wheezy.web introduction

a lightweight, high performance, high concurrency WSGI web framework with the key features to **build modern, efficient web**

Andriy Kornatskyy

Lviv, Ukraine
September, 2012

https://bitbucket.org/akorn

wheezy.web performace

- Live Demo: http://wheezy.pythonanywhere.com/
- Concurrency **500**
- Throughtput **10K+** requests per seconds
- **Double** above for Content Cache
- 99% requests served within **84** ms or less
- Bandwidth 180 Mbit per second
- Intel Xeon CPU X3430 @ 2.40GHz x 4
- Debian, Kernel 3.2.0-3-amd64, uwsgi 1.2.6

wheezy.web key notes

- Requires Python 2.4-2.7, 3.2+ or PyPy
- Lightweight, high-level web framework
- Optimized for high performance, high concurrency
- Well tested and documented
- MVC architectural pattern (push-based)
  - Encourage separation of code and content
  - Work more productively at higher level of abstraction
- Template Engine Agnostic

wheezy.web design

- Modular design
- Strong separation of domain concerns
  - routing
  - validation
  - http
  - html
  - handler
  - security
  - template
  - caching

wheezy.web routing

- Maps URL patterns to a handler
- Route Builders: plain simple strings, curly expressions or regular expressions

```
urls = [
     ('posts/2012', posts_for_2012, {'year': 2012}),
     ('posts/{year}', posts_by_year),
     ('posts/(?P<year>\d+)/(?P<month>\d+)', posts_by_month)
]
```

- Mapping can include other mappings and constructed dynamically

```
all_urls = [
    ('membership/', membership_urls)
]

all_urls += [
    url('{url:any}', not_found)
]
```

wheezy.web routing

- Named groups (data extracted from URL)
- Extra parameters (merged with named groups)
- Named mapping (construct paths by name)

```
all_urls += [
    url('static/{path:any}', static_files, name='static'),
    url('favicon.ico', static_files, {'path': 'img/favicon.ico'})
]


# wheezy.template syntax
@path_for('static', path='site.css')
```

wheezy.web handler

- Handler is **any** callable that accepts an instance of HTTPRequest and returns HTTPResponse

```
def welcome(request):
    query = last_item_adapter(request.query)
    response = HTTPResponse()
    response.write('Hello %s!' % query.get('user', 'World'))
    return response
```

- HTTPRequest is a wrapper around WSGI **environ**
  - query, form, files, cookies, path (SCRIPT_NAME + PATH_INFO), etc
- HTTPResponse calls WSGI **start_response**
  - buffers output, manages headers, cookies, etc

wheezy.web handler

- BaseHandler provides methods that integrate:
  - Routing
  - Model Update / Validation
  - Authentication / Authorization
  - Template Rendering / Internationalization
  - AJAX / JSON
  - XSRF / Resubmission Protection

```
class WelcomeHandler(BaseHandler):

    def get(self):
        query = last_item_adapter(request.query)
        response = HTTPResponse()
        response.write('Hello %s!' % query.get('user', 'World'))
        return response
```

wheezy.web model update

- Converts user input into data models (numbers, date/time, lists, custom value providers)
- Uses **plain** python objects

```
class Credential(object):
    username = u('')
    password = u('')

class SignInHandler(BaseHandler):

    def get(self, credential=None):
        credential = credential or Credential()
        return self.render_response('membership/signin.html',
                                    credential=credential)
    def post(self):
        credential = Credential()
        if not self.try_update_model(credential):
            return self.get(credential)
        return self.redirect_for('default')
```

wheezy.web model validation

- Validates user input

```
credential_validator = Validator({
    'username': [required, length(max=10)],
    'password': [required, length(min=8, max=12)]
})

class SignInHandler(BaseHandler):

    def post(self):
        credential = Credential()
        if (not self.try_update_model(credential)
            or not self.validate(credential, credential_validator)):
            return self.get(credential)
        return self.redirect_for('default')
```

- Built-in rules (thread safe), i18n ready:
  - required, length, range, compare, predicate, regex, slug, email, and_, or_, iterator, one_of, relative_date, etc.

wheezy.web model validation

- Nested validators for composite models

```
registration_validator = Validator({
    'credential': credential_validator
})
```

- Custom validation rules, custom messages

```
def check(self, value, name, model, result, gettext):
    return True
```

