Testing Django applications with py.test

Andreas Pelme
andreas@pelme.se
@andreaspelme
Topics

• Testing in Django?
• py.test
• Django and py.test
• Django testing tips & tricks
Testing in Django?
Django’s testing documentation
“If you’ve been using Python for a while, doctest will probably feel more “pythonic”.”

- Django <= 1.5 documentation
Django’s testing documentation
Django’s test runner
It is getting better!

Photo: http://www.flickr.com/photos/bee/3290452839/
tox  Twisted Trial

Other testing tools?

nose

zope.testrunner
py.test

a full-featured Python testing tool
Plugins

- Distributed/parallelized (pytest-xdist)
- Django (pytest-django)
- Log capture (pytest-capturelog)
- Coverage reporting (pytest-cov)
- ... and many more
Pythonic tests

without boilerplate
from django.test import TestCase

class TestHelloWorld(TestCase):
    def test_hello_world(self):
        response = self.client.get('/hi/)

        self.assertEqual(response.status_code, 200)
        self.assertEqual(response.content, 'Hello World!')
def test_hello_world(client):
    response = client.get('/hi/')

    assert response.status_code == 200
    assert response.content == 'Hello World!'
def test_hello_world(client):
    response = client.get('hi/)

    assert response.status_code == 200
    assert response.content == 'Hello World!'
def test_hello_world(client):
    response = client.get('/hi/')
    assert response.status_code == 200
    assert response.content == 'Hello World!'
E   assert 'Hello EuroPython!' == 'Hello World!
E   - Hello EuroPython!
E   + Hello Hello World!

test_hello_world.py:5: AssertionError
assertAlmostEqual
assertAlmostEquals
assertDictContainsSubset
assertDictEqual
assertEqual
assertEquals
assertFalse
assertGreater
assertGreaterEqual
assertIn
assertIs
assertIsInstance
assertIsNone
assertIsNot
assertIsNotNone
assertItemsEqual
assertLess
assertLessEqual

assertListEqual
assertMultiLineEqual
assertNotAlmostEqual
assertNotAlmostEquals
assertNotEqual
assertNotEquals
assertNotIn
assertNotIsInstance
assertNotRegexpMatches
assertRaises
assertRaisesRegexp
assertRegexpMatches
assertSequenceEqual
assertSetEqual
assertTupleEqual

fredag 5 juli 13
pytest-django

Integration between py.test and Django
pip install pytest-django

Installs pytest-django and pytest
pip install -e .

Adds your project to Python path (requires a setup.py to be present)
export DJANGO_SETTINGS_MODULE=settings

Define Django settings in your shell environment
Bonus:
Launch a Python shell without manage.py

Photo: http://www.flickr.com/photos/mark_i_geo/3498456758
[pytest]
DJANGO_SETTINGS_MODULE=settings

Can also be configured in pytest.ini
How to run the tests?
$ py.test

Run all the tests!
$ py.test test_views.py

Run all tests in a specific file
$ py.test -k test_foo
Run all test cases that are named *test_foo*
Where to put the tests?
By default, tests are found in `test_*.py`
Configurable in pytest.ini:

[pytest]
python_files = *.py
polls/tests/views.py
blog/tests/views.py

Tests can live under Django apps
Tests can live in their own root directory
unit_tests/blog/models.py
functional_tests/test_foo.py
Tests be split up into different root directories
Django TestCases just works
py.test --reuse-db
Reuse database from last run
[pytest]
addopts = --reuse-db

Put this in pytest.ini to always reuse the database
Pass `--create-db` to force re-creation
Testable Django code
“There is no secret to writing tests... there are only secrets to writing testable code!”

