restfulgrok Documentation

Release 1.0.2

Espen Angell Kristiansen

Sep 27, 2017

Contents

1 Getting started **3**

2 Extending and styling the HtmlContentType view **5**

3 Documentation **7**
- 3.1 restfulgrok API 7
- 3.2 Add custom content-type/mimetype 11

4 Indices and tables **13**

Python Module Index **15**

`restfulgrok` provides a very simple RESTful view mixin for `five.grok.View`. It is not meant to be a full-featured REST library, only a quick solution for simple REST APIs using `five.grok.View`.

Features:

- **Content negotiation:**
  - Assumes same input and out mimetype (simplifies the implementation)
  - Can be specified in a GET parameter (E.g.: `?mimetype=application/yaml`)
  - Can be specified use HTTP ACCEPT header.
  - **Supports:**
    * JSON
    * YAML
    * HTML (read only)
    * ... *Add custom content-type/mimetype*
- HTTP response helpers for common response types.

CHAPTER 1

Getting started

Create a class that inherit from GrokRestViewMixin:

```
from restfulgrok.fancyhtmlview import GrokRestViewWithFancyHtmlMixin


class ExampleRestViewMixin(GrokRestViewWithFancyHtmlMixin):
    def handle_get(self):
        # Return something that can be encoded by JSON and YAML
        return {'hello': 'world'}

    def handle_post(self):
        try:
            # Decode request body as a dict (JSON or YAML)
            request_data = self.get_requestdata_dict()
        except ValueError, e:
            # Did not get a dict
            return self.response_400_bad_request({'error': str(e)})
        else:
            # save to database or something....
            # --- not included in example ---

            # Respond with 201 status and the request_data
            # NOTE: If you just return normally, 200 response status is used
            return self.response_201_created(request_data)

    def handle_put(self):
        try:
            # Decode request body as a dict (JSON or YAML)
            request_data = self.get_requestdata_dict()
        except ValueError, e:
            # Did not get a dict
            return self.response_400_bad_request({'error': str(e)})
        else:
            data = request_data.copy() # pretend we got this from a database
            data['last-modified'] = 'now' # Update some data
```

```
            # would save here if this was real...

            # NOTE: If you just return normally, 200 response status is used
            return {'Updated': 'yes',
                    'result': data}
```

And a testcase:

```
from unittest import TestCase
from restfulgrok.mock import MockResponse
from restfulgrok.mock import MockRequest
from restfulgrok.mock import MockContext

from example import ExampleRestViewMixin


class MockExampleRestMixin(ExampleRestViewMixin):
    def __init__(self, method='GET', body=''):
        self.response = MockResponse()
        self.request = MockRequest(method, body=body)
        self.context = MockContext()

class TestExampleRestMixin(TestCase):
    def test_get(self):
        result = MockExampleRestMixin('GET').handle()
        self.assertEquals(result, {'hello': 'world'})

    def test_post(self):
        import json
        data = {'a': 'test'}
        result = MockExampleRestMixin('POST', body=json.dumps(data)).handle()
        self.assertEquals(result, {'a': 'test'})

    def test_put(self):
        import json
        data = {'a': 'test'}
        result = MockExampleRestMixin('PUT', body=json.dumps(data)).handle()
        self.assertEquals(result,
                          {'Updated': 'yes',
                           'result': {'a': 'test',
                                      'last-modified': 'now'}})


if __name__ == '__main__':
    import unittest
    unittest.main()
```

And finally use the mixin to create a `grok.View`:

```
from five import grok
class ExampleRestView(ExampleRestViewMixin, grok.View):
    grok.context(IMyInterface)
    grok.name('rest')
```

CHAPTER 2

Extending and styling the HtmlContentType view

The html provided with *restfulgrok.fancyhtmlview.HtmlContentType* does not have a stylesheet, however it is designed to work with Bootstrap. Just override the template, and override the `head_extra` block. For example:

1. Create your own html content type (example assumes your app is `my.package`):

```
from jinja2 import Environment, PrefixLoader, PackageLoader
class MyHtmlContentType(HtmlContentType):
    template_name = 'fancyhtmlview.jinja.html'
    template_environment = Environment(loader = PrefixLoader({
        'restfulgrok': PackageLoader('restfulgrok'),
        'mypackage': PackageLoader('my.package')
    }))
```

2. Create `my/package/templates` and `my/package/staticfiles`.

3. Add static directory to `configure.zcml`:

```
<browser:resourceDirectory
    name="my.package"
    directory="staticfiles"
    />
```

4. Create your own template, `my/package/templates/view.jinja.html`, extending the one from `restfulgrok`:

```
{% extends "restfulgrok/fancyhtmlview.jinja.html" %}
{% block head_extra %}
    <link rel="stylesheet/less" href="++resource++my.package/bootstrap/less/
↪bootstrap.less">
    <script src="++resource++my.package/less-1.3.0.min.js"></script>
{% endblock %}
```

CHAPTER 3

Documentation

restfulgrok API

restfulgrok.view

exception `restfulgrok.view.`**`CouldNotDetermineContentType`**(*querystring_error*, *acceptheader_error*, *acceptable_mimetypes*)

Bases: `exceptions.Exception`

Raised when *`GrokRestViewMixin.get_content_type()`* fails to detect a supported content-type.

class `restfulgrok.view.`**`GrokRestViewMixin`**

Bases: `object`

Mix-in class for `five.grok.View`.

