PyCon Namibia 2017

21st-23rd February

Windhoek

TESTING YOUR DJANGO APP

Anna Makarudze

@amakarudze

About me

- Python/Django Developer
- PyCon Zimbabwe organiser
- Django Girls Harare/Masvingo organiser
- PyLadies Harare organiser
- Twitter - @amakarudze

Overview

- An interactive tutorial on testing
- Introduction to testing in Django
- Introduction to Test-Driven Development (TDD)

Credits

- Ana Balica, about Testing, Django: Under The Hood 2016
- San Diego Python - Learning Django by Testing
- Django Documentation

Instructions

- Repo - github.com/amakarudze/pycon_na_2017/
- Installation and setting up a new project

Why write tests?

- Identify defects in your code
- Reduces bugs at run-time
- New code – validate your code works as expected
- Refactoring or modifying old code – ensure your changes haven’t affected your application’s behaviour unexpectedly

Why is testing complex?

- Several layers of logic make up a web application
- HTTP-level request handling
- Form validation and processing
- Template rendering (including static files)
- Models
- Sending emails

Tests in a Django project

- `python manage.py startapp` creates a `tests.py` in the new app.
- Works for a few tests

Tests in a Django project

- Larger test suite requires restructuring into a tests package
- Split your tests into different submodules, i.e.

  `test_models.py`
  `test_views.py`
  `test_forms.py` etc.

Running Tests in Django

```
$ ./manage.py test
```

or

```
$ python manage.py test
```

Running Tests in Django

 # Run all the tests in a module, e.g. `animals` *module containing*

`tests.py`*, i.e. the* `animals.tests` *module*

```
$ ./manage.py test animals.tests
```

Running Tests in Django

```
# Run all the tests found within the 'animals'
# package

$ ./manage.py test animals
```

Running Tests in Django

 # Run just one test case

```
$ ./manage.py test
animals.tests.AnimalTestCase
```

Running Tests in Django

 # Run just one test method

```
$ ./manage.py test
animals.tests.AnimalTestCase.test
_animals_can_speak
```

Running tests

 # Provide a path to a directory to discover tests
 # below that directory

```
$ ./manage.py test animals/
```

Running tests

 # Specify a custom filename pattern match using
 # the -p (or --pattern) option, for test files named
 # differently from the test.py pattern:*

```
$ ./manage.py test --
pattern="tests_*.py"
```

Tagging tests

```
class SampleTestCase(TestCase):
    @tag('slow')
    def test_slow(self):
        ...
```

```
./manage.py test --tag=slow
./manage.py test --exclude-tag=slow
```

Testing tools

- Test client – Client

  `django.test.Client`

- RequestFactory – limited version of Client

  `django.test.RequestFactory`

Client

- Python class that acts as a dummy Web browser
- Simulate `GET` and `POST` requests on a URL
- Observe the response
  - low-level `HTTP` (result headers and status codes),
  - page content,

Client

- Chain of redirects (if any),
  - check the URL
  - status code at each step.
- Request is rendered
  - by a given Django template,
  - a template context that contains certain values.

RequestFactory

- Uses the same API as test client
- Restricted subset of the test client API
- Only generate a request instance that can be used as the first argument to any view
- Does not act as a browser

RequestFactory

- Only has access to the `HTTP` methods `get()`, `post()`, `put()`, `delete()`, `head()`, `options()`, and `trace()`.
- All accept the same arguments except for `follows`.

RequestFactory

- Is just a factory for producing requests,
- It’s up to you to handle the response.

