pyrs-resource Documentation

Release 0.3.0

Csaba Palankai

October 02, 2015

Project homepage: https://github.com/palankai/pyrs-resource
Documentation: yy‘<http://pyrs-resource.readthedocs.org>‘_
Issue tracking: https://github.com/palankai/pyrs-resource/issues

CHAPTER 1

What is this package for

In the python world there are many RESTFul framework. Some of them based on Django others are based on Flask. I've tried some but I had the feeling, I want to learn one, the use with Django or Flask or even Odoo. And I don't mention sometimes I found them not flexible enough. So, I've decided write my own independent framework what anybody can use in at least the mentioned 3 different worlds.

CHAPTER 2

Nutshell (notice that, it would be the achivement)

```
from pyrs import resource
from pyrs.resource import GET

class UserResouce:

    @GET(response=ArrayOfUserSchema)
    def get_users(self):
        return User.objects.all()

    @PUT(path='/<int:user_id>', response=UserSchema, request=UserSchema)
    def update_user(self, user_id, body):
        user = get_object_or_404(User, pk=user_id)
        user.name = body['name']
        user.email = body['email']
        user.save()
        return user

app = resource.Application()
app.add('/user', UserResouce)
```

In this example I've shown Django (like) example. The schema is based on pyrs.schema. Even if I tend to use that framework, you would be able to use any other.

CHAPTER 3

Features

- Using simple classes or even functions (no inheritance)
- Wrapped error handling, errors can be serialised
- Extensible API
- Works with python 2.7, 3.3, 3.4 (tested against these versions)
- Hooks for extending the dispatching process

CHAPTER 4

Installation

```
$ pip install pyrs-resource
```

CHAPTER 5

Dependencies

See *requirements.txt* for details, but mainly depends on Werkzeug. I'm using that project routing capabilities. Also depends on *pyrs.schema* as I mentioned in nutshell section.

CHAPTER 6

Important caveats

This code right now really in beta state. I plan to release soon as possible a completely working code, but right now it's just shaping.

CHAPTER 7

The ecosystem

This work is part of pyrs framework. The complete framework follow the same intention to implement flexible solution.

CHAPTER 8

Contribution

I really welcome any comments! I would be happy if you fork my code or create pull requests. I've already really strong opinions what I want to achieve and how, though any help would be welcomed.

Feel free drop a message to me!

CHAPTER 9

Contents:

9.1 Application

class `pyrs.resource.base.`**`App`**(*parent=None*, ***config*)

Bases: *`pyrs.resource.base.Dispatcher`*

`wsgi`(*request*)

class `pyrs.resource.base.`**`Directory`**(*parent=None*, ***config*)

Bases: `object`

`_add_class`(*path*, *resource*, *prefix=''*)

`_add_function`(*path*, *resource*, *prefix=''*)

`_make_rule`(*path*, *methods*, *endpoint*)

`add`(*path*, *resource*, *prefix=''*)

`host` **= 'localhost'**

`match`(*path_info*, *method*)

`parent`

`resources` **= None**

Tuple should be presented as ('/path', Resource, [namespace])

`root`

`setup`()

class `pyrs.resource.base.`**`Dispatcher`**(*parent=None*, ***config*)

Bases: *`pyrs.resource.base.Directory`*

class **`Response`**(*response=None*, *status=None*, *headers=None*, *mimetype=None*, *content_type=None*, *direct_passthrough=False*)

Bases: `werkzeug.wrappers.Response`, `pyrs.resource.gateway.CompatibilityMixin`, `pyrs.resource.gateway.ProducerMixin`, `pyrs.resource.gateway.ExceptionMixin`

`default_mimetype` **= 'application/json'**

`Dispatcher.`**`dispatch`**(*request*, *path=None*, *scope=None*)

classmethod `Dispatcher.`**`forward`**(*scope*, *path*, *resource*)

class `pyrs.resource.base.`**`Scope`**(*request*, *application*)

Bases: `object`

class `Response` (*response=None*, *status=None*, *headers=None*, *mimetype=None*, *content_type=None*, *direct_passthrough=False*)

