

Pyramid: A Drive-by Tutorial

Paul Everitt@dcpython - Feb 5 2013

I Am...

- Prehistoric
 - Web, Python, Zope, Plone, Pyramid
- Agendaless Consulting with Chris and Tres, Fredericksburg
- Manage large web projects...yep, I'm pimping

Why Pyramid?







What is Pyramid?

- Python (2 and 3) web framework
- Merger of repoze.bfg (from the Zope guys) and Pylons
- Target audience: Projects that start small and finish big

Show Me The Money

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello')

def main():
    config = Configurator()
    config.add_route('hello', '/')
    config.add_view(hello_world, route_name='hello')
    app = config.make_wsgi_app()
    return app

if __name__ == '__main__':
    app = main()
    server = make_server('0.0.0.0', 6547, app)
    print ('Starting up server on http://localhost:6547')
    server.serve_forever()
```


What...what's that print?

- `print ('Starting up server on http://localhost:6547')`
- Yep, we're going to do this on Python 3.3

Tutorial walkthrough at:
<https://github.com/pauleveritt/dcpython>

Goals of Pyramid

Only Pay For What You Eat

- Easy to start
- Few choices forced on you
- Small codebase

Quality

Quality

- Full test and docs coverage (culture of docs and tests)
- Performance, profiling

Small to Big

- Starting point and finishing point
- Unique features that let you scale *your* design (configuration, events, custom renderers, view predicates, traversal, ...)

Back to 3.3

Setting Up Your virtualenv

```
$ wget http://python-distribute.org/distribute\_setup.py  
$ pyvenv-3.3 env33  
$ env33/bin/python3.3 distribute_setup.py  
$ env33/bin/easy_install-3.3 pip
```

Installing and Running Pyramid

```
$ env33/bin/pip-3.3 install pyramid  
...chuggalugga...  
$ env33/bin/python3.3 hello.py
```

Boring

- Single-file “applications” are possible
- Just not the target
- Let’s do a simple package

2: Hello Package

- `setup.py`
- `tutorial/`
 - `__init__.py`
 - `helloworld.py`
 - `views.py`

02: setup.py

```
from setuptools import setup
```

```
requires = [  
    'pyramid',  
]
```

```
setup(name='tutorial',  
      entry_points="""\  
[paste.app_factory]  
main = tutorial:main  
""",  
      )
```


02: __ini__.py

```
# package
```

02: helloworld.py

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
```

```
def main():
    config = Configurator()
    config.add_route('hello', '/')
    config.scan('views')
    app = config.make_wsgi_app()
    return app
```

```
if __name__ == '__main__':
    app = main()
    server = make_server('0.0.0.0', 6547, app)
    print ('Starting up server on http://localhost:6547')
    server.serve_forever()
```

02: views

```
from pyramid.response import Response  
from pyramid.view import view_config
```

```
@view_config(route_name='hello')  
def hello_world(request):  
    return Response('Hello')
```

03: Pyramid Configuration

- .ini files
- pserve (and friends)

03: development.ini

[app:main]

use = egg:tutorial

pyramid.reload_templates = true

[server:main]

use = egg:pyramid#wsgiref

host = 0.0.0.0

port = 6543

[loggers]

keys = root

[handlers]

keys = console

[formatters]

keys = generic

[logger_root]

level = INFO

handlers = console

[handler_console]

class = StreamHandler

args = (sys.stderr,)

level = NOTSET

formatter = generic

[formatter_generic]

format = %(asctime)s %(levelname)-5.5s [%(name)s][%(threadName)s] %(message)s

03: `__init__.py`

```
from pyramid.config import Configurator
```

```
def main(global_config, **settings):  
    config = Configurator(settings=settings)  
    config.add_route('hello', '/')  
    config.scan()  
    return config.make_wsgi_app()
```


04: Running

```
$ pserve development.ini --reload
```

```
Starting server in PID 32130.
```

```
Starting HTTP server on http://0.0.0.0:6547
```

```
127.0.0.1 - - [03/Feb/2013 20:03:58] "GET / HTTP/1.1" 200 5
```

Yikes! Culture of Testing!