- Validation Mixin

```
class MyService(ValidationMixin):

     def authenticate(self, user):
         if not self.validate(user, user_validator):
             return False
         if not self.repository.membership.authenticate(credential):
             self.error(self.gettext(
                 "The username or password provided is incorrect."))
             return False
         return True
```

wheezy.web authentication

- Integration with Python Cryptography Toolkit
- Ticket (a proof of authorization that can not easily be forged):
  - Valid for certain period of time
  - Value is signed to prove it authenticity
  - Encrypted to protect sensitive information
  - Noise to harden forgery

```
options = {
    'CRYPTO_VALIDATION_KEY': 'LkLlYR5WbTk54kaIgJOp',
    'CRYPTO_ENCRYPTION_KEY': 'rH64daeXBZdgrR7WNawf'
}
ticket = Ticket(max_age=1200, salt='CzQnV0KazDKElBYiIC2w', digestmod=sha1,
    cypher=aes192, options=options)
```

wheezy.web authentication

```
protected_value = ticket.encode('hello')
assert 'hello' == ticket.decode(protected_value)
```

- Principal (attributes: id, roles, alias, extra)

```
class SignInHandler(BaseHandler):

    def get(self, credential=None):
        if self.principal:
            return self.redirect_for('default')
        ...

    def post(self):
        credential = Credential()
        if not self.try_update_model(credential)
            or not self.validate(credential, credential_validator)
            or not self.factory.membership.authenticate(credential)):
            ...
        self.principal = Principal(
            id=credential.username,
            alias=credential.username,
            roles=tuple(self.factory.membership.roles(
                credential.username)))
        return self.see_other_for('default')
```

wheezy.web authorization

- Authorization specify access rights to resources and provide access control in particular to your application

```
class MyHandler(BaseHandler):

    @authorize(roles=('business',))
    def get(self):
        ...
        return response
```

- **authorize** decorator return HTTP status code 401 (Unauthorized)
- Use **HTTPErrorMiddleware** to route 401 status code to signin page

wheezy.web middleware

- Key Characteristics
  - Transparent interceptor of incoming HTTP request to handler
  - Chained/Stacked, so one pass request to following
- Capable
  - Supply additional information in request context
  - Inspect response, override it, extend or modify

```
def middleware(request, following):
    if following is not None:
        response = following(request)
    else:
        response = ...
    return response
```

wheezy.web middleware

- Built-in middlewares:
  - Bootstrap Defaults, Path Routing, HTTP Error, HTTP Cache
- Middleware factories are explicitely passed to WSGI Application (**order is essential**):

```
from config import options
from urls import all_urls


main = WSGIApplication(
    middleware=[
        bootstrap_defaults(url_mapping=all_urls),
        http_cache_middleware_factory,
        http_error_middleware_factory,
        path_routing_middleware_factory
    ],
    options=options)
```

wheezy.web configuration

- Python **dict** as configuration options

```
options = {}

# HTTPErrorMiddleware
options.update({
    'http_errors': defaultdict(lambda: 'http500', {
        # HTTP status code: route name
        400: 'http400',
        401: 'signin',
        500: 'http500'
    }),
})
```

- Python **ConfigParser**

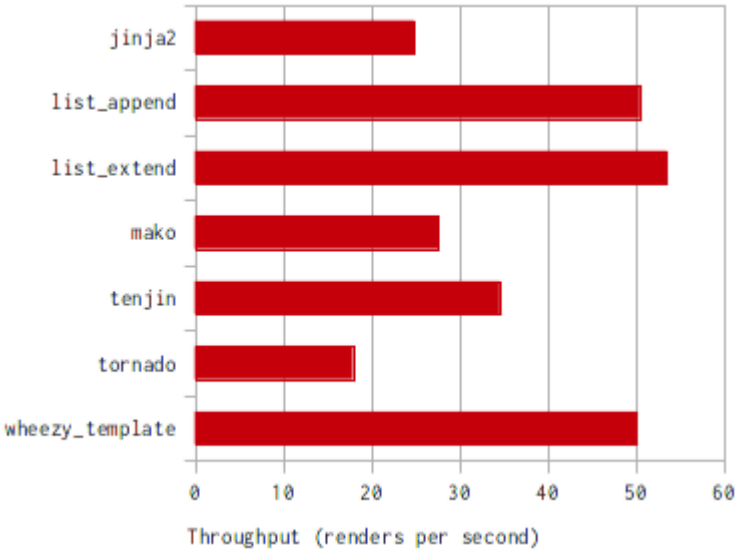
```
config.read('production.ini')

options['CRYPTO_ENCRYPTION_KEY'] = config.get('crypto', 'encryption-key')
```