Bases: `werkzeug.wrappers.Response`, `pyrs.resource.gateway.CompatibilityMixin`, `pyrs.resource.gateway.ProducerMixin`, `pyrs.resource.gateway.ExceptionMixin`

`default_mimetype` = **'application/json'**

`Scope.`**`forward`** (*resource*, *path='/'*)

`Scope.`**`response`**

9.2 Resource

`pyrs.resource.resource.`**`DELETE`** (*_func=None*, ***kwargs*)

Decorator function Ensure the given function will be available for DELETE method

`pyrs.resource.resource.`**`FORWARD`** (*_func=None*, *path='/'*, *forward=None*)

Ensure forwarding the request to an other resource

`pyrs.resource.resource.`**`GET`** (*_func=None*, ***kwargs*)

Decorator function Ensure the given function will be available for GET method

`pyrs.resource.resource.`**`PATCH`** (*_func=None*, ***kwargs*)

Decorator function Ensure the given function will be available for PATCH method

`pyrs.resource.resource.`**`POST`** (*_func=None*, ***kwargs*)

Decorator function Ensure the given function will be available for POST method

`pyrs.resource.resource.`**`PUT`** (*_func=None*, ***kwargs*)

Decorator function Ensure the given function will be available for PUT method

`pyrs.resource.resource.`**`RPC`** (*_func=None*, ***kwargs*)

Decorator function Ensure the given function will be available for POST method This action tend to use as Remote procedure call

`pyrs.resource.resource.`**`endpoint`** (*_func=None*, *path='/'*, ***kwargs*)

Deadly simple decorator, add options to the given function. Can be user with or without any keyword arguments. The default options would contain the path and the name of the function. Based on configuration: *`conf.decorate`*

9.3 Response handling

9.4 Response handling

9.5 Error handling

exception `pyrs.resource.errors.`**`BadRequestError`** (*message=None*, *errors=None*, ***details*)

Bases: *`pyrs.resource.errors.ClientError`*

Request cannot be processed because of an error.

`error` = **'BadRequest'**

`schema`

alias of *`BadRequestErrorSchema`*

class `pyrs.resource.errors.`**`BadRequestErrorSchema`**(*extend=None*, ***attrs*)

Bases: *`pyrs.resource.errors.ErrorSchema`*

`_attrs` **= OrderedDict([('additional', False)])**

`_definitions` **= None**

`_fields` **= OrderedDict([('error', <pyrs.schema.types.String object at 0x7f3141544ac8>), ('error_description', <pyrs.sch**

`to_raw`(*value*, *context=None*)

exception `pyrs.resource.errors.`**`ClientError`**(**args*, ***details*)

Bases: *`pyrs.resource.errors.Error`*

Generic Client Error. Normally the client errors have 4xx status codes.

`status` **= 400**

class `pyrs.resource.errors.`**`DetailsSchema`**(*extend=None*, ***attrs*)

Bases: `pyrs.schema.types.Object`

Details part of the error schema. Additional properties possible.

`_attrs` **= OrderedDict([('additional', True)])**

`_definitions` **= None**

`_fields` **= OrderedDict([('traceback', <pyrs.schema.types.Array object at 0x7f3141544908>), ('args', <pyrs.schema.type**

exception `pyrs.resource.errors.`**`Error`**(**args*, ***details*)

Bases: `Exception`

This is the base exception of this framework. The response based on this exception will be a JSON data

`description` **= None**

Description of error. Should give details about the error In the message it will appearing as error_description

`details` **= None**

None used as empty dict. Gives extra information about this error which could be parsed by the consumer of API.

`error` **= None**

Error code should be a string. If it's not specified the class fully qualified name will be used

`get_details`(*debug=False*)

Gives back detailed information about the error and the context. As this is part of the message should conform with the *DetailsSchema*.

`get_headers`()

This method gives back the header property by default or an empty dict, but you can override, then provide special headers based on the context

`get_status`()

This method gives back the status property by default which will be threated as HTTP status code. You can override, then provide your own status code based on the context.