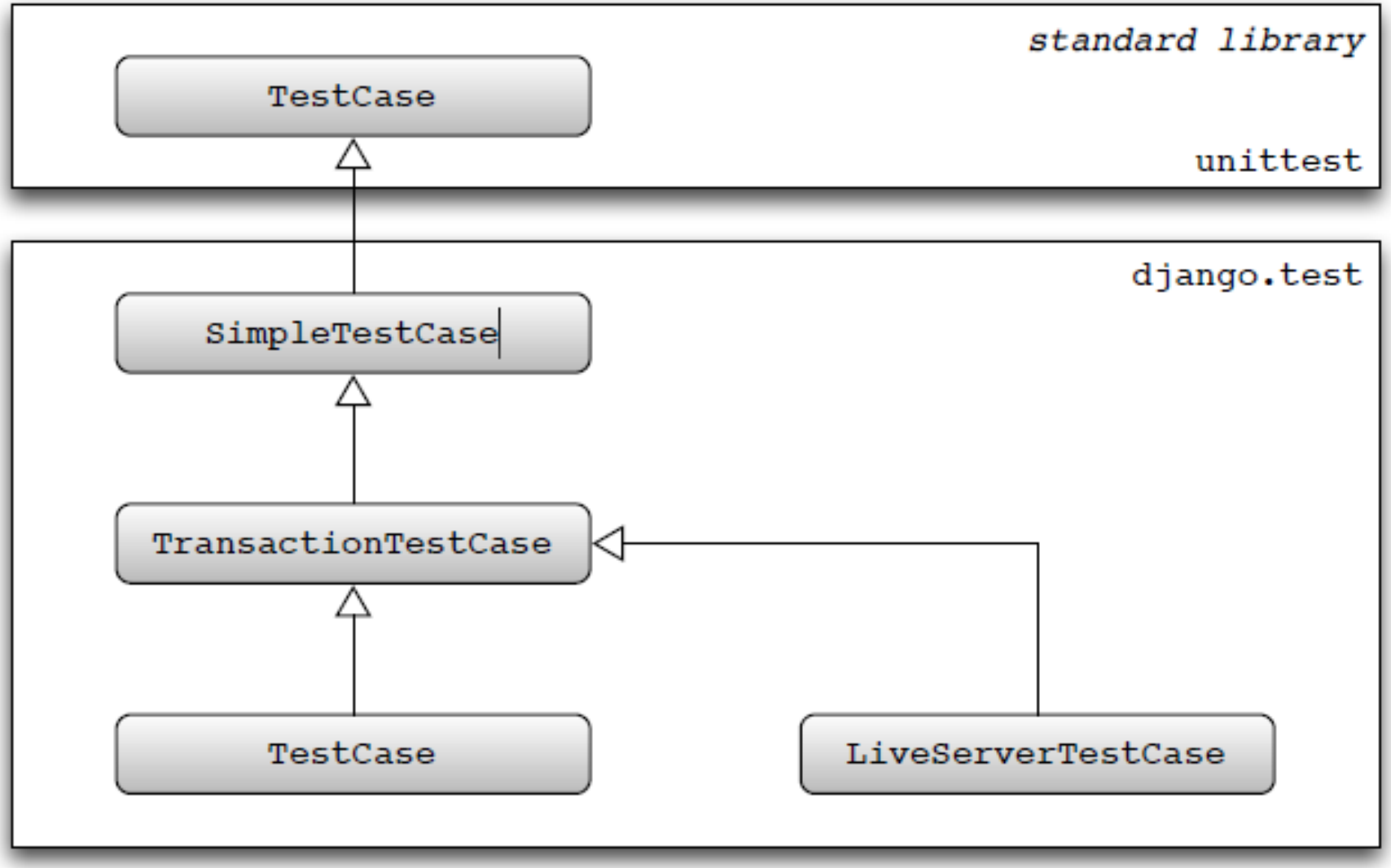
RequestFactory

- It does not support middleware.
- Session and authentication attributes must be supplied by the test itself if required for the view to function properly.

