTestCase changed to use pyromanager.py
  - Added pyrocumulus script
  - Changed BaseRestHandler to return a dict on _list_object and to accept json_extra_params

  - Corrected treatment for lists on BaseConverter.sanitize_dict
- 0.1.1
  - Correction on setup.py
- 0.1 - First release

3.3 Contributors

Tcha-tcho <https://github.com/tcha-tcho/>

CHAPTER 4

Reference

4.1 pyrocumulus

4.1.1 pyrocumulus Package

pyrocumulus Package

conf Module

class pyrocumulus.conf.**Settings**
Bases: builtins.object

pyrocumulus.conf.**get_settings_module**(*module_name=None*)
Returns the module to be used as the settings for this project. If *module_name* is None (default) get_settings_module_name is used.

pyrocumulus.conf.**get_settings_module_name**()
Returns the module's name to be used as the settings module for this project. It can be changed by the environment variable *PYROCYMULUS_SETTINGS_MODULE*.

Defaults to 'settings'.

converters Module

db Module

exceptions Module

exception pyrocumulus.exceptions.**PyrocumulusCommandNotFound**
Bases: pyrocumulus.exceptions.PyrocumulusException

exception pyrocumulus.exceptions.**PyrocumulusConfusionError**
Bases: pyrocumulus.exceptions.PyrocumulusException

exception pyrocumulus.exceptions.**PyrocumulusConverterException**
Bases: pyrocumulus.exceptions.PyrocumulusException

exception pyrocumulus.exceptions.**PyrocumulusException**
Bases: builtins.Exception

exception pyrocumulus.exceptions.**StaticFileError**
Bases: pyrocumulus.exceptions.PyrocumulusException

exception pyrocumulus.exceptions.**TemplateNotFound**
Bases: pyrocumulus.exceptions.PyrocumulusException

parsers Module

class pyrocumulus.parsers.**DocumentParser**(*model*)
Bases: builtins.object

Parse a Document and return a dict with {'field_name': field_type}. ReferenceFields, EmbeddedDocumentFields and ListFields are returned in repareted keys.

parse()

pyrocumulus.parsers.**get_parser**(*model*)

testing Module

class pyrocumulus.testing.**TornadoApplicationTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Base class for tests with tornado. setUpClass() runs a tornado server in background and tearDownClass kills it.

application = None

python_exec = 'python'

runtornado_command = 'pyromanager.py runtornado'

classmethod setUpClass()

classmethod tearDownClass()

utils Module

pyrocumulus.utils.**fqualname**(*obj*)
Returns the full qualified name for an object, like pyrocumulus.converters.DocumentConverter

pyrocumulus.utils.**get_value_from_settings**(*key*, *default=None*)

Subpackages

commands Package

commands Package

base Module

class pyrocumulus.commands.base.**BaseCommand**

Bases: builtins.object

Base class for all commands. Your subclass must implement run().

If your command accept command line arguments, just add they to the user_options list. Each command in this list is a dictionary, containing two keys: 'args', and 'kwargs'. Like this:

[{'args': ('opt1',) 'kwargs': {'default': True}}, ...]

'args' and 'kwargs' are arguments passed to ArgumentParser.add_argument from argparse module. For details take a look here

http://docs.python.org/3.3/howto/argparse.html#id1

and here

http://docs.python.org/3.3/library/argparse.html#module-argparse

add_args()

Add the arguments in options

default_options = [{'kwargs': {'help': 'settings module to use', 'nargs': '?', 'default': None}, 'args': ('–settings',)}]

description = None

get_command_name()

parse_args()

Parse the args from the command line

run()

Execute your action. Your command must implement this

setenv()

Set the env variable PYROCUMULUS_SETTINGS_MODULE using the value passed on –setting option

user_options = []

pyrocumulus.commands.base.**get_command**(*name*)

Retuns a command based on *name*. The name of the command will be the file name, without the extension. So, if we have a mycommand.py file, the command name will be mycommand.

pyrocumulus.commands.base.**get_command_name**()

Returns the pyrocumulus command name based on the command line

pyrocumulus.commands.base.**list_commands**()

pyrocumulus.commands.base.**run_command**()

runs a pyrocumulus command based on the command line

pyrocumulus.commands.base.**show_help_message**()

Print a help message about command on screen and exit.

createproject Module

class pyrocumulus.commands.createproject.**CreateProjectCommand**

Bases: pyrocumulus.commands.base.BaseCommand

run()

user_options = [{'kwargs': {'help': 'Path for this project'}, 'args': ('path',)}, {'kwargs': {'help': "Project's name", 'n

runtornado Module

test Module

class pyrocumulus.commands.test.**TestCommand**

Bases: pyrocumulus.commands.base.BaseCommand

description = 'run unit tests WITHOUT in-place build'

run()

user_options = [{'kwargs': {'help': "Test suite to run (e.g. 'some_module.test_suite')", 'nargs': '?'}, 'args': ('--test-sui

web Package

applications Module

handlers Module

request_handlers Module

template Module

urlmappers Module

4.2 Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

p