pyramid_pystache Documentation

Release 0.2

Repoze Developers

May 26, 2015

Contents

CHAPTER 1

Overview

`pyramid_pystache` is a set of bindings that make templates written for the *Mustache* templating system work under the Pyramid web framework.

CHAPTER 2

Installation

Install using setuptools, e.g. (within a virtualenv):

```
$ $myvenv/bin/easy_install pyramid_pystache
```

CHAPTER 3

Setup

There are several ways to make sure that *pyramid_pystache* is active. They are completely equivalent:

1. Add pyramid_pystache to the *pyramid.includes* section of your applications main configuration section:

```
[app:main]
...
pyramid.includes = pyramid_pystache
```

2. Use the `includeme` function via `config.include`:

```
config.include('pyramid_pystache')
```

Once activated, files with the `.mustache` extension are considered to be *Mustache* templates.

CHAPTER 4

Using Mustache Templates

Once *pyramid_pystache* been activated `.mustache` templates can be loaded either by looking up names that would be found on the *Mustache* search path or by looking up an absolute asset specification (see Understanding Asset Specifications for more information).

Quick example 1. Look up a template named `foo.mustache` within the `templates` directory of a Python package named `mypackage`:

```
@view_config(renderer="mypackage:templates/foo.mustache)
def sample_view(request):
    return {'foo':1, 'bar':2}
```

Quick example 2. Look up a template named `foo.mustache` within the `templates` directory of the "current" Python package (the package in which this Python code is defined):

```
@view_config(renderer="templates/foo.mustache)
def sample_view(request):
    return {'foo':1, 'bar':2}
```

Quick example 3: manufacturing a response object using the result of `render()` (a string) using a Mustache template:

```
from pyramid.renderers import render
from pyramid.response import Response

def sample_view(request):
    result = render('mypackage:templates/foo.mustache',
                    {'foo':1, 'bar':2},
                    request=request)
    response = Response(result)
    response.content_type = 'text/plain'
    return response
```

Here's an example view configuration which uses a Mustache renderer registered imperatively:

```
# config is an instance of pyramid.config.Configurator

config.add_view('myproject.views.sample_view',
                renderer='myproject:templates/foo.mustache')
```

4.1 Mustache Templates

The language definition documentation for Mustache templates is available from the Mustache manual.

Given a *Mustache* template named `foo.mustache` in a directory in your application named `templates`, you can render the template as a renderer like so:

```
from pyramid.view import view_config

@view_config(renderer='templates/foo.mustache')
def my_view(request):
    return {'foo':1, 'bar':2}
```

When a Mustache renderer is used in a view configuration, the view must return a Response object or a Python *dictionary*. If the view callable with an associated template returns a Python dictionary, the named template will be passed the dictionary as its keyword arguments, and the template renderer implementation will return the resulting rendered template in a response to the user. If the view callable returns anything but a Response object or a dictionary, an error will be raised.

Before passing keywords to the template, the keyword arguments derived from the dictionary returned by the view are augmented. The callable object – whatever object was used to define the view – will be automatically inserted into the set of keyword arguments passed to the template as the `view` keyword. If the view callable was a class, the `view` keyword will be an instance of that class. Also inserted into the keywords passed to the template are `renderer_name` (the string used in the `renderer` attribute of the directive), `renderer_info` (an object containing renderer-related information), `context` (the context resource of the view used to render the template), and `request` (the request passed to the view used to render the template). `request` is also available as `req` in Pyramid 1.3+.

4.1.1 A Sample Mustache Template

Here's what a simple *Mustache* template used under Pyramid might look like:

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>{{project}} Application</title>
</head>
  <body>
     <h1>Welcome to <code>{{project}}</code>, an
      application generated by the <a
      href="http://docs.pylonsproject.org/projects/pyramid/current/"
     >pyramid</a> web
      application framework.</h1>
  </body>
</html>
```

The above template expects to find a `project` key in the set of keywords passed in to it via `render()` or `render_to_response()`.

4.2 Template Variables provided by Pyramid

Pyramid by default will provide a set of variables that are available within your templates, please see System Values Used During Rendering for more information about those variables.

4.3 Changing the Content-Type of a Mustache-Renderered Response

Here's an example of changing the content-type and status of the response object returned by a Mustache-rendered Pyramid view:

```
@view_config(renderer='foo.mustache')
def sample_view(request):
    request.response.content_type = 'text/plain'
    response.status_int = 204
    return response
```

See Varying Attributes of Rendered Responses for more information.

4.4 Unit Testing

When you are running unit tests, you will be required to use `config.include('pyramid_pystache')` to add `pyramid_pystache` so that its renderers are added to the config and can be used.:

```
from pyramid import testing
from pyramid.response import Response
from pyramid.renderers import render

# The view we want to test
def some_view(request):
    return Response(
        render('mypkg:templates/home.mustache', {'var': 'testing'})
    )

class TestViews(unittest.TestCase):
    def setUp(self):
        self.config = testing.setUp()
        self.config.include('pyramid_pystache')

    def tearDown(self):
        testing.tearDown()

    def test_some_view(self):
        from pyramid.testing import DummyRequest
        request = DummyRequest()
        response = some_view(request)
        # templates/home.mustache starts with the standard <html> tag
        self.assertTrue('<html' in response.body)
```

CHAPTER 5

More Information

5.1 Glossary

Mustache Mustache is a logic-free templating system implemented via Pystache

5.2 `pyramid_pystache` API

pyramid_pystache.**includeme**(*config*)
Adds renderers for .mustache

interface pyramid_pystache.interfaces.**IMustacheLookup**

CHAPTER 6

Reporting Bugs / Development Versions

Visit http://github.com/darrenlucas/pyramid_pystache to download development or tagged versions.

Visit http://github.com/darrenlucas/pyramid_pystache/issues to report bugs.

6.1 Indices and tables

- *Glossary*
- genindex
- modindex
- search

p

I

M

P

T