`headers` **= None**

HTTP Response headers, (default None processed as empty)

`schema`

You can specify your schema class for validating your message By default the application default error schema the *ErrorSchema* will be used

alias of *`ErrorSchema`*

status = 500
HTTP status code (default=500)

uri = None
Reference for this error. You can pointing out a documentation which gives more information about how could this error happen and how could be possible to avoid

classmethod wrap (*original*)
Wraps the exception gives back an *Error* instance. The created *Error* instance *error* property will be updated by the fully qualified name of the *original* exception. You could use it for *Error* instances as well, though is not recommended.

class pyrs.resource.errors.**ErrorSchema** (*extend=None*, ***attrs*)
Bases: pyrs.schema.types.Object

Describe how the error response should look like. Goal of this schema is a minimalistic but usable error response.

_attrs = OrderedDict([('additional', False)])

_definitions = None

_fields = OrderedDict([('error', <pyrs.schema.types.String object at 0x7f3141544ac8>), ('error_description', <pyrs.sch

get_details (*ex*)
Gives back detailed information about the error and the context. By default its an empty dictionary. The *debug* depends on the debug parameter should give back traceback information and the positional arguments of the exception. As this is part of the message should conform with the *ErrorSchema*.

to_raw (*value*, *context=None*)

exception pyrs.resource.errors.**InternalServerError** (**args*, ***details*)
Bases: *pyrs.resource.errors.Error*

error = 'InternalServerError'

status = 500

exception pyrs.resource.errors.**NotFound** (**args*, ***details*)
Bases: *pyrs.resource.errors.ClientError*

error = 'NotFound'

status = 404

9.6 Hooks

Hooks in general the way to override amend the exist functionality of app. Even you could extend the app, sometimes much easier if you attach a hook like authentication hook and the will process the request, make request.auth available. But also you can create your own hook handling special header values or give special error handling strategy.

The *Hook* class provide the skeleton of any further hooks.

class pyrs.resource.hooks.**Hook**
Bases: object

Hooks help to extend the functionality of application. The 3 hooks executed in different time of execution. This class should be the base class of any further hook.

exception (*request*, *exception*)
If the function raise any exception it can be handled with this hook. return will be used as response if it gives back any (should be *Response* instance or *None*)

request (*request*)
Executed when the request is created. It can amend the request. If has any return value it will be used as return value of the call, the the function will be not called. Can raise any exception and that will be treated as the function exception, in that case the function will be not called.

response (*response*)
Executed after successful call of the function. Response object created and passed to the hook. Can modify the response or give back a new response. Have to return the response object.

9.7 Configuration

This module contains the default configurations. The *pyrs.resource.base.App* config will be based on these values.

pyrs.resource.conf.**debug = False**
You can get more information in response like traceback and args of exception

pyrs.resource.conf.**decorate = '_endpoint'**
This option will be used for decorators. Usage *getattr(func, conf.decorate)*

pyrs.resource.conf.**host = 'localhost'**
Default host for the application

pyrs.resource.conf.**inject_app = False**
Enable/disable injecting the *base.App* as keyword argument

pyrs.resource.conf.**inject_app_name = 'app'**
Name used for app injection

pyrs.resource.conf.**inject_auth = False**
Enable/disable injecting the `request.auth` as keyword argument

pyrs.resource.conf.**inject_auth_name = 'auth'**
With this name the auth will be injected

pyrs.resource.conf.**inject_body = True**
Enable/disable injecting the request body

pyrs.resource.conf.**inject_cookies = False**
Enable/disable injecting the cookies

pyrs.resource.conf.**inject_cookies_name = 'cookies'**
With this name the cookies will be injected

pyrs.resource.conf.**inject_path = True**
Enable/disable injecting the path arguments If a name provided the path arguments will be injected as specified

pyrs.resource.conf.**inject_query = True**
Enable/disable injecting the query arguments If a name provided the query arguments will be injected as specified

CHAPTER 10

License

10.1 Indices and tables

- genindex
- modindex
- search

Python Module Index

Index