django-party-pack Documentation

Release 0.2.0

Daniel Greenfeld

May 29, 2015

Contents

Because these are great patterns and tools that beginners should be exposed to right away. I've learned them from the various *Contributors*, who are people I admire as Django and Python developers.

CHAPTER 1

Basic Stuff

1.1 Installation

Note: For things with the following it means type it at the command line and hit enter:

```
$ ls -al
```

1.1.1 Before you start

Do you have pip, virtualenv, virtualenvwrapper, and git-scm installed? If not, you'll need to get those on your machine before proceeding.

If you need to install pip:

```
$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
$ python get-pip.py
```

If you need to install virtualenv:

```
$ pip install virtualenv
```

If you need to install virtualenvwrapper:

```
$ pip install virtualenvwrapper
```

If you need to install git:

- http://git-scm.com

1.1.2 The Basics

Create a virtualenv for this project. We do this so we isolate all our work from the rest of Python on our computer:

```
$ mkvirtualenv dpkenv
```

Now we clone django-party-pack and go into django-party-pack:

```
$ git clone https://pydanny@github.com/pydanny/django-party-pack.git
$ cd django-party-pack
```

Now let's install our dependencies:

```
$ pip install -r requirements.txt
```

This may take a few minutes. Feel free to go get some coffee. :)

1.1.3 Settings setup

We're going to follow what Django BDFL Jacob Kaplan-Moss advocates as best practices for dealing with settings. That means we're going to ignore the manage.py file in the root of our Django project and use the django-admin.py script. In order to do that, we need to take a few more steps.

First, we add some virtualenv bits to allow us to access the settings properly:

```
$ echo "export DJANGO_SETTINGS_MODULE=settings.dev" >> $VIRTUAL_ENV/bin/postactivate
$ echo "unset DJANGO_SETTINGS_MODULE" >> $VIRTUAL_ENV/bin/postdeactivate
```

This will allow you to eschew passing in –settings= into management commands.

Now we add to the virtualenv paths our pollaxe project:

```
add2virtualenv <<path to django-party-pack repo>>/pollaxe
```

1.1.4 Running standard Django Commands

Try out the project:

```
$ django-admin.py syncdb
$ django-admin.py runserver
```

1.1.5 Running django-coverage

Simply run this command:

```
$ django-admin.py test
```

Now open the pollaxe/coverage/index.html file in your favorite browser.

1.1.6 Building these sphinx docs

Want to have a local copy of these documents? Easy! Change to our docs directory:

```
$ cd docs
```

Now we generate the sphinx docs in html format:

```
$ make html
```

1.2 Setting up a test runner

Ned Batchelder's coverage.py is an invaluable tool for any Python project. django_coverage makes coverage.py run inside of Django, and this is my preferred way of using that tool.

1.2.1 Step 1 - environment prep

In your virtualenv install the necessary requirements:

```
$ pip install -r requirements.txt
```

Make a *coverage* directory in your project directory:

```
# This is done for you in django-party-pack
#   but you'll need to remember it for future projects
$ mkdir coverage
```

1.2.2 Step 2 - create testrunner.py

Create a testrunner.py file into your project root and paste in the following code:

```
# Make our own testrunner that by default only tests our own apps

from django.conf import settings
from django.test.simple import DjangoTestSuiteRunner
from django_coverage.coverage_runner import CoverageRunner

class OurTestRunner(DjangoTestSuiteRunner):
    def build_suite(self, test_labels, *args, **kwargs):
        return super(OurTestRunner, self).build_suite(test_labels or settings.PROJECT_APPS, *args, *

class OurCoverageRunner(OurTestRunner, CoverageRunner):
    pass
```

1.2.3 Step 3 - settings customization

The first thing you'll notice about dpp is that apps installment is broken up into three variables:

- *PREREQ_APPS* - These are either built-in Django apps or third-party apps you don't want to test.
- *PROJECT_APPS* - These are the custom apps you've written for your project. You want to test these.
- *INSTALLED_APPS* - This is what Django loads into it's app cache. We generate this iterable by adding *PREREQ_APPS* to *PROJECT_APPS*.

Here is the sample code from dpp/pollaxe project settings.py file:

```
PREREQ_APPS = (
     'django.contrib.auth',
     'django.contrib.contenttypes',
     'django.contrib.sessions',
     'django.contrib.sites',
     'django.contrib.messages',
     'django.contrib.admin',
)

PROJECT_APPS = (
    'polls', # or whatever your custom project uses
)

INSTALLED_APPS = PREREQ_APPS + PROJECT_APPS
```

Also in settings.py, underneath where you have defined the *PREREQ_APPS* setting, add the following:

```
TEST_RUNNER = 'testrunner.OurCoverageRunner'
COVERAGE_MODULE_EXCLUDES = [
    'tests$', 'settings$', 'urls$', 'locale$',
    'migrations', 'fixtures', 'admin$',
]
COVERAGE_MODULE_EXCLUDES += PREREQ_APPS
COVERAGE_REPORT_HTML_OUTPUT_DIR = "coverage"
```

1.2.4 Step 4 - run it!

From the command-line:

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

Open file:///path-to-your-project/coverage/index.html in a web browser and check out your coverage.

1.3 JavaScript

Some ideas to incorporate JavaScript into your application.