Provided test classes

- SimpleTestCase
- TransactionTestCase
- TestCase
- LiveServerTestCase
- StaticLiveServerTestCase

Hierarchy of Django unit testing classes



SimpleTestCase

- no database queries
- access to test client
- fast

TransactionTestCase

- allows database queries
- access to test client
- ~~fast~~
- allows database transactions
- flushes database after each test

TestCase

- allows database queries
- access to test client
- faster
- restricts database transactions
- runs each test in a transaction

LiveServerTestCase

- acts like TransactionTestCase
- launches a live HTTP server in a
- separate thread

StaticLiveServerTestCase

- acts like TransactionTestCase
- launches a live HTTP server in a separate thread
- <u>serves static files</u>

Order in which tests are executed

- To guarantee that all TestCase code starts with a clean database:
- TestCase subclasses are run first.
- Other Django-based tests (test cases based on SimpleTestCase, including TransactionTestCase).
- unittest.TestCase tests (including doctests).

unittest.TestCase vs django.test.TestCase

- Tests that require database access should subclass `django.test.TestCase`
- `unittest.TestCase` avoids running each test in a transaction and flushing the database

unittest.TestCase vs django.test.TestCase

- Behaviour of tests varies based on the order of execution by the test runner
- Result - unit tests that pass when run in isolation but fail when run in a suite.

Now, let’s write some tests in Django!

Activate a virtualenv

Windows

```
$ myvenv\scripts\activate
```

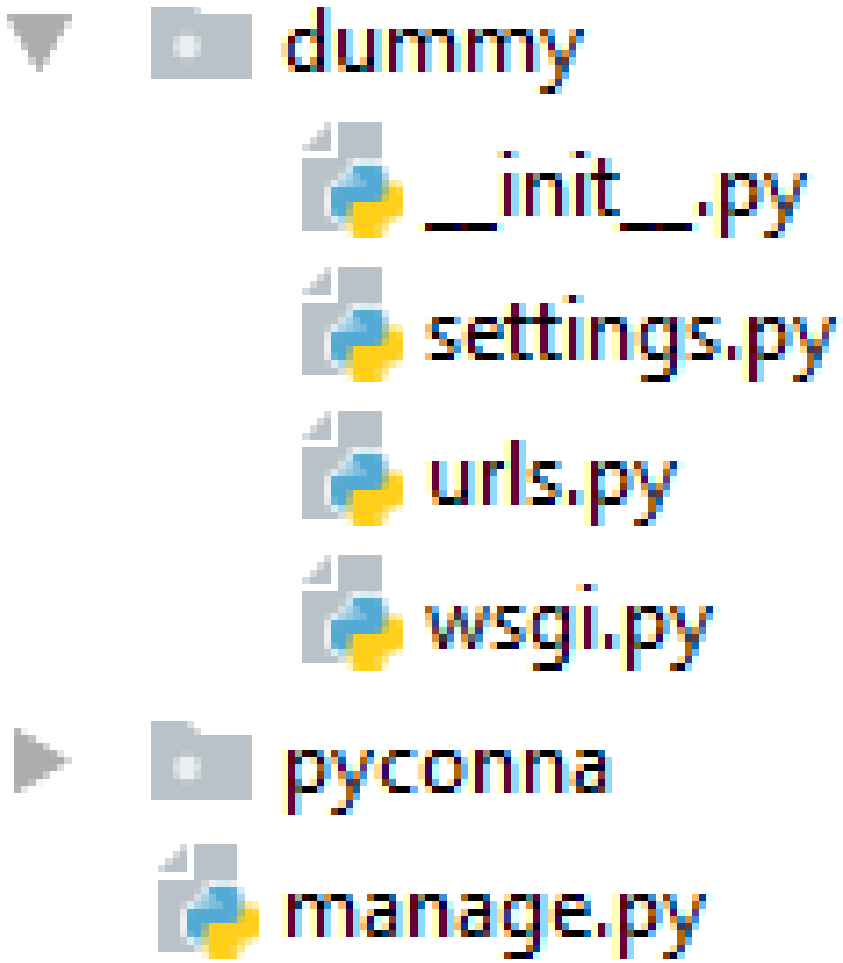
Linux/Mac

```
$ source/bin
```

Starting a Django project

```
$ django-admin startproject dummy
```

dummy project



Run our project

```
$ python manage.py runserver
```

Menu | Inbox (21) - amakarudze@ | (447) Twitter | Welcome to Django | pyladieshre (PyLadies Hara | Programme - PyCon Nami

127.0.0.1:8000

Some downloads were interrupted when the browser was closed. Would you like to resume those downloads now? Yes No Ask me later

It worked!

Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [app_label]`.

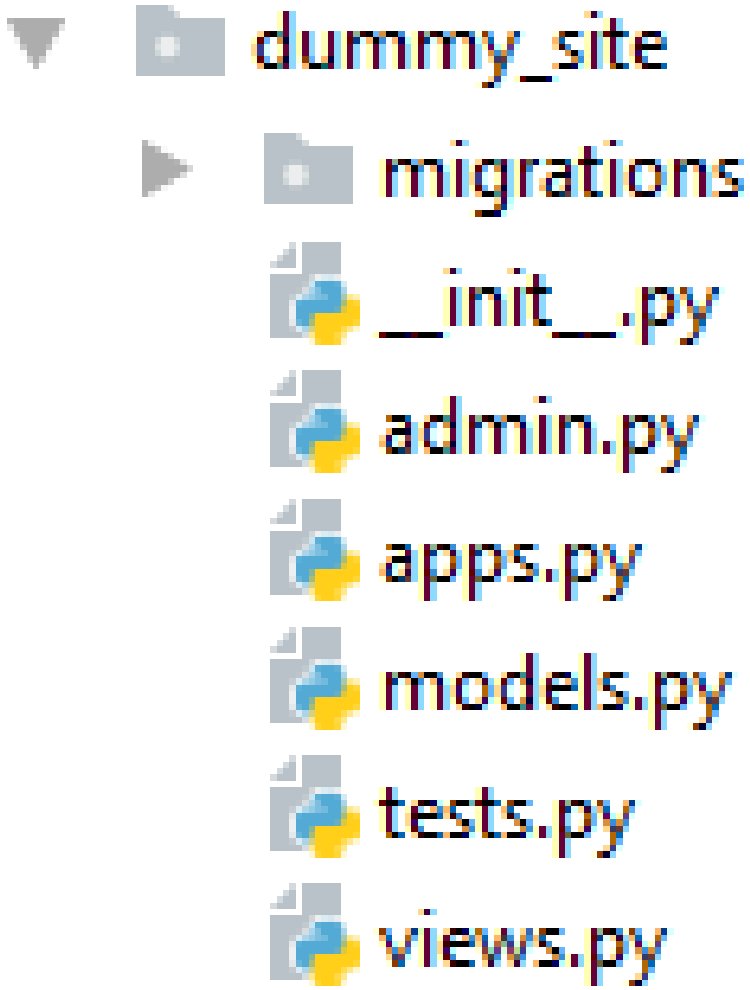
You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

ENG 11:37 AM 17/02/2017

Django app

```
$ python manage.py startapp
dummy_site
```

Dummy_site app



Writing our first test

tests.py × | urls.py × | models.py ×

```python
from django.test import TestCase


class TestViews(TestCase):

    # Test homepage rendering
    def test_home(self):
        response = self.client.get("")
        self.assertEquals(response.status_code, 200)

```

Running our first test

```
C:\Users\Anna\pyconna>pyconna\scripts\activate
(pyconna) C:\Users\Anna\pyconna>python manage.py test dummy_site.tests
```

Test result

```
(pyconna) C:\Users\Anna\pyconna>python manage.py test dummy_site.tests
Creating test database for alias 'default'...
F
======================================================================
FAIL: test_home (dummy_site.tests.TestViews)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Anna\pyconna\dummy_site\tests.py", line 12, in test_home
    self.assertEquals(response.status_code, 200)
AssertionError: 404 != 200

----------------------------------------------------------------------
Ran 1 test in 0.563s

FAILED (failures=1)
Destroying test database for alias 'default'...

(pyconna) C:\Users\Anna\pyconna>
```

