rivr Documentation

Release 0.8.0

Kyle Fuller

Oct 13, 2020

Contents

rivr is a BSD Licensed WSGI web framework, written in Python.

```
import rivr

def hello_world(request):
    return rivr.Response('<html><body>Hello world!</body></html>')

if __name__ == '__main__':
    rivr.serve(hello_world)
```

Contents:

CHAPTER 1

Release History

1.1 Master

1.1.1 Breaking

- *rivr.templates* has been removed, please migrate to *rivr-jinja*.
- *RESTView* has been removed. Either vendor the old RESTView, or migrate to *rivr-rest*.

1.2 0.6.0

1.2.1 Views

- Generic View class now has a default implementation of *HEAD*.
- Generic View class now has a default implementation of *OPTIONS*, which simply returns the supported HTTP methods.

1.2.2 Contrib

- Completely removed the MongoDB module. This hasn't been maintained in a while, if anyone still uses it or is interested in maintaining it. It should be moved to a separate module.
- Moved jinja support out into a separate project *rivr-jinja*.
- Removed the BrowserID module.

CHAPTER 2

Views

A view is a simple function which takes a request and returns a response. The response is then sent to the user, it can be a simple HTML page, a request or anything.

```
import rivr

def example_view(request):
    return rivr.Response('<html><body>Hello world!</body></html>')
```

2.1 Class-based views

A class-based view is a class which is callable, so it acts just like a normal function based view. rivr provides a base view class which allows you to implement methods for the different HTTP methods it wants to handle. For exaple:

```
class BasicView(View):
    def get(self, request):
        return rivr.Response('Get request!')

    def post(self, request):
        return rivr.Response('Post request!')
```

rivr also provides a other views such as *RedirectView*.

class rivr.views.**View**(***kwargs*)

classmethod as_view(***kwargs*)

This method will return a callable which will generate a new instance of the class passing it the kwargs passed to it.

Usage:

```
view = View.as_view()
response = view(request)
```

class rivr.views.**RedirectView**(***kwargs*)
The redirect view will redirect on any GET requests.

Members url, permanent

get_redirect_url(*self*, ***kwargs*)
Return the URL we should redirect to

CHAPTER 3

Request and Response objects

3.1 HTTP Message

class rivr.http.message.**HTTPMessage**(*headers: Union[wsgiref.headers.Headers, Dict[str, str], None] = None*)

Members headers, content_type, content_length

3.2 Request

class rivr.http.request.**Query**(*query: Union[str, Dict[str, str], None] = None*)

__str__() → str

```
>>> query = Query({'category': 'fruits'})
>>> str(query)
'category=fruits'
```

__len__() → int

```
>>> query = Query('category=fruits&limit=10')
>>> len(query)
2
```

__contains__(*name: str*) → bool

```
>>> query = Query('category=fruits')
>>> 'category' in query
True
```

__getitem__(*name: str*) → Optional[str]

```
>>> query = Query('category=fruits')
>>> query['category']
'fruits'
```

class rivr.http.**Request**(*path: str = '/', method: str = 'GET', query: Optional[Dict[str, str]] = None, headers: Optional[Dict[str, str]] = None, body: Union[bytes, IO[bytes], None] = None*)

A request is an object which represents a HTTP request. You wouldn't normally create a request yourself but instead be passed a request. Each view gets passed the clients request.

Members method, path, query, headers, cookies, body

3.3 Response

class rivr.http.**Response**(*content: Union[str, bytes] = '', status: Optional[int] = None, content_type: Optional[str] = 'text/html; charset=utf8'*)

Response is an object for describing a HTTP response. Every view is responsible for either returning a response or raising an exception.

status_code = 200

The HTTP status code for the response.

class rivr.http.**ResponseNoContent**(*content: Union[str, bytes] = '', status: Optional[int] = None, content_type: Optional[str] = 'text/html; charset=utf8'*)

A response that uses the 204 status code to indicate no content.

class rivr.http.**ResponseRedirect**(*redirect_to: str*)

Acts just like a ResponseRedirect, but uses a 302 status code. It takes a URL to redirect the user to.

url

A property that returns the URL for the redirect.

class rivr.http.**ResponsePermanentRedirect**(*redirect_to: str*)

Acts just like a ResponseRedirect, but uses a 301 status code.

class rivr.http.**ResponseNotFound**(*content: Union[str, bytes] = '', status: Optional[int] = None, content_type: Optional[str] = 'text/html; charset=utf8'*)

Acts just like a Response, but uses a 404 status code.

class rivr.http.**ResponseNotModified**(*content: Union[str, bytes] = '', status: Optional[int] = None, content_type: Optional[str] = 'text/html; charset=utf8'*)

Acts just like a Response, but uses a 304 status code.

class rivr.http.**ResponseNotAllowed**(*permitted_methods: List[str]*)

A response that uses the 405 status code and takes a list of permitted HTTP methods.