wheezy.web template

- Template engine **agnostic**, integration with
  - jinja2, mako, tenjin and wheezy template
- Wheezy Template
  - Compact, **Extensible**, Blazingly Fast
  - Intuitive, No time to Learn
  - Do Not Repeat Yourself

```
@require(user, items)
Welcome, @user.name!
@if items:
    @for i in items:
        @i.name: @i.price!s.
    @end
@else:
    No items found.
@end
```



wheezy.web template

- Wheezy Template Context Preprocessor
  - Process templates with syntax for preprocessor engine and vary runtime templates (with runtime engine factory) by some key function that is context driven
  - Helpful in preprocessing
    - gettext messages, static routes, includes, etc

```
#_('My Site')

<script src="#path_for('static', path='js/core.js')"
    type="text/javascript"></script>

#include("shared/snippet/menu-signin.html")
```

wheezy.web widgets

- Derived from the idea of **code reuse**
- Small snippets of HTML
- Built-in widgets
  - hidden, textbox, password, textarea, checkbox, label, dropdown, radio, error, etc
- Take care of **validation errors**
- Integration with
  - jinja2, mako, tenjin and wheezy template

wheezy.web widgets

- Increase Template Readability

```
@model.remember_me.checkbox(class_='i')
```

- Translated at template **preprocessing** phase

```
<input id="remember-me" name="remember_me" type="checkbox" value="1"
@if 'remember_me' in errors:
 class="error i"
@else:
 class="i"
@end
@if model.remember_me:
 checked="checked"
@end
/>
```

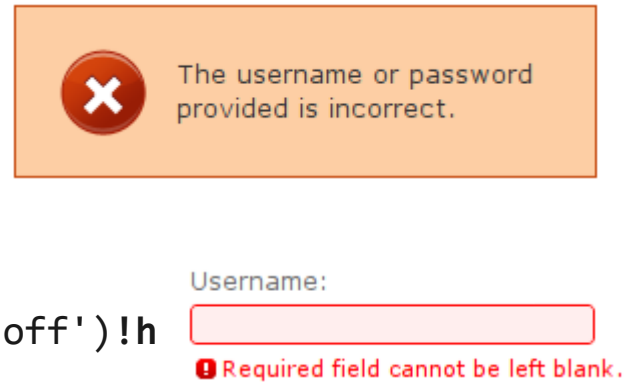
- Effectively renders the HTML at runtime

```
<input id="remember-me" name="remember_me" type="checkbox"
       value="1" class="i" />
```

wheezy.web widgets

- Compact, Expressive and Clean. No Limits

```
@model.error()
<form action="@path_for('signup')" method="post">
    <fieldset>
       <legend>Account Information</legend>
       <p>
            @credential.username.label('Username:')
            @credential.username.textbox(autocomplete='off')!h
            @credential.username.error()
       </p>
             ...
       <p>
            @model.questionid.label('Security question:')
            @model.questionid.dropdown(choices=questions)
            @model.questionid.error()
       </p>
   </fieldset>
</form>
```



wheezy.web protection

- Cross-site request forgery (CSRF/XSRF)
  - Handler Validation

```
class MyHandler(BaseHandler):

    def post(self):
        if not self.validate_xsrf_token():
            return self.redirect_for(self.route_args.route_name)
        ...
```

  - HTML Widget

```
<form action="..." method="post">
    @xsrf()
```

wheezy.web protection

- Form Resubmission
  - Handler Validation

```
class MyHandler(BaseHandler):

    def post(self):
        if not self.validate_resubmission():
            self.error('Your request has been queued.')
            return self.get()
        ...
```

  - HTML Widget

```
<form action="..." method="post">
    @resubmission()
```

wheezy.web json and ajax

- Understands AJAX requests
  - redirect_for, see_other_for
    - Changes HTTP status code to 207 while preserving HTTP header Location
    - JavaScript to handle it

```
class SignInHandler(BaseHandler):
            ...
            def post(self):
                ...
                credential = Credential()
                if (not self.try_update_model(credential)
                        ...):
                    if self.request.ajax:
                        return self.json_response({'errors': self.errors})
                    return self.get(credential)
                ...
                return self.see_other_for('default')
```

wheezy.web internationalization

- GNU **gettext**

```
i18n/
    en/
        LC_MESSAGES/
            membership.po
            shared.po
            validation.po
```

- Easy Configurable

```
options.update({
    'translations_manager': TranslationsManager(
        directories=['i18n'],
        default_lang='en')
})
```