Debugging and fixing the error

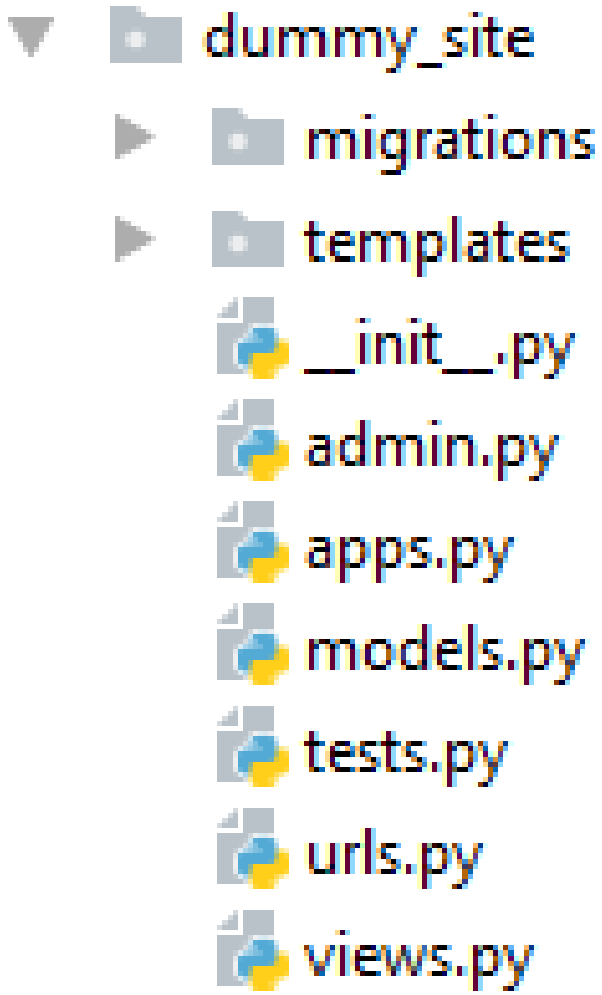
- 404 response = Page Not Found
- No view for home
- No template index.html
- No URL for home
- No app named dummy_site

Configure our app in settings.py

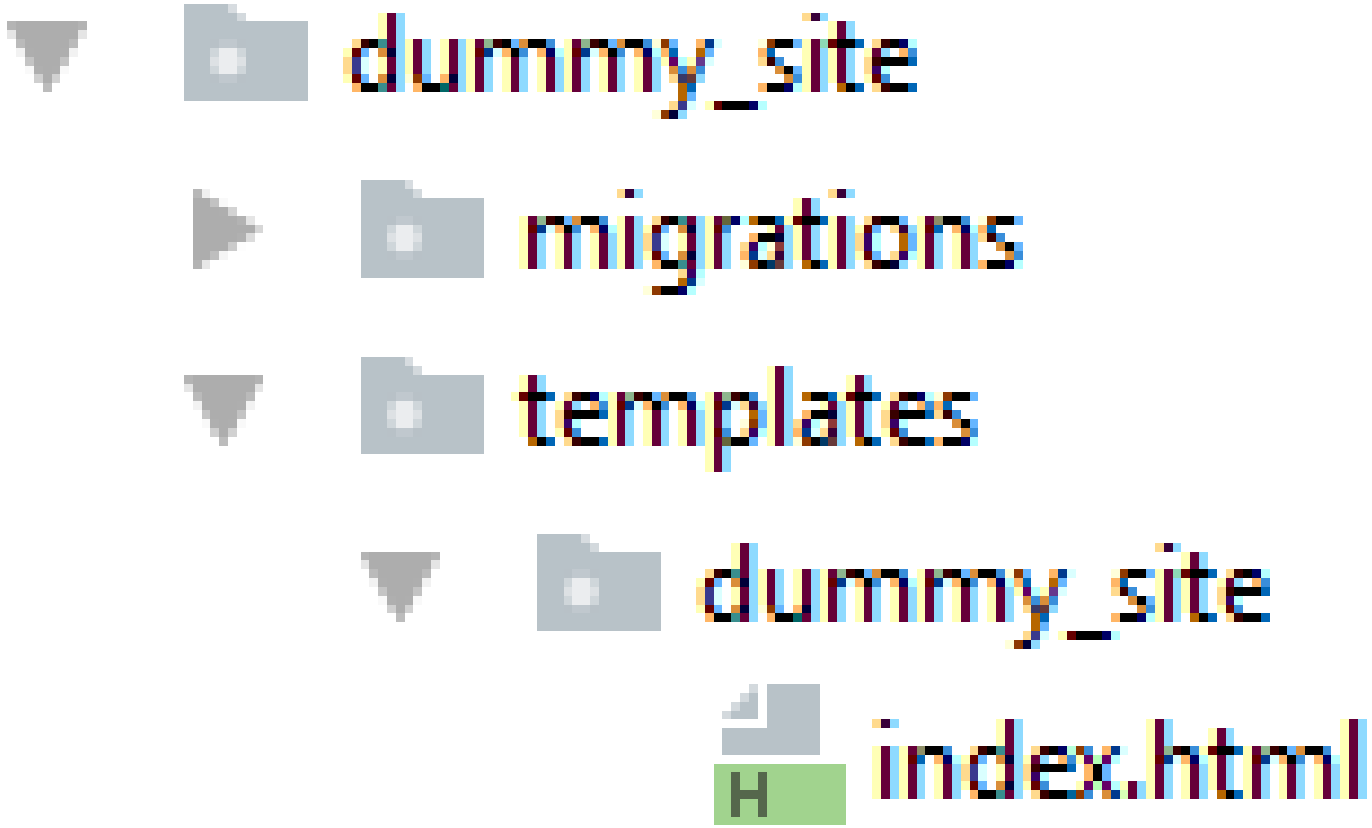
```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'dummy_site',
]
```