CHAPTER 4

Middleware

class rivr.middleware.**Middleware**(***kwargs*)

classmethod wrap(*view*, ***kwargs*)

The wrap method allows you to wrap a view calling the middleware's process_request and process_response before and after the view.

If the view raises an exception, the process_exception method will be called.

Example:

```
view = Middleware.wrap(view)
response = view(request)
```

process_request(*self*, *request*)

This method is called before the view on each request. This method should either return a response or None. If it returns a response, then the middleware will not call the view. If it returns None, then we will call the view.

process_response(*self*, *request*, *response*)

This method will take the response, either from process_request or the view. This method will always be called for each request unless there is an exception.

This method must return a response, this can either be the response passed to it, or a completely new response.

class rivr.middleware.**MiddlewareController**(**middleware*)

The middleware controller allows you to wrap a view in multiple middleware.

Example usage:

```
view = MiddlewareController.wrap(view,
    FirstMiddleware(),
    SecondMiddleware()
)
```

(continues on next page)

(continued from previous page)

```
response = view(request)
```

CHAPTER 5

Server

You can serve rivr with either the build in development server, or you can use any WSGI compatible server such as gunicorn.

5.1 Development Server

By default, the development server can be used with any rivr callable view, this includes middleware and views.

`rivr.`**`serve`**(*handler: Callable[[rivr.http.request.Request], rivr.http.response.Response], host: str = 'localhost', port: int = 8080, debug: bool = True*)

Starts a developent server on the local machine. By default, the server runs on port 8080 on localhost. You can pass in a different hostname and/or IP using the keyword arguments.

The development server automatically returns pretty HTML error pages, this can be turned off by setting the debug keyword to *False*.

```
import rivr

def example_view(request):
    return rivr.Response('<html><body>Hello world!</body></html>')

if __name__ == '__main__':
    rivr.serve(example_view)
```

5.2 WSGI Server

It's really simple to use a WSGI server, you can use rivr's WSGIHandler to wrap your rivr callable view.

```
import rivr
from rivr.wsgi import WSGIHandler
```

(continues on next page)

(continued from previous page)

```
def example_view(request):
    return rivr.Response('<html><body>Hello world!</body></html>')

wsgi = WSGIHandler(example_view)
```

Then you can simply point your WSGI server to the wsgi method. For example, to do this with gunicorn you can run the following:

```
$ gunicorn example_wsgi:wsgi
```

CHAPTER 6

Router

rivr comes with a powerful regex based path router similar to the url patterns system in Django.

Example usage:

```
import rivr

router = rivr.Router()

@router.register(r'^$')
def index(request):
    return rivr.Response('Hello world')

@router.register(r'example/$')
def example(request):
    return rivr.Response('Example')
```

Similar to Django, it will also pull out pattern matches from the regex and feed them as arguments and keyword-arguments to your view. For example:

```
@router.register(r'^(?P<username>[-\w]+)/$')
def index(request, username):
    return rivr.Response('Hello %s' % username)
```

class rivr.router.**Router**(**urls*)

__init__(**urls*)

Router takes URLs which you can register on creation.

Example:

```
router = rivr.Router(
    (r'^$', index),
    (r'^test/$', test),
)
```

append_slash = True
When append_slash is True, if the request URL does not match any patterns in the router and it doesn't end in a slash. The router will HTTP redirect any issues to the same URL with a slash appended.

register(*t)
Register a URL pattern with a view. This can either be used as a decorator, or it can be used as a method with a view.

Decorator Example:

```
@router.register(r'^$')
def view(request):
    return Response()
```

View Example:

```
router.register(r'^$', view)
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

Symbols

A

G

H

M

P

Q

R

S

U

V

W