Testing your Django App

Ratnadeep Debnath
rtnpro@indifex.com

Indifex

Why test?

- Check if your app behaves as it is supposed to
- Refactoring code does not break existing app's behaviour

Django test frameworks

- Doctests
- Unit tests

Doctests

Tests that are embedded in your functions' docstrings and are written in a way that emulates a session of the Python interactive interpreter

```
def my_func(a_list, idx):
    """
    >>> a = ['larry', 'curly', 'moe']
    >>> my_func(a, 0)
    'larry'
    >>> my_func(a, 1)
    'curly'
    """
    return a_list[idx]
```

Unit tests

Unit testing is a method by which individual units of source code are tested to determine if they are fit for use

```
import unittest

class MyFuncTestCase(unittest.TestCase):
    def testBasic(self):
        a = ['larry', 'curly', 'moe']
        self.assertEqual(my_func(a, 0), 'larry')
        self.assertEqual(my_func(a, 1), 'curly')
```

Benefits of Unit tests

- Facilitates change
- Simplifies Integration
- Documentation
- Design

Writing unit tests

- Django's unit tests use a Python standard library module: unittest. This module defines tests in class-based approach.

```
from django.utils import unittest
```

- Django test runner looks for unit tests in two places:
  - models.py
  - tests.py, in the same directory as models.py

  The test runner looks for any subclass of unittest.TestCase in this module.

Writing unit tests

```
from django.test import TestCase
from django.test.client import Client
from posts.models import Post

class TestPosts(TestCase):
    def setUp(self):
        self.post = Post.objects.create(title='foo', body='foo')

    def test_views(self):
        self.assertEqual(self.post.title, 'foo')
        self.assertEqual(self.post.body, 'foo')
```

Running tests

- $ ./manage.py test
- $ ./manage.py test posts
- $ ./manage.py test posts.TestPosts
- $ ./manage.py test posts.TestPosts.test_views

The Test Client

- A python class that acts as a dummy web browser
- Allows testing your views
- Interact with your Django app programatically

Let's run tests

The Test Client

- A way to build tests of the full stack
- Acts more or less like a browser
- Stateful
- Default instance on django.test.TestCase
- Can be instantiated outside test framework

The test client isn't

- A live browser test framework
  - Selenium
  - Twill
  - Windmill

Why is it different?

- TestClient can't do some things
  - No JavaScript
  - DOM validation or control
- Can do some things that browsers can't

Returns a response

- response.status_code
- response.content
- response.cookies
- response['Content-Disposition']

... and a little more

- response.template
  - The template(s) used in rendering
- response.context
  - The context objects used in rendering
  - response.context['foo']

...and it maintains state

- self.cookies
- self.session

Login/Logout

```
from django.test.client import Client
c = Client()
c.login(username='foo',
        Password='password')

c.logout()
```

Get a page

```
from django.test.client import Client

c = Client()

c.get('/foo/')

c.get('/foo/?page=bar')

c.get('/foo/', {'page': 'bar'})
```

Post a page

```
from django.test.client import Client

c = Client()

c.post('/foo/')

c.post('/foo/?bar=3')

c.post('/foo/', data={'bar': 3})

c.get('/foo/?whiz=4', data={'bar':3})
```

Rest of HTTP

```
c.put('/foo/')

c.options('/foo/')

c.head('/foo/')

c.delete('/foo/')
```

A common problem

```
r = self.client.get('/foo')
self.assertEquals(r.status_code, 200)
```

FAIL Assertion Error: 301 != 200

The fix

```
r = self.client.get('/foo', follow=True)

self.assertEquals(r.status_code, 200)

response.redirect_chain
   -- links visited before a non-redirect was found
```

You can do more

- Extra Headers

```
c = TestClient(HTTP_HOST='foo.com')
c.get('/foo/', HTTP_HOST='foo.com')
```

- Files

```
f = open('text.txt')
c.post('/foo/',
        content_type=MULTIPART_CONTENT),
        data = {'file': f})
f.close()
```

Assertions

- assertContains()
- assertNotContains()
- assertFormError()
- assertTemplateUser()
- assertTemplateNotUsed()
- assertRedirects()

Do's and Don'ts

- Don't rely on assertContains
  - Assertions on template content are weak
- Test at the source
  - Is the context right?
  - Are the forms correct?
- Django's templates make this possible!

When to use TestClient

- When you need to test the full stack
  - Interaction of view and middleware
- Great for testing idempotency

Let's run some tests again

Coverage

- Why do I need coverage tests?
  An small example:
  - You write some new code :-)
  - Does it really work as expected? :-(
  - Now, you write tests for it :-)
  - Does your test really test your code? :-(

  This is where coverage comes to the rescue :D

What is coverage

- A tool for measuring code coverage of Python programs.
- It monitors your program, noting which parts of the code have been executed,
- It then analyzes the source to identify code that could have been executed but was not.
- Coverage measurement is typically used to gauge the effectiveness of tests. It can show which parts of your code are being exercised by tests, and which are not.

Who is behind Coverage

Ned Batchelder

Indifex

Install Coverage

- easy_install coverage
- pip install coverage

How to use Coverage?

- Use coverage to run your program and gather data
  - $ coverage -x my_program.py arg1 arg2
- Use coverage to report on the results

  $ coverage -rm

```
Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
my_program              20      4    80%   33-35, 39
my_other_module         56      6    89%   17-23
--------------------------------------------------
TOTAL                   76     10    87%
```

How to use Coverage

- Generate better reports for presentation, say html:
  - coverage html -d <dir>
- Erase existing coverage data
  - coverage -e

Let's use Coverage

Questions?

Indifex

Useful links

- https://docs.djangoproject.com/en/dev/topics/testing/
- http://nedbatchelder.com/code/coverage/

Contact

Ratnadeep Debnath
Email: rtnpro@indifex.com
IRC: rtnpro
Blog: ratnadeepdebnath.wordpress.com

Thank You :)

Indifex