`add_attachment_header`()

Adds Content-Disposition header for filedownload if "downloadfile=yes" is in the querystring (request.get).

`authorize`()

Called by *`render()`* to authorize the user before calling *`handle()`*.

The permissions required for each method is defined in *`permissions`*.

Raises `AccessControl.Unauthorized` – If the current user do not have permission to perform the requested method.

`content_types` = <restfulgrok.contenttype.ContentTypesRegistry object>

A `ContentTypesRegistry` object containing all content-types supported by the API.

`create_response`(*status*, *statusmsg*, *body*)

Respond with the given `status` and `statusmessage`, with `body` as response body.

`decode_input_data`(*rawdata*)

Decode the given `rawdata`.

Raises *restfulgrok.contenttype.ContentTypeLoadError* – If `rawdata` can not be decoded.

encode_output_data(*pydata*)
Encode the given python datastructure.

Raises *restfulgrok.contenttype.ContentTypeDumpError* – If `pydata` can not be encoded.

get_content_type()
Detect input/output content type.

get_requestdata()
Decode the body of the request using *decode_input_data()*, and return the decoded data.

get_requestdata_dict()
Just like *get_requestdata()*, however, the `ValueError` is raised if the decoded data is not a `dict`.

get_requestmethod()
Get the request method as lowercase string.

handle()
Takes care of all the logic for *render()*, except that it does not *encode_output_data()* the response data, and it does not do authorization. This makes this method very suitable for use in tests or custom render() implementations.

handle_delete()
Override in subclasses. Defaults to *response_405_method_not_allowed()*.

handle_get()
Override in subclasses. Defaults to *response_405_method_not_allowed()*.

handle_head()
Override in subclasses. Defaults to *response_405_method_not_allowed()*.

handle_options()
Override in subclasses. Defaults to *response_405_method_not_allowed()*.

handle_post()
Override in subclasses. Defaults to *response_405_method_not_allowed()*.

handle_put()
Override in subclasses. Defaults to *response_405_method_not_allowed()*.

permissions = {'put': 'Modify portal content', 'default': 'Modify portal content', 'post': 'Add portal content', 'get': '
Map of request method to permission. You should have one (lowercase) key for each request method in *supported_methods*, or a "default" key defining a default permission.

render()
Called to render the view. Uses *handle()* to handle all the logic and *encode_output_data()* to encode the response from *handle()*.

response_201_created(*body*)
Run `self.response.setStatus(201, 'Created')` and return body.

response_400_bad_request(*body*)
Respond with 400 Bad Request, and the `body` parameter as response body.

response_401_unauthorized(*error='Unauthorized'*)
Respond with 401 Unauthorized, and `{'error': 'Unauthorized'}` as body.

response_405_method_not_allowed()
Respond with 405 Method Not Allowed.

set_contenttype_header(*mimetype=None*)
Set the content type header. Called by *handle()*, and may be overridden.

supported_methods = ['get', 'post', 'put', 'delete', 'options', 'head']
List of supported HTTP request methods in lowercase. Defaults to `['get', 'post', 'put', 'delete', 'options', 'head']`. You do not have to override this to disallow any of those request methods since their default `handle_<method>()` implementations responds with *405 Method Not Allowed*. However, if you implement other methods, such as TRACE, you need to add them to the list.

restfulgrok.fancyhtmlview

class restfulgrok.fancyhtmlview.**GrokRestViewWithFancyHtmlMixin**
Bases: *restfulgrok.view.GrokRestViewMixin*

Adds *HtmlContentType* to `content_types`.

class restfulgrok.fancyhtmlview.**HtmlContentType**
Bases: *restfulgrok.contenttype.ContentType*

XHTML content type. Provides a dumps-method that uses a jinja2-template to generate a bootrap-styled HTML-document which is suitable as a default view for a REST API.

error_template_name = 'restfulgrok/errorview.jinja.html'
jinja2 template name for the `errorview()`.

classmethod get_previewdata(*pydata*)
Get the data that should be added to the data preview box.

Returns A string containing the data.

classmethod get_template_data(*pydata*, *view*)
Get the template data.

Returns Template data.

