
Rhetoric Documentation

Release 0.1.9

Maxim Avanov

March 24, 2014

1	Why it is worth your while	3
2	Project premises	5
3	Installation	7
4	Integration with Django	9
5	Route Pattern Syntax	11
6	View Configuration Parameters	13
6.1	Non-Predicate Arguments	13
6.2	Predicate Arguments	13
7	Renderers	15
7.1	Built-in renderers	15
7.2	Varying Attributes of Rendered Responses	16
8	Predicates	17
9	@view_defaults Class Decorator	19
10	Sources	21
11	Authors	23
12	Changelog	25
13	Indices and tables	27

Status: **Beta, Unstable API.**

Naive implementation of Pyramid-like routes for Django projects.

Why it is worth your while

There's a great article on why Pyramid routing subsystem is so convenient for web developers - [Pyramid view configuration: Let me count the ways](#).

As a person who uses Pyramid as a foundation for his pet-projects, and Django - at work, I (the author) had a good opportunity to compare two different approaches to routing configuration provided by these frameworks. And I totally agree with the key points of the article - Pyramid routes are more flexible and convenient for developers writing RESTful services.

The lack of flexibility of standard Django url dispatcher motivated me to create this project. I hope it will be useful for you, and if you liked the idea behind Rhetoric URL Dispatcher, please consider [Pyramid Web Framework](#) for one of your future projects.

Project premises

- Rhetoric components try to follow corresponding Pyramid components whenever possible.
- Integration with django applications shall be transparent to existing code whenever possible.
- Performance of Rhetoric URL Dispatcher is worse than of the one of Pyramid, due to naivety of the implementation and limitations imposed by the compatibility with Django API.

Installation

Rhetoric is available as a PyPI package:

```
$ pip install Rhetoric
```

The package shall be compatible with Python2.7, and Python3.3 or higher.

Integration with Django

1. Replace `django.middleware.csrf.CsrfViewMiddleware` with `rhetoric.middleware.CsrfProtectedViewMiddleware` in your project's `MIDDLEWARE_CLASSES`:

```

1  # somewhere in a project_name.settings module
2
3  MIDDLEWARE_CLASSES = [
4      # ...
5      'rhetoric.middleware.CsrfProtectedViewDispatchMiddleware',
6      'django.middleware.csrf.CsrfViewMiddleware',
7      # ...
8  ]

```

2. Inside the project's root `urlpatterns` (usually `project_name.urls`):

```

1  from django.conf.urls import patterns, include, url
2  # ... other imports ...
3  from rhetoric import Configurator
4
5  # ... various definitions ...
6
7  urlpatterns = patterns('',
8      # ... a number of standard django url definitions ...
9  )
10
11 # Rhetorical routing
12 # -----
13 config = Configurator()
14 config.add_route('test.new.routes', '/test/new/routes/{param:[a-z]+}')
15 config.scan(ignore=[
16     # do not scan test modules included into the project tree
17     re.compile('^.*[.]?tests[.]*$').match,
18     # do not scan settings modules
19     re.compile('^project_name.settings[_]?[_a-z09]*$').match,
20 ])
21 urlpatterns.extend(config.django_urls())

```

3. Register views:

```

1  # project_name.some_app.some_module
2
3  from rhetoric import view_config
4
5

```

```
6 @view_config(route_name="test.new.routes", renderer='json')
7 def view_get(request, param):
8     return {
9         'Hello': param
10    }
11
12 @view_config(route_name="test.new.routes", renderer='json', request_method='POST')
13 def view_post(request, param):
14     return {
15         'Hello': 'POST'
16    }
```

4. From this point you can request `/test/new/routes/<param>` with different methods.

Route Pattern Syntax

Note: This section is copied from [Pyramid Docs](#), since Rhetoric provides the same pattern matching functionality.

The *pattern* used in route configuration may start with a slash character. If the pattern does not start with a slash character, an implicit slash will be prepended to it at matching time. For example, the following patterns are equivalent:

```
{foo}/bar/baz
```

and:

```
/ {foo}/bar/baz
```

A pattern segment (an individual item between / characters in the pattern) may either be a literal string (e.g. `foo`) or it may be a replacement marker (e.g. `{foo}`) or a certain combination of both. A replacement marker does not need to be preceded by a / character.

