Django Test Addons Documentation

Release 0.3.6

Hakampreet Singh Pandher

March 08, 2016

Contents

Django test addons provides support for testing different databases along with Django Web Framework. By default, django provides support for relational databases only. Since no-sql database systems are being widely used in django community, testing support for them is vital. As of now, django test addons provides testing support for Mongodb, Redis, Neo4j, Memcache, Django Rest Framework APIs only. Support for more databases might be provided in future.

CHAPTER 1

Installation

```
pip install django-test-addons
```

CHAPTER 2

Requirements

Django test addons requires the following:

1. Python(2.7+)
2. Django(1.6, 1.7, 1.8, 1.9)

The following packages are optional:

- Mongoengine (0.8.7)+ - Testing support for Mongo DB.
- Django Redis (3.8.2)+ - Testing support Redis.
- Py2neo (2.0.6)+ - Testing support for Neo4j graph database.
- Python Memcached (1.53)+ - Testing support for Memcache.
- Django Rest Framework (3.0.5)+ - Testing support for Django Rest Framework Apis

Note: Package may work perfectly for older versions than specified. It's just that it is not tested with them. So feel free to give it a try.

CHAPTER 3

User Guide

3.1 Installation

```
pip install django-test-addons
```

3.2 Requirements

Django test addons requires the following:

1. Python(2.7+)
2. Django(1.6, 1.7, 1.8, 1.9)

The following packages are optional:

- Mongoengine (0.8.7)+ - Testing support for Mongo DB.
- Django Redis (3.8.2)+ - Testing support Redis.
- Py2neo (2.0.6)+ - Testing support for Neo4j graph database.
- Python Memcached (1.53)+ - Testing support for Memcache.
- Django Rest Framework (3.0.5)+ - Testing support for Django Rest Framework Apis

Note: Package may work perfectly for older versions than specified. It's just that it is not tested with them. So feel free to give it a try.

3.3 Tutorial

This tutorial provides a step-by-step description on how to use django test addons for testing different database systems.

3.3.1 Getting Started

It is recommended to have local installation of respective databases just for testing. Staging or shared database or any database with critical data should never be used in testing, as database is cleaned after each test is ran. It is recommended to use a separate settings file for testing.

Warning: **Be Careful** to use correct settings for test databases. Using staging or any other database may result in cleaning of the entire database.

If you haven't installed django test addons already, use

```
pip install django-test-addons
```

3.3.2 Testing Mongodb

Defining test settings

Make sure you have running installation of mongodb and have mongoengine installed. Just specify the settings for connection to mongodb instance in the settings file. Define *TEST_MONGO_DATABASE* dict in your test file containing connection information.

Example:

Add this code to test settings file -

```
TEST_MONGO_DATABASE = {
    'db': 'test',
    'host': ['localhost'],
    'port': 27017,
}
```

Make sure to use same test database for all mongo database aliases. To clarify, say you have following mongo connection settings in your development/production settings containing two mongodb aliases.

```
MONGO_DATABASES = {
    'default': {
        'db': 'main',
        'host': ['193.34.32.11'], # random development server
        'port': 27017,
    },
    'miscellaneous': {
        'DB_NAME': 'misc',
        'HOST': ['193.34.32.11'],
        'PORT': 27017,
    }
}
```

In your test settings, make sure to disconnect all existing connections and connect all mongodb aliases to test db.

```
# import MONGO_DATABASES variable from development settings file or just use the
# variable if you are using single file for testing with some environment settings.

import mongoengine

TEST_MONGO_DATABASE = {
    'db': 'test',
    'host': ['localhost'],
    'port': 27017,
}

map(lambda connection: mongoengine.connection.disconnect(connection), MONGO_DATABASES.keys())

MONGO_DATABASES = {connection: TEST_MONGO_DATABASE for connection in MONGO_DATABASES.keys()}
```

```
for connection_name, attrs in MONGO_DATABASES.items():
    mongoengine.connect(**dict(zip(['alias'] + attrs.keys(), [connection_name] + attrs.values())))
```

Writing Tests

Just import *MongoTestCase* from test_addons, and inherit test class from it.

Example

```
import test_addons

class TestSomething(test_addons.MongoTestCase):

    def test_instantiation(self):
        pass
```

3.3.3 Testing Memcache

Just specify *CLEAR_CACHE=TRUE* in your test class, if you want to clear cache too(it could be Memcache or Redis or any other caching framework that works with django). You must have CACHES configured in your test settings for this to work.