1.3.1 Coding Standard for JavaScript

- https://github.com/jbalogh/zamboni/blob/master/STYLE.rst

1.4 Coding Conventions

So we are all on the same track.

1.4.1 Philosophy

Zen of Python

Try it out at the shell:

```
import this
```

My favorite parts:

- Explicit is better than implicit.
- Simple is better than complex.
- Readability counts.
- Errors should never pass silently.

PEP-8 is my friend

No *import* *! Even in *urls.py*!

All Docs go on rtfd.org!

No alternative compares to http://rtfd.org. Not github, bitbucket, or google project wikis compare. And even the python.packages.com site is out of the lead of rtfd.org. Stop trying other things and come to the current leader in documentation hosting. Why?

1. It takes your repo and makes it look awesome.
2. It puts all the Python docs into one place for good searching.
3. It plays nice with git, hg, and svn. Wikis generally are through the web.
4. You can accept pull requests on docs. This way you can edit/reject bad documentation.
5. Makes your project and work much more visible.
6. The lead maintainer, Eric Holscher, is incredibly supportive and has both PSF and Revsys support.

1.4.2 Code Bits

Docs

Besides *admin.py*, all new python files need to be added to the appropriate *app_<app_name>.rst* or *reference_<app_name>.rst* file.

Templates

- *snippets/_<name>.html* is for templates that are added via *include* or *templatetags*.
- `{# unstyled #}` is a flag for designers that the template is still untouched by their hands.

urls.py

Even in urls.py you want clean code, right?

Explicit imports

See how it is done:

```
# See this commented out? 'import *' usually slows things down AND makes it harder to debug
# import *

# Explicit imports are easier to debug
from polls import views
...
```

Using the url() function

Pythonistas love explicitly but this is implicit and henceforth not ideal:

```
# Don't do this!
url(
    r'^$',
    views.poll_list,
```

```
    'poll_list',
),

# Or this!
(
    r'^$',
    views.poll_list,
    'poll_list',
),
```

And here is the preferred and wonderfully explicit Jacob Kaplan-Moss / Frank Wiles pattern:

```
url(
    regex=r'^$',
    view=views.poll_list,
    name='poll_list',
),
```

See how each argument is explicitly named? Wonderful!

Calling specific views

This is hard to debug because Django gets a bit too 'magical' and the trace often doesn't give you as much or is longer:

```
# Don't do this!
url(
    regex=r'^$',
    view='polls.views.standard.poll_list', # this single bit makes it harder to debug on errors
    name='poll_list',
),
```

Instead we do this:

```
url(regex=r'^$',
    view=views.poll_list,
    name='poll_list',
),
```

Generic Exceptions are the DEVIL

This is the DEVIL:

```
try:
    do_blah()
except:
    pass
```

Do this instead:

```
class BlahDoesNotWork(Exception): pass

try:
    do_blah
except ImportError:
    # do something
except AttributeError:
    # do something else
```

```
except Exception as e:
    msg = "{0} has failed!".format(str(e))
    logging.error(msg)
    raise BlahDoesNotWork(msg)
```

1.5 Contributors

I didn't do this in a vacuum. This is built off of packages and libraries created by a huge number of incredible people. And everything on this was taught to me either on the job or by looking at other people's examples of how to do things. Here we go:

- Audrey Roy for coming up with this idea and agreeing to marry me.
- Aaron Kavlie, Geoffrey Jost, and Preston Holmes for helping organize things.
- Chris Shenton for showing me that more notes are better - even if they seem stupid to take at the time.
- Eric Holscher for rtfd.org
- Evgany Fadeev for showing me how to work Sphinx autodoc.
- Frank Wiles for general Django code cleanliness and inspiring the Cartwheel way.
- Georg Brandl for Sphinx
- George Song for django-coverage
- Gisle Aas for the way django-coverage is implemented in this project
- Jacob Kaplan-Moss for a culture of documentation plus schooling me personally hard on exceptions and writing better tests.
- James Tauber and Alex Gaynor for settings.PROJECT_ROOT.
- Nate Aune for teaching me that tests should be a story.
- Ned Batchelder for coverage.py
- Steve Holden for teaching me better skills explaining technical things in text.

  If I missed anyone I apologize!

CHAPTER 2

API/Reference Docs

2.1 Reference for Polls App

The polls app is a copy of the Django tutorial with some mild PEP-8 cleanup.

2.1.1 polls.models

class polls.models.**Choice**(**args*, ***kwargs*)
Choices on a poll

class polls.models.**Poll**(**args*, ***kwargs*)
An individual poll to be tested

2.1.2 polls.views

polls.views.**detail**(*request*, *poll_id*, *template_name='polls/detail.html'*)
Show detail on a poll

polls.views.**index**(*request*, *template_name='polls/index.html'*)
Show a list of polls

polls.views.**vote**(*request*, *poll_id*, *template_name='polls/detail.html'*)
user votes on a poll

2.1.3 polls.tests

class polls.tests.test_models.**TestPolls**(*methodName='runTest'*)

test_poll_create()
Can we create a poll?

- Seems trivial now
- But for complex systems what started out as a simple create can get complex
- Get your test coverage up!

test_was_published_today()

class polls.tests.test_views.**TestPollSample**(*methodName='runTest'*)

setUp()

test_poll_detail()
Check if the poll detail displays

test_poll_index()
Check if the poll index displays

test_poll_vote()
vote on a poll

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

p

C

D

I

P

S

T

V