A replacement marker is in the format `{name}`, where this means “accept any characters up to the next slash character and use this as the input parameter for a view callable.

A replacement marker in a pattern must begin with an uppercase or lowercase ASCII letter or an underscore, and can be composed only of uppercase or lowercase ASCII letters, underscores, and numbers. For example: `a`, `a_b`, `_b`, and `b9` are all valid replacement marker names, but `0a` is not.

A `matchdict` is the dictionary representing the dynamic parts extracted from a URL based on the routing pattern. It is available as `request.matchdict`. For example, the following pattern defines one literal segment (`foo`) and two replacement markers (`baz`, and `bar`):

```
foo/{baz}/{bar}
```

The above pattern will match these URLs, generating the following `matchdicts`:

```
foo/1/2      -> {'baz':u'1', 'bar':u'2'}
foo/abc/def   -> {'baz':u'abc', 'bar':u'def'}
```

It will not match the following patterns however:

```
foo/1/2/      -> No match (trailing slash)
bar/abc/def    -> First segment literal mismatch
```

Replacement markers can optionally specify a regular expression which will be used to decide whether a path segment should match the marker. To specify that a replacement marker should match only a specific set of characters as defined by a regular expression, you must use a slightly extended form of replacement marker syntax. Within braces, the replacement marker name must be followed by a colon, then directly thereafter, the regular expression. The *default* regular expression associated with a replacement marker `[^/]+` matches one or more characters which are not a slash. For example, under the hood, the replacement marker `{foo}` can more verbosely be spelled as `{foo:[^/]+}`. You

can change this to be an arbitrary regular expression to match an arbitrary sequence of characters, such as `{foo:\d+}` to match only digits.

It is possible to use two replacement markers without any literal characters between them, for instance `{foo}{bar}`. However, this would be a nonsensical pattern without specifying a custom regular expression to restrict what each marker captures.

Segments must contain at least one character in order to match a segment replacement marker. For example, for the URL `/abc/`:

- `/abc/{foo}` will not match.
- `{foo}/` will match.

View Configuration Parameters

Note: This section is partly copied from the [Pyramid documentation](#), since Rhetoric provides almost the same functionality.

6.1 Non-Predicate Arguments

`renderer`

6.2 Predicate Arguments

`route_name`

`request_method`

`api_version`

New in version 0.1.7.

Available patterns:

Renderers

Note: This section is copied from the [Pyramid Renderers documentation](#), since Rhetoric provides almost the same rendering functionality.

7.1 Built-in renderers

7.1.1 `string`: String Renderer

The `string` renderer is a renderer which renders a view callable result to a string. If a view callable returns a non-Response object, and the `string` renderer is associated in that view's configuration, the result will be to run the object through the Python `str` function to generate a string.

7.1.2 `json`: JSON Renderer

The `json` renderer renders view callable results to *JSON*. By default, it passes the return value through the `django.core.serializers.json.DjangoJSONEncoder`, and wraps the result in a response object. It also sets the response content-type to `application/json`.

Here's an example of a view that returns a dictionary. Since the `json` renderer is specified in the configuration for this view, the view will render the returned dictionary to a JSON serialization:

```
from rhetoric import view_config

@view_config(renderer='json')
def hello_world(request):
    return {'content': 'Hello!'}
```

The body of the response returned by such a view will be a string representing the JSON serialization of the return value:

```
{"content": "Hello!"}
```

7.1.3 `.html`: Django Template Renderer

The `.html` template renderer renders views using the standard Django template language. When used, the view must return a `HttpResponse` object or a Python *dictionary*. The dictionary items will then be used as the template context objects.

7.2 Varying Attributes of Rendered Responses

Note: This section is partly copied from the [Pyramid Renderers documentation](#), since Rhetoric provides almost the same API.

New in version 0.1.8.

Before a response constructed by a *renderer* is returned to Django, several attributes of the request are examined which have the potential to influence response behavior.

View callables that don't directly return a response should use the API of the `django.http.HttpResponse` attribute available as `request.response` during their execution, to influence associated response behavior.

