Flask-RESTPlus Documentation