Return type dict

html_brandingtitle = 'REST API'
Variable forwarded to the template as `brandingtitle`.

html_heading = 'REST API'
Variable forwarded to the template as `heading`.

html_title = 'REST API'
Variable forwarded to the template as `title`.

template_environment = <jinja2.environment.Environment object>
The `jinja2.Environment`

template_name = 'restfulgrok/fancyhtmlview.jinja.html'
jinja2 template name for the normal html view (not for errors)

restfulgrok.contenttype

class restfulgrok.contenttype.**ContentType**
Bases: `object`

Superclass for all content-types for the *ContentTypesRegistry*.

description = ''
A short description for users of the content-type.

classmethod dumps (*pydata*, *view*)
Dump `pydata` to a string and return the string.

Parameters

- **pydata** – The python data to encode.
- **view** – A `GrokRestViewMixin` instance.

extension = None
Extension for files of this type. Must be set in subclasses.

classmethod loads (*rawdata*, *view*)
Load the `rawdata` bytestring and return it as a decoded pyton object.

Parameters

- **rawdata** – The bytestring to decode.
- **view** – A `GrokRestViewMixin` instance.

mimetype = None
The mimetype of this content type. Must be set in subclasses.

exception `restfulgrok.contenttype.`**ContentTypeDumpError**
Bases: *restfulgrok.contenttype.ContentTypeError*

Raised when *ContentType.dumps()* fails.

exception `restfulgrok.contenttype.`**ContentTypeError**
Bases: `exceptions.Exception`

Superclass for *ContentType* errors.

exception `restfulgrok.contenttype.`**ContentTypeLoadError**
Bases: *restfulgrok.contenttype.ContentTypeError*

Raised when *ContentType.loads()* fails.

class `restfulgrok.contenttype.`**ContentTypesRegistry** (**content_types*)
Bases: `object`

Registry of *ContentType* objects.

add (*content_type*)
Add the given `content_type` to the registry. They are added indexed by their `mimetype`, so adding multiple content-types with the same mimetype will only add the last one.

addmany (**content_types*)
Run *add()* for each item in `content_types`.

aslist ()
Return list of :class:‘ContentType‘s in the registry.

negotiate_accept_header (*acceptheader*)
Parse the HTTP accept header and find any acceptable mimetypes from the registry.

Returns An acceptable mimetype, or `None` if no acceptable mimetype is found.

Return type str

class `restfulgrok.contenttype.`**JsonContentType**
Bases: *restfulgrok.contenttype.ContentType*

JSON content type. Implements both loads and dumps.

class restfulgrok.contenttype.**YamlContentType**
Bases: *restfulgrok.contenttype.ContentType*

YAML content type. Implements both loads and dumps.

restfulgrok.mock

Mock classes to simplify testing. See the sourcecode (or the *source* links below).

class restfulgrok.mock.**MockContext**(*parentnode=None*, *id=None*)
Bases: object

getParentNode()

class restfulgrok.mock.**MockRequest**(*method='GET'*, *body=''*, *getdata={}*, *headers={'Accept': 'application/json'}*)
Bases: object

get(*key*, *default=None*)

getHeader(*header*)

class restfulgrok.mock.**MockResponse**
Bases: object

getStatus()

setHeader(*header*, *value*)

setStatus(*code*, *msg*)

class restfulgrok.mock.**MockRestView**(*request=None*, *response=<restfulgrok.mock.MockResponse object>*, *context=None*)
Bases: *restfulgrok.view.GrokRestViewMixin*

class restfulgrok.mock.**MockRestViewWithFancyHtml**(*request=None*, *response=<restfulgrok.mock.MockResponse object>*, *context=None*)
Bases: *restfulgrok.fancyhtmlview.GrokRestViewWithFancyHtmlMixin*