Add a urls.py to dummy_site



Add templates folder for dummy_site and create index.html file



Create a view for home in views.py

py × | index.html × | views.py × | urls.py × | settings.py ×

```python
from django.shortcuts import render
from django.http import HttpRequest

from datetime import datetime


"""Renders the home page"""
def home(request):
    assert isinstance(request, HttpRequest)
    return render(
        request,
        'dummy_site/index.html',
        {
            'title': 'Home Page',
            'message': 'Home Page.',
            'year': datetime.now().year,
        }
    )
```

Add url for home in dummy_site/urls.py

ts.py × | index.html × | views.py × | urls.py × | settings.py ×

```
from django.conf.urls import url
from dummy_site import views


urlpatterns = [
    url(r'^$', views.home, name='home'),
]
```

Include dummy_site.urls in dummy.urls

.py × | views.py × | dummy\urls.py × | settings.py ×

```
"""dummy URL Configuration..."""
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'', include('dummy_site.urls')),
]
```

Run dummy_site.tests again

```
(pyconna) C:\Users\Anna\pyconna>python manage.py test dummy_site
Creating test database for alias 'default'...
.
----------------------------------------------------------------------
Ran 1 test in 0.234s

OK
Destroying test database for alias 'default'...

(pyconna) C:\Users\Anna\pyconna>
```

More assertions for page rendering

- For context variables in the template
  - Title
  - Message
  - Year
- Content in the page

Testing models

- Test string representation
- Test verbose name plural, etc

In tests.py

tests.py × | urls.py × | models.py ×

```python
from django.test import TestCase

from .models import Entry


class TestViews(TestCase):...


class TestModel(TestCase):
    # Test string representation of model in admin
    def test_string_representation(self):
        entry = Entry(title="test entry")
        self.assertEqual(str(entry), entry.title)

    # Test model plural in admin
    def test_verbose_name_plural(self):
        self.assertEqual(str(Entry._meta.verbose_name_plural), "entries")
        self.assertNotEqual((Entry._meta.verbose_name_plural), "entrys")
```