- \$ pip-3.3 install nose WebTest

04: tests.py

```
import unittest
```

```
from pyramid import testing
```

```
class ViewTests(unittest.TestCase):  
    def setUp(self):  
        self.config = testing.setUp()  
  
    def tearDown(self):  
        testing.tearDown()  
  
    def test_my_view(self):  
        from tutorial.views import hello_world  
        request = testing.DummyRequest()  
        response = hello_world(request)  
        self.assertEqual(response.status, '200 OK')
```

```
# ....con't
```

```
class FunctionalTests(unittest.TestCase):  
    def setUp(self):  
        from tutorial import main  
        settings = {}  
        app = main(settings)  
        from webtest import TestApp  
        self.testapp = TestApp(app)  
  
    def test_it(self):  
        res = self.testapp.get('/', status=200)  
        self.assertIn(b'Hello', res.body)
```

04: Test Running

```
$ nosetests .
```

```
..
```

```
-----
```

```
Ran 2 tests in 0.498s
```

```
OK
```

05: Let's Do a Template

- Change our view to use a “renderer”
- In this case, a template
- Makes test-writing better
- tutorial/templates/
 - helloworld.pt

05: views.py

```
from pyramid.view import view_config

@view_config(route_name='hello', renderer='templates/helloworld.pt')
def hello_world(request):
    return dict(title='Hello World')
```


05: templates/helloworld.pt

```
<html>
<head>
  <title>${title}</title>
</head>
<body>
<div>
  <h1>${title}</h1>
</div>
</body>
</html>
```

05: tests.py

```
import unittest
```

```
from pyramid import testing
```

```
class ViewTests(unittest.TestCase):
```

```
    def setUp(self):  
        self.config = testing.setUp()
```

```
    def tearDown(self):  
        testing.tearDown()
```

```
    def test_my_view(self):  
        from tutorial.views import hello_world  
        request = testing.DummyRequest()  
        response = hello_world(request)  
        self.assertEqual(response['title'],  
                          'Hello World')
```

```
class FunctionalTests(unittest.TestCase):
```

```
    def setUp(self):  
        from tutorial import main  
        settings = {}  
        app = main(settings)  
        from webtest import TestApp  
        self.testapp = TestApp(app)
```

```
    def test_it(self):  
        res = self.testapp.get('/', status=200)  
        self.assertIn(b'Hello World', res.body)
```

06:View Classes

06: views.py

```
from pyramid.view import view_config

class HelloWorld(object):
    def __init__(self, request):
        self.request = request

    @view_config(route_name='hello',
                  renderer='templates/helloworld.pt')
    def hello_world(self):
        return dict(title='Hello World')
```

06: tests.py

```
def test_my_view(self):  
    from tutorial.views import HelloWorld  
    request = testing.DummyRequest()  
    inst = HelloWorld(request)  
    response = inst.hello_world()  
    self.assertEqual(response['title'],  
                      'Hello World')
```

07: Static Assets

- tutorial/static/
 - hello.css
 - logo.png

07: __init__.py

```
from pyramid.config import Configurator
```

```
def main(global_config, **settings):  
    config = Configurator(settings=settings)  
    config.add_route('hello', '/')  
    config.add_static_view(name='static', path='tutorial:static')  
    config.scan()  
    return config.make_wsgi_app()
```

07: helloworld.pt

```
<html>
<head>
  <title>${title}</title>
  <link rel="stylesheet"
    href="${request.static_url('tutorial:static/hello.css')}" />
</head>
<body>
<div>
  <h1>${title}</h1>
  <p>
    
  </p>
</div>
</body>
</html>
```


08: Forms

- Use Deform for CRUD (with Colander and Peppercorn)
- `$ pip install deform`
- `deform_bootstrap` optional
- Several other form libraries for Pyramid
- This step is **ONLY** rendering

