Django Test Query Count Documentation

Release 0.1.0

Ignacio Avas

Oct 26, 2017

Contents

Contents:

CHAPTER 1

Django Test Query Counter

logo.png

A Django Toolkit for controlling Query count when testing. It controls the number of queries done in the tests stays below a particular threshold between Test Runs.Specially useful when paired with a CI like Jenkins or Travis to control each commit doesn't slow down the Database considerably.

Requirements

- Python 3
- Django

Documentation

The full documentation is at https://django-test-query-counter.readthedocs.io.

Installation

There are ways to install it into your project

Clone this repository into your project:

```
git clone https://github.com/sophilabs/django-test-query-counter.git
```

Download the zip file and unpack it:

```
wget https://github.com/sophilabs/django-test-query-counter/archive/master.zip
unzip master.zip
```

Install with pip:

```
pip install django-test-query-counter
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'test_query_count',
    ...
)
```

Usage

Run your django tests as you do. After the run the `reports` directory with two files `query_count.json` and `query_count_detail.json`.

To check your tests Query Counts run:

```
$ python manage.py check_query_count
```

Then you would see an output like the following one (if you at least ran the tests twice):

If the current test run had more queries than the last one, it will tell you:

Configuration

You can customize how the Test Query Count works by defining `TEST_QUERY_COUNTER` in your settings file as a dictionary. The following code shows an example

```
TEST_QUERY_COUNTER = {
    # Global switch
    'ENABLE': True,

    # Paths where the count files are generated (relative to the current
    # process dir)
    'DETAIL_PATH': 'reports/query_count_detail.json',
    'SUMMARY_PATH': 'reports/query_count.json',

    # Tolerated percentage of count increase on successive
    # test runs.A value of 0 prevents increasing queries altoghether.
    'INCREASE_THRESHOLD': 10
}
```

Excluding Tests

Individual tests or classes can be excluded for the count using the `@exclude_query_count` decorator. For example

```
# To exclude all methods in the class
@exclude_query_count()
class AllTestExcluded(TestCase):
    def test_foo(self):
        self.client.get('path-1')

    def test_foo(self):
        self.client.get('path-2')

# To exclude one test only
class OneMethodExcluded():
    def test_foo(self):
        self.client.get('path-1')

    @exclude_query_count()
    def test_foo(self):
        self.client.get('path-2')
```

More specifically, `exclude_query_count` accept parameters to conditionally exclude a query count by path, method or count. Where `path` the or regex of the excluded path(s). The `method` specifies the regex of the method(s) to exclude, and `count` is minimum number of queries tolerated. Requests with less or same amount as "count" will be excluded.

For example:

```
class Test(TestCase):
    @exclude_query_count(path='url-2')
    def test_exclude_path(self):
        self.client.get('/url-1')
        self.client.post('/url-2')

    @exclude_query_count(method='post')
    def test_exclude_method(self):
        self.client.get('/url-1')
        self.client.post('/url-2')

    @exclude_query_count(count=2)
    def test_exclude_count(self):
        #  succesive urls requests are additive
        for i in range(3):
            self.client.get('/url-1')
```

Implementing into your CI

Currently the query count works locally. However, it shines when it is integrated with Jenkins, or other CIs. You have to do this manually:

Requirements

- Jenkins with a Job that build you project.

From now on let's suppose your job is available at `http://127.0.0.1:8080/job/ZAP_EXAMPLE_JOB/`.

1. *Activate Build Archive*: Go to the job configuration page and add the archive artifacts build Post-build actions.

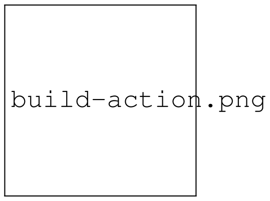
   archive-artifact.png

2. Set `reports/query_count.json` in the files to archive path

3. Create a new Django custom Command to run the validation against the archived Jenkins file. For example:

```
from urllib.request import urlretrieve
from django.core.management import BaseCommand, CommandError
from django.conf import settings
from test_query_counter.apps import RequestQueryCountConfig
from test_query_counter.query_count import QueryCountEvaluator


class Command(BaseCommand):
    JENKINS_QUERY_COUNT = 'https://yourci/job/' \
                          'yourproject/lastSuccessfulBuild/' \
                          'artifact/app/reports/query_count.json'

    def handle(self, *args, **options):
        current_file_path = RequestQueryCountConfig.get_setting('SUMMARY_
↪FILE')

        with open(current_file_path) as current_file, \
                open(urlretrieve(self.JENKINS_QUERY_COUNT)[0]) as last_
↪file:
            violations = QueryCountEvaluator(10, current_file,last_file).
↪run()

            if violations:
                raise CommandError('There was at least one test with an␣
↪API '
                                   'call excedding the allowed threshold.
↪')
```

4. Add a build step to run this command:



After that, it will run fine, and the build would fail if any Query count is over the limit.

TODO

- Include support for stacktraces in `query_count_detail.json`.
- Generate an HTML report of executed Queries.
- Make query count configurable
- Include Jenkins support out of the box (using *django_jenkins*)

Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

License

Django Test Query Counter is MIT Licensed. Copyright (c) 2017 Sophilabs, Inc.

Credits

This tool is maintained and funded by Sophilabs, Inc. The names and logos for sophilabs are trademarks of sophilabs, inc.

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-test-query-counter
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-test-query-counter
$ pip install django-test-query-counter
```

CHAPTER 3

Usage

To use Django API Query Count in a project, add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'test_query_counter.apps.DjangoApiQueryCountConfig',
    ...
)
```

Add Django API Query Count's URL patterns:

```
from test_query_counter import urls as test_query_counter_urls


urlpatterns = [
    ...
    url(r'^', include(test_query_counter_urls)),
    ...
]
```

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/sophilabs/django-test-query-counter/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 API Query Count could always use more documentation, whether as part of the official Django API Query Count 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/sophilabs/django-test-query-counter/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-test-query-counter* for local development.

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

   ```
   $ git clone git@github.com:your_name_here/django-test-query-counter.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-test-query-counter
   $ cd django-test-query-counter/
   $ 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 test_query_counter 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/sophilabs/django-test-query-counter/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python runtests.py tests.test_test_query_counter
```

CHAPTER 5

Credits

Development Lead

- Ignacio Avas <iavas@sophilabs.com>

Contributors

None yet. Why not be the first?

CHAPTER 6

History

0.1.0 (2017-07-10)

- First release on PyPI.