- Miško Hevery
models.py
views.py
models.py
views.py
admin.py
forms.py
Testing views

Keep them small (few branches)
from datetime import datetime, time
from django.http import HttpResponse

def greet(request):
    now = datetime.now().time()

    if time(5) < now < time(12):
        greeting = 'morning'
    elif time(18) < now < time(21):
        greeting = 'evening'
    else:
        greeting = 'day'

    return HttpResponse('Good %s!' % greeting)
from datetime import datetime, time
from django.http import HttpResponse

def greet(request):
    now = datetime.now().time()

    if time(5) < now < time(12):
        greeting = 'morning'
    elif time(18) < now < time(21):
        greeting = 'evening'
    else:
        greeting = 'day'

    return HttpResponse('Good %s!' % greeting)
# greeter.py

from datetime import time

def greeting_at(time_of_day):
    if time(5) < time_of_day < time(12):
        return 'morning'
    elif time(18) < time_of_day < time(21):
        return 'evening'
    else:
        return 'day'
# views.py
rom datetime import time
from django.http import HttpResponse
from .greeter import greeting_at

def greet(request):
    now = datetime.now().time()
    greeting = greeting_at(now)
    return HttpResponse('Good %s!' % greeting)
Database access

... makes your tests slow
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.
THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S testing"

HEY! GET BACK TO WORK!

OH. CARRY ON.
Mind your queries

The ORM blurs the line between application code and query code
Separate the query code into its own methods
U+1F4A9
[pk]  [master]  ~/code/personalkollen  $  py.test --poo
Re-using existing test database for alias 'default'...
pytest-django and databases
import pytest

@ pytest.mark.django_db
def test_user_count():
    assert User.objects.count() == 0

This will allow database access
def test_user_count():
    assert User.objects.count() == 0

This will fail, database access is not allowed!
test_user_count

test_db_access.py:4: in test_user_count
>    assert User.objects.count() == 0

... snip ...

E   Failed: Database access not allowed, use the "django_db" mark to enable
import pytest

pytestmark = pytest.mark.django_db

def test_foo(): pass
def test_bar(): pass

Mark an entire module for database usage
$ py.test -m 'not django_db'

Run all tests which does not touch the database
Model test data

- Create objects manually
- Django fixtures
- Factories
Django fixtures

Slow & hard to maintain.. avoid them!
The solution - factories

factory_boy is an excellent library
from django.db import models

class Group(models.Model):
    name = models.TextField()

class Person(models.Model):
    first_name = models.TextField()
    last_name = models.TextField()
    group = models.ForeignKey(Group)

    def group_letter(self):
        return self.group.name[0].upper()
import factory
from yourapp.models import Person, Group

class GroupFactory(factory.Factory):
    FACTORY_FOR = Group
    name = 'Developers'

class PersonFactory(factory.Factory):
    FACTORY_FOR = Person
    first_name = 'John'
    last_name = 'Doe'
    group = factory.SubFactory(GroupFactory)
from yourfactories import PersonFactory

def test_group_letter():
    person = PersonFactory.build(
        group__name='admins')

    assert person.group_letter() == 'A'
import pytest

@pytest.mark.django_db
def test_group_letter():
    person = PersonFactory.create(
        group__name='admins')

    assert person.group_letter() == 'A'
py.test fixtures

Replacement for unittest setUp/tearDown
def test_with_client(client):
    response = client.get('/foo/')
    assert response.status_code == 200

Get an instance of the test client
def test_foo(settings):
    settings.DATE_FORMAT = 'Y-m-d'

The settings will be automatically reset after the test
import pytest

@pytest.fixture
def person():
    return PersonFactory.build()
import pytest
from your_factories import UserFactory

@ pytest.fixture
def person_in_db(db):
    return PersonFactory.create()
Great resources

- Carl Meyer, Testing and Django, PyCon 2012:
  - Django testing best practices and inspiration for this talk
  - http://www.youtube.com/watch?v=ickNQcNXiS4

- Holger Krekel, pytest - rapid and simple testing with Python, EuroPython 2013
  - http://www.youtube.com/watch?v=k6Z2JHUUmZXMi

- pytest documentation:
  - http://pytest.org/

- pytest-django documentation:
  - http://pytest-django.readthedocs.org/
pytest sprint
Works with both worlds
Thank You!

Slides: http://speakerdeck.com/pelme

andreas@pelme.se
@andreaspelme