Add custom content-type/mimetype

Content-types are defined as subclasses of *restfulgrok.contenttype.ContentType*. Take look at its docs, and the sourcecode of *restfulgrok.contenttype.JsonContentType*:

```
class JsonContentType(ContentType):
    """
    JSON content type. Implements both loads and dumps.
    """
    mimetype = 'application/json'
    extension = 'json'
    description = json_description

    @classmethod
    def dumps(cls, pydata, view=None):
        try:
            return json.dumps(pydata, indent=2)
        except TypeError, e:
            raise ContentTypeDumpError(str(e))
```

```
        except ValueError, e:
            raise ContentTypeDumpError(str(e))

    @classmethod
    def loads(cls, rawdata, view=None):
        try:
            return json.loads(rawdata)
        except TypeError, e:
            raise ContentTypeLoadError(str(e))
        except ValueError, e:
            raise ContentTypeDumpError(str(e))
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

restfulgrok.contenttype, 9
restfulgrok.fancyhtmlview, 9
restfulgrok.mock, 11
restfulgrok.view, 7

Index

A

add() (restfulgrok.contenttype.ContentTypesRegistry method), 10
add_attachment_header() (restfulgrok.view.GrokRestViewMixin method), 7
addmany() (restfulgrok.contenttype.ContentTypesRegistry method), 10
aslist() (restfulgrok.contenttype.ContentTypesRegistry method), 10
authorize() (restfulgrok.view.GrokRestViewMixin method), 7

C

content_types (restfulgrok.view.GrokRestViewMixin attribute), 7
ContentType (class in restfulgrok.contenttype), 9
ContentTypeDumpError, 10
ContentTypeError, 10
ContentTypeLoadError, 10
ContentTypesRegistry (class in restfulgrok.contenttype), 10
CouldNotDetermineContentType, 7
create_response() (restfulgrok.view.GrokRestViewMixin method), 7

D

decode_input_data() (restfulgrok.view.GrokRestViewMixin method), 7
description (restfulgrok.contenttype.ContentType attribute), 9
dumps() (restfulgrok.contenttype.ContentType class method), 10

E

encode_output_data() (restfulgrok.view.GrokRestViewMixin method), 8
error_template_name (restfulgrok.fancyhtmlview.HtmlContentType attribute), 9
extension (restfulgrok.contenttype.ContentType attribute), 10

G

get() (restfulgrok.mock.MockRequest method), 11
get_content_type() (restfulgrok.view.GrokRestViewMixin method), 8
get_previewdata() (restfulgrok.fancyhtmlview.HtmlContentType class method), 9
get_requestdata() (restfulgrok.view.GrokRestViewMixin method), 8
get_requestdata_dict() (restfulgrok.view.GrokRestViewMixin method), 8
get_requestmethod() (restfulgrok.view.GrokRestViewMixin method), 8
get_template_data() (restfulgrok.fancyhtmlview.HtmlContentType class method), 9
getHeader() (restfulgrok.mock.MockRequest method), 11
getParentNode() (restfulgrok.mock.MockContext method), 11
getStatus() (restfulgrok.mock.MockResponse method), 11
GrokRestViewMixin (class in restfulgrok.view), 7
GrokRestViewWithFancyHtmlMixin (class in restfulgrok.fancyhtmlview), 9

H

handle() (restfulgrok.view.GrokRestViewMixin method), 8
handle_delete() (restfulgrok.view.GrokRestViewMixin method), 8

handle_get() (restfulgrok.view.GrokRestViewMixin method), 8
handle_head() (restfulgrok.view.GrokRestViewMixin method), 8
handle_options() (restfulgrok.view.GrokRestViewMixin method), 8
handle_post() (restfulgrok.view.GrokRestViewMixin method), 8
handle_put() (restfulgrok.view.GrokRestViewMixin method), 8
html_brandingtitle (restfulgrok.fancyhtmlview.HtmlContentType attribute), 9
html_heading (restfulgrok.fancyhtmlview.HtmlContentType attribute), 9
html_title (restfulgrok.fancyhtmlview.HtmlContentType attribute), 9
HtmlContentType (class in restfulgrok.fancyhtmlview), 9

J

JsonContentType (class in restfulgrok.contenttype), 10

L

loads() (restfulgrok.contenttype.ContentType class method), 10

M

mimetype (restfulgrok.contenttype.ContentType attribute), 10
MockContext (class in restfulgrok.mock), 11
MockRequest (class in restfulgrok.mock), 11
MockResponse (class in restfulgrok.mock), 11
MockRestView (class in restfulgrok.mock), 11
MockRestViewWithFancyHtml (class in restfulgrok.mock), 11

N

negotiate_accept_header() (restfulgrok.contenttype.ContentTypesRegistry method), 10

P

permissions (restfulgrok.view.GrokRestViewMixin attribute), 8

R

render() (restfulgrok.view.GrokRestViewMixin method), 8
response_201_created() (restfulgrok.view.GrokRestViewMixin method), 8
response_400_bad_request() (restfulgrok.view.GrokRestViewMixin method), 8
response_401_unauthorized() (restfulgrok.view.GrokRestViewMixin method), 8
response_405_method_not_allowed() (restfulgrok.view.GrokRestViewMixin method), 8
restfulgrok.contenttype (module), 9
restfulgrok.fancyhtmlview (module), 9
restfulgrok.mock (module), 11
restfulgrok.view (module), 7

S

set_contenttype_header() (restfulgrok.view.GrokRestViewMixin method), 8
setHeader() (restfulgrok.mock.MockResponse method), 11
setStatus() (restfulgrok.mock.MockResponse method), 11
supported_methods (restfulgrok.view.GrokRestViewMixin attribute), 9

T

template_environment (restfulgrok.fancyhtmlview.HtmlContentType attribute), 9
template_name (restfulgrok.fancyhtmlview.HtmlContentType attribute), 9

Y

YamlContentType (class in restfulgrok.contenttype), 10