Example

```
import test_addons

class TestSomething(test_addons.MongoTestCase):

    CLEAR_CACHE = True

    def test_instantiation(self):
        pass
```

3.3.4 Testing Redis

Defining test settings

Make sure you have redis db installed and a running redis server. Just specify *TEST_CACHES* dictionary in your test settings containing redis connection info.

Example:

```
TEST_CACHES = {
    'default': {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "127.0.0.1:6379:0",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    'redis1': {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "127.0.0.1:6379:1",
        "OPTIONS": {
```

```
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
}
```

Note: 'django_redis.cache.ShardClient' does not allow flushing all db as of now, so make sure not to use it. Sharding is not required in testing environment anyway.

Writing Tests

Just import *RedisTestCase* from test_addons, and inherit test class from it.

Example

```
import test_addons

class TestSomething(test_addons.RedisTestCase):

    def test_instantiation(self):
        pass
```

3.3.5 Testing Neo4j Graph database

Defining test settings

Make sure you have neo4j graph installed and a running neo4j server. Just specify *NEO4J_TEST_LINK* pointing to ip address of running neo4j server in your test settings file.

Example

```
NEO4J_TEST_LINK = 'http://localhost:7474/db/data'
```

Note: Since neo4j 2.0, it requires authentication to connection to your neo4j server. Considering it is unnecessary for testing environment, make sure to set 'dbms.security.auth_enabled=false' in your neo4j-server.properties file

Writing Tests

Just import *Neo4jTestCase* from test_addons, and inherit test class from it.

Example

```
import test_addons

class TestSomething(test_addons.Neo4jTestCase):

    def test_instantiation(self):
        pass
```

3.3.6 Testing Django Rest Framework APIs

It provides support for testing Django rest framework api's along with one or more databases.

Note: Test cases described above would have worked for apis as well, but they use default Test Client provided by Django, whereas it uses Test Client provided by DRF having some additional facilities like forcing authentication.

Writing Tests

Just import APITestCase for the specific database you are using (specify settings accordingly).

Available options are:

- APIRedisTestCase
- APIMongoTestCase
- APINeo4jTestCase
- APIMongoRedisTestCase
- APIRedisMongoNeo4jTestCase

Example Say we want to use test DRF apis along with mongodb.

```
import test_addons

class TestSomething(test_addons.APIMongoTestCase):

    def test_instantiation(self):
        pass
```

3.3.7 Composite Testing

Often multiple databases are used simulataneously, thereby creating the need of testing them simulataneously. Just to cater this need, django test addons provide different combinations of TestCases for respective database combinations.

Composite Test Cases:

- MongoNeo4jTestCase
- MongoRedisTestCase
- RedisMongoNeo4jTestCase
- APIRedisTestCase
- APIMongoTestCase
- APINeo4jTestCase
- APIMongoRedisTestCase
- APIRedisMongoNeo4jTestCase

3.3.8 Facing Issues

Make sure you have defined settings exactly as mentioned. If you still can't resolve the issue, you can use Django test addons mailing list or raise an issue on github or just mail me directly at *hspandher@outlook.com*

3.4 Changelog

3.4.1 Changes in version 1.0

- Support for Django 1.9 along with Python 3

3.4.2 Changes in version 0.3.6

- Updated pypi download url to the latest version (Minor update)

3.4.3 Changes in 0.3.5

- Fix APIClient bug. It was not working due to incorrect name error

3.5 Community

To get help with using MongoEngine, use the Django test addons mailing list , raise an issue on github or just mail me directly at *hspandher@outlook.com*.

3.6 Contributing

Yes please! I am always looking for contributions, additions and improvements. Support for testing more databases is specifically required.

The source is available on GitHub and contributions are always encouraged. Contributions can be as simple as minor tweaks to this documentation, the website or the core.

To contribute, fork the project on GitHub and send a pull request.

3.7 License

3.8 Indices and tables

- genindex
- search

CHAPTER 4

Changelog

4.1 Changes in version 1.0

- Support for Django 1.9 along with Python 3

4.2 Changes in version 0.3.6

- Updated pypi download url to the latest version (Minor update)

4.3 Changes in version 0.3.5

- Fix APIClient bug. It was not working due to incorrect name error (use of self instead of cls)

4.4 Community

To get help with using MongoEngine, use the Django test addons mailing list , raise an issue on github or just mail me directly at *hspandher@outlook.com*.

4.5 Contributing

Yes please! I am always looking for contributions, additions and improvements. Support for testing more databases is specifically required.

The source is available on GitHub and contributions are always encouraged. Contributions can be as simple as minor tweaks to this documentation, the website or the core.

To contribute, fork the project on GitHub and send a pull request.

4.6 License

4.7 Indices and tables

- genindex
- search