For example, if you need to change the response status from within a view callable that uses a renderer, assign the `status_code` attribute to the `response` attribute of the request before returning a result:

```
1 from rhetoric import view_config
2
3 @view_config(name='dashboard', renderer='dashboard.html')
4 def myview(request):
5     request.response.status_code = 404
6     return {'URL': request.get_full_path() }
```

Note that mutations of `request.response` in views which return a `HttpResponse` object directly will have no effect unless the response object returned *is* `request.response`. For example, the following example calls `request.response.set_cookie`, but this call will have no effect, because a different `Response` object is returned.

```
1 from django.http import HttpResponse
2
3 def view(request):
4     request.response.set_cookie('abc', '123') # this has no effect
5     return HttpResponse('OK') # because we're returning a different response
```

If you mutate `request.response` and you'd like the mutations to have an effect, you must return `request.response`:

```
1 def view(request):
2     request.response.set_cookie('abc', '123')
3     return request.response
```

7.3 Request properties

`request.json_body` - http://docs.pylonsproject.org/projects/pyramid/en/latest/api/request.html#pyramid.request.Request.json_body

Predicates

@view_defaults Class Decorator

Note: This section is copied from [Pyramid Docs](#), since Rhetoric provides the same functionality.

New in version 0.1.7.

If you use a class as a view, you can use the `rhetoric.view.view_defaults` class decorator on the class to provide defaults to the view configuration information used by every `@view_config` decorator that decorates a method of that class.

For instance, if you’ve got a class that has methods that represent “REST actions”, all which are mapped to the same route, but different request methods, instead of this:

```

1  from rhetoric import view_config
2
3  class RESTView(object):
4      def __init__(self, request, *args, **kw):
5          self.request = request
6
7      @view_config(route_name='rest', request_method='GET', renderer='json')
8      def get(self):
9          return {'method': 'GET'}
10
11     @view_config(route_name='rest', request_method='POST', renderer='json')
12     def post(self):
13         return {'method': 'POST'}
14
15     @view_config(route_name='rest', request_method='DELETE', renderer='json')
16     def delete(self):
17         return {'method': 'DELETE'}
```

You can do this:

```

1  from rhetoric import view_config
2  from rhetoric import view_defaults
3
4  @view_defaults(route_name='rest', renderer='json')
5  class RESTView(object):
6      def __init__(self, request, *args, **kw):
7          self.request = request
8
9      @view_config(request_method='GET')
10     def get(self):
11         return {'method': 'GET'}
```

```
13     @view_config(request_method='POST')
14     def post(self):
15         return {'method': 'POST'}
16
17     @view_config(request_method='DELETE')
18     def delete(self):
19         return {'method': 'DELETE'}
```

In the above example, we were able to take the `route_name='rest'` and `renderer='json'` arguments out of the call to each individual `@view_config` statement, because we used a `@view_defaults` class decorator to provide the argument as a default to each view method it possessed.

Arguments passed to `@view_config` will override any default passed to `@view_defaults`.

Sources

Rhetoric is licensed under the [MIT License](#).

We use GitHub as a primary code repository - <https://github.com/avanov/Rhetoric>

Authors

Rhetoric package was created by Maxim AvanoV.

- [GitHub profile](#).
- [Google+ profile](#).

Changelog

- 0.1.9
 - Added support for the `request.json_body` property.
- 0.1.8
 - Added support for the `request.response` API.
- 0.1.7
 - Added support for the `api_version` predicate.
 - Added the `view_defaults` decorator.
- 0.1.5
 - Feature: added support for `decorator` argument of `view_config`.
- 0.1.4
 - Feature: added support for custom renderers.
- 0.1.2
 - [Bugfix #2]: resolved race condition in `rhetoric.view.ViewCallback`.
 - [API]: `rhetoric.middleware.UrlResolverMiddleware` was renamed to `rhetoric.middleware.CsrfProtectedViewDispatchMiddleware`.
 - [Django integration]: `rhetoric.middleware.CsrfProtectedViewDispatchMiddleware` should now completely substitute `django.middleware.csrf.CsrfViewMiddleware` in `MIDDLEWARE_CLASSES`.
- 0.1.0 - initial PyPI release. Early development, unstable API.

Indices and tables

- *genindex*
- *modindex*
- *search*