Run our tests.py

```
(pyconna) C:\Users\Anna\pyconna>python manage.py test dummy_site
Creating test database for alias 'default'...
E
======================================================================
ERROR: dummy_site.tests (unittest.loader.ModuleImportFailure)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Python34\lib\unittest\case.py", line 58, in testPartExecutor
    yield
  File "C:\Python34\lib\unittest\case.py", line 577, in run
    testMethod()
  File "C:\Python34\lib\unittest\loader.py", line 32, in testFailure
    raise exception
ImportError: Failed to import test module: dummy_site.tests
Traceback (most recent call last):
  File "C:\Python34\lib\unittest\loader.py", line 312, in _find_tests
    module = self._get_module_from_name(name)
  File "C:\Python34\lib\unittest\loader.py", line 290, in _get_module_from_name
    __import__(name)
  File "C:\Users\Anna\pyconna\dummy_site\tests.py", line 4, in <module>
    from .models import Entry
ImportError: cannot import name 'Entry'


----------------------------------------------------------------------
Ran 1 test in 0.031s

FAILED (errors=1)
Destroying test database for alias 'default'...
```

Create a model Entry in models.py

py × urls.py × models.py × index.html ×

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User


class Entry(models.Model):
    title = models.CharField(max_length=200)
    date_posted = models.DateTimeField("date posted", default=timezone.now)
    author = models.ForeignKey(User)
```

Run tests.py

```
  File "C:\Users\Anna\pyconna\pyconna\lib\site-packages\django\db\utils.py", line 94, in __exit__
    six.reraise(dj_exc_type, dj_exc_value, traceback)
  File "C:\Users\Anna\pyconna\pyconna\lib\site-packages\django\utils\six.py", line 685, in reraise
    raise value.with_traceback(tb)
  File "C:\Users\Anna\pyconna\pyconna\lib\site-packages\django\db\backends\utils.py", line 64, in execute
    return self.cursor.execute(sql, params)
  File "C:\Users\Anna\pyconna\pyconna\lib\site-packages\django\db\backends\sqlite3\base.py", line 337, in execute
    return Database.Cursor.execute(self, query, params)
django.db.utils.OperationalError: no such table: dummy_site_entry

(pyconna) C:\Users\Anna\pyconna>
```

python manage.py makemigrations

```
(pyconna) C:\Users\Anna\pyconna>python manage.py makemigrations
Migrations for 'dummy_site':
  dummy_site\migrations\0001_initial.py:
    - Create model Entry

(pyconna) C:\Users\Anna\pyconna>
```

python manage.py migrate

```
(pyconna) C:\Users\Anna\pyconna>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, dummy_site, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying dummy_site.0001_initial... OK
  Applying sessions.0001_initial... OK

(pyconna) C:\Users\Anna\pyconna>
```

```
pyconna) C:\Users\Anna\pyconna>python manage.py test dummy_site
reating test database for alias 'default'...
F.
======================================================================
AIL: test_string_representation (dummy_site.tests.TestModel)
----------------------------------------------------------------------
raceback (most recent call last):
 File "C:\Users\Anna\pyconna\dummy_site\tests.py", line 19, in test_string_representation
   self.assertEqual(str(entry), entry.title)
ssertionError: 'Entry object' != 'test entry'
 Entry object
 test entry


======================================================================
AIL: test_verbose_name_plural (dummy_site.tests.TestModel)
----------------------------------------------------------------------
raceback (most recent call last):
 File "C:\Users\Anna\pyconna\dummy_site\tests.py", line 22, in test_verbose_name_plural
   self.assertEqual(str(Entry._meta.verbose_name_plural), "entries")
ssertionError: 'entrys' != 'entries'
 entrys
     ^
 entries
     ^^


----------------------------------------------------------------------
an 3 tests in 3.110s

AILED (failures=2)
estroying test database for alias 'default'...

pyconna) C:\Users\Anna\pyconna>
```

Modify Entry model in models.py

py × urls.py × models.py ×

```python
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User


class Entry(models.Model):
    title = models.CharField(max_length=200)
    date_posted = models.DateTimeField("date posted", default=timezone.now)
    author = models.ForeignKey(User)

    class Meta:
        managed = True
        verbose_name_plural = "entries"

    def __str__(self):
        return self.title
```

Run tests.py again

```
(pyconna) C:\Users\Anna\pyconna>python manage.py test dummy_site
Creating test database for alias 'default'...
...
----------------------------------------------------------------------
Ran 3 tests in 0.453s

OK
Destroying test database for alias 'default'...