08: views.py

```
import colander
import deform
from pyramid.view import view_config

class Person(colander.MappingSchema):
    name = colander.SchemaNode(colander.String())
    age = colander.SchemaNode(colander.Integer(),
                              validator=colander.Range(0, 200))

class HelloWorld(object):
    def __init__(self, request):
        self.request = request

    @view_config(route_name='hello',
                  renderer='templates/helloworld.pt')
    def hello_world(self):
        schema = Person()
        form = deform.Form(schema, buttons=('submit',))

        return dict(title='Hello World', form=form,
                      reqts=form.get_widget_resources())
```

08: helloworld.pt

```
<html>
<head>
  <title>${title}</title>
  <link rel="stylesheet"
    href="${request.static_url('tutorial:static/hello.css')}" />
  <tal:block repeat="reqt reqts['css']|[]">
    <link rel="stylesheet" type="text/css"
      href="${request.static_url('deform:static/' + reqt)}" />
  </tal:block>
  <tal:block repeat="reqt reqts['js']|[]">
    <script src="${request.static_url('deform:static/' + reqt)}"
      type="text/javascript"></script>
  </tal:block>
</head>
<body>
<div>
  <h1> ${title}</h1>
  <p tal:content="structure form.render()">
  </p>
</div>
</body>
</html>
```

08: `__init__.py`

```
from pyramid.config import Configurator
```

```
def main(global_config, **settings):  
    config = Configurator(settings=settings)  
    config.add_route('hello', '/')  
    config.add_static_view(name='static', path='tutorial:static')  
    config.add_static_view('deform_static', 'deform:static/')  
    config.scan()  
    return config.make_wsgi_app()
```

09: Form Validation

09: views.py

```
import colander
import deform
from pyramid.decorator import reify
from pyramid.view import view_config, view_defaults

class Person(colander.MappingSchema):
    name = colander.SchemaNode(colander.String())
    age = colander.SchemaNode(colander.Integer(),
                             validator=colander.Range(0, 200))

@view_defaults(route_name='hello',
               renderer='templates/helloworld.pt')
class HelloWorld(object):
    title = 'Hello World'
    status_message = None

    def __init__(self, request):
        self.request = request

    @reify
    def form(self):
        schema = Person()
        return deform.Form(schema, buttons=('submit',))

    @reify
    def reqts(self):
        reqts = self.form.get_widget_resources()
        return dict(
            js_links=reqts.get('js', []),
            css_links=reqts.get('css', [])
        )
```

09: views.py

```
@view_defaults(route_name='hello',
                renderer='templates/helloworld.pt')
class HelloWorld(object):
    title = 'Hello World'
    status_message = None

    @view_config()
    def hello_world(self):

        return dict(form=self.form)

    @view_config(request_param='submit')
    def submit_handler(self):
        form = self.form
        controls = self.request.POST.items()

        try:
            appstruct = form.validate(controls)
        except deform.ValidationFailure as e:
            return dict(form=e)


        ## Process the valid form data
        self.status_message = 'Added person: ' + appstruct['name']

        return dict(form=form, appstruct=appstruct)
```

09: helloworld.pt

```
<html>
<head>
  <title>${view.title}</title>
  <link rel="stylesheet"
    href="${request.static_url('tutorial:static/hello.css')}" />
  <tal:block repeat="reqt view.reqs['css_links']">
    <link rel="stylesheet" type="text/css"
      href="${request.static_url('deform:static/' + reqt)}" />
  </tal:block>
  <tal:block repeat="reqt view.reqs['js_links']">
    <script src="${request.static_url('deform:static/' + reqt)}"
      type="text/javascript"></script>
  </tal:block>
</head>
<body>
<div>
  <h1> ${view.title}</h1>
  <p tal:condition="view.status_message">
    <em>${view.status_message}</em>
  </p>
  <p tal:content="structure form.render()">
  </p>
</div>
</body>
</html>
```


Validation

 Pyramid™ **Hello World**


There was a problem with your submission
Errors have been highlighted below

Name*

Required

Age*

Required

 Pyramid™ **Hello World**

Added person: Paul

Name*

Age*

...and more

- Other template languages
- SQLAlchemy (or others) for storage/retrieval
- Authentication and authorization
- Sessions, events, i18n, resources and traversal, advanced configuration, ...
- Substance D