- Up to Per-Handler Translations

```
class MyHandler(BaseHandler):

   @attribute
   def translation(self):
       return self.translations['membership']
```

wheezy.web caching

- Cache Backends
  - Memory, Null, CacheClient
  - Integration with python-memcached, pylibmc
  - Namespace / Partition aware
  - Key Encoding (utf-8, base64, hash)
- Cache Dependency
  - A wire between cache items so they can be invalidated via a single operation
  - Simplify code necessary to manage dependencies in cache
  - Not related to any particular cache implementation
  - Can be used to invalidate items across different cache partitions / namespaces

wheezy.web caching

- Nothing faster **content cache**
  - HTTPCacheMiddleware + **CacheProfile**s
  - Precise control through Cache Dependency

```
cache_profile = CacheProfile('server', duration=15)
none_cache_profile = CacheProfile('none', no_store=True)

@response_cache(cache_profile)
def list_of_goods(request):
    ...
    response.dependency = CacheDependency('list_of_goods')
    return response

@response_cache(none_cache_profile)
def change_price(request):
    ...
    with cache_factory() as cache:
        dependency = CacheDependency('list_of_goods')
        dependency.delete(cache)
    return response
```

wheezy.web caching

- **CacheProfile**
  - location – cache **strategy** (none, server, client, both, public)
  - duration – time for the cache item to be cached
  - no_store – instructs state of no-store http response header
  - vary_query, vary_form, vary_environ – a list of items that should be included into **cache key**
  - namespace

wheezy.web testing

- Functional Tests
  - Black Box Testing for WSGI
  - Tested by feeding input and examining the output
  - Internal program structure is rarely considered
- Primary Actors:
  - Functional Page
  - Functional Mixin
  - Test Case

wheezy.web testing

- **Functional Page**
  - Asserts to prove the current content is related to given page
  - Fulfills a **micro** use case

```
class SignInPage(object):

    def __init__(self, client):
        assert '- Sign In</title>' in client.content
        assert AUTH_COOKIE not in client.cookies
        self.client = client
        self.form = client.form

    def signin(self, username, password):
        form = self.form
        form.username = username
        form.password = password
        self.client.submit(form)
        return self.client.form.errors()
```

wheezy.web testing

- **Functional Mixin**
  - High level **actor**
  - Fulfills use cases
  - Operates with several Functional Pages
  - Test Case can combine them to fulfill its goal

```
class MembershipMixin(object):

    def signin(self, username, password):
        client = self.client
        assert 200 == client.get('/en/signin')
        page = SignInPage(client)
        return page.signin(username, password)
```

wheezy.web testing

- **Test Case**
  - Can use several Functional Mixins
  - A set of conditions under which we can determine whether an application is working correctly or not

```
class SignInTestCase(unittest.TestCase, SignInMixin):

    def setUp(self):
        self.client = WSGIClient(wsgi_app)

    def test_validation_error(self):
        """ Ensure sigin page displays field validation errors.
        """
        errors = self.signin('', '')
        assert 2 == len(errors)
        assert AUTH_COOKIE not in self.client.cookies
```

wheezy.web testing

- **Benchmarks**
  - Any test cases

```
class BenchmarkTestCase(PublicTestCase):

    def runTest(self):
        """ Perform bachmark and print results.
        """
        p = Benchmark((
            self.test_home,
            self.test_about
            ), 1000)
        p.report('public', baselines={
                'test_home': 1.0,
                'test_about': 0.926
        })
```

  - Shows productivity gain

```
public: 2 x 1000
baseline throughput change target
  100.0%     839rps  +0.0% test_home
   96.2%     807rps  +3.9% test_about
```

wheezy.web resources

- Examples
  - Tutorial: http://packages.python.org/wheezy.web/tutorial.html
    - Source: https://bitbucket.org/akorn/wheezy.web/src/tip/demos/guestbook
  - Real World Demo: http://wheezy.pythonanywhere.com
    - Source: https://bitbucket.org/akorn/wheezy.web/src/tip/demos/template
- Documentation
    - Caching: http://packages.python.org/wheezy.caching
    - Core: http://packages.python.org/wheezy.core
    - HTML: http://packages.python.org/wheezy.html
    - HTTP: http://packages.python.org/wheezy.http
    - Routing: http://packages.python.org/wheezy.routing
    - Security: http://packages.python.org/wheezy.security
    - Template: http://packages.python.org/wheezy.template
    - Validation: http://packages.python.org/wheezy.validation
    - Web: http://packages.python.org/wheezy.web