(pyconna) C:\Users\Anna\pyconna>
```

Testing email

- Django’s test runner diverts emails sent during tests to a dummy outbox
- Test runner transparently replace email backend with test backend
- Test emails are sent to `django.core.mail.outbox`

Email test

tests.py × urls.py × models.py ×

```python
from django.test import TestCase
from django.core import mail


class EmailTest(TestCase):
    def test_send_email(self):
        # Send message.
        mail.send_mail(
                        'Subject here', 'Here is the message.',
                        'from@example.com', ['to@example.com'],
                        fail_silently=False,
                       )
        # Test that one message has been sent.
        self.assertEqual(len(mail.outbox), 1)
        # Verify that the subject of the first message is correct.
        self.assertEqual(mail.outbox[0].subject, 'Subject here')
```

Running tests.py

```
(pyconna) C:\Users\Anna\pyconna>python manage.py test dummy_site
Creating test database for alias 'default'...
....
----------------------------------------------------------------------
Ran 4 tests in 1.235s

OK
Destroying test database for alias 'default'...

(pyconna) C:\Users\Anna\pyconna>
```

- Test mailbox is emptied at the start of every test in a `django.test.TestCase`
- Manual reset is done by:

```
from django.core import mail

# Empty the test outbox
mail.outbox = []
```

In settings.py add email backend settings

```
# Email settings
EMAIL_USE_TLS = True
EMAIL_HOST = 'mail.server.com'
EMAIL_PORT = 2525
EMAIL_HOST_USER = ‘test@test.com'
EMAIL_HOST_PASSWORD = 'password’
```

In views.py

- Write a view for sending email

Testing login

- Client methods login() and logout() methods to simulate user login and logout
- Allows simulation/testing of roles or priviledges granted to logged in users
- Also allows simulation/testing or roles/priviledges granted to anonymous users

tests.py × urls.py × models.py ×

```python
from django.test import TestCase
from django.core import mail
from django.contrib.auth.models import User


class LoginTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username="test_user", email="test@test.com", password="pass1234")

    def test_login_success(self):
        response = self.client.login(username="test_user", password="pass1234")
        self.assertTrue(response)
```

RequestFactory

- Doesn’t act as a browser
- Provides a mechanism for generating requests that can be used as the first argument in a view
- Does not cater for login and logout
-

```python
class SimpleTest(TestCase):
    def setUp(self):
        # Every test needs access to the request factory.
        self.factory = RequestFactory()
        self.user = User.objects.create_user(
                                            username='jacob', email='jacob@test.com', password='top_secret')

    def test_details(self):
        # Create an instance of a GET request.
        request = self.factory.get('/')

        # Recall that middleware are not supported. You can simulate a
        # logged-in user by setting request.user manually.
        request.user = self.user

        # Or you can simulate an anonymous user by setting request.user to
        # an AnonymousUser instance.
        request.user = AnonymousUser()

        # Test my_view() as if it were deployed at /
        response = home(request)
        self.assertEqual(response.status_code, 200)
```

8 tips on how to speed up your tests

1. use MD5PasswordHasher
2. consider in-memory sqlite3
3. have more SimpleTestCase
4. use setUpTestData()
5. use mocks EVERYWHERE
6. be vigilant of what gets created in setUp()
7. don’t save model objects if not necessary
8. isolate unit tests

In conclusion…

“When testing,
more is better” *– Ana Balica (2016)*

Questions???

References/ Resources

- https://speakerdeck.com/**anabalica/duth-testing**-in-django
- https://www.youtube.com/watch?v=EHyKzPQFXzo
- https://**docs.dango**project.com/en/**1.10/**
- https://**tutorial.djangogirls**.org/en/
- https://readthedocs.org/projects/**test-driven-django-development**/downloads/pdf/la**test/**
- http://test-driven-django-development.readthedocs.io/en/latest/

End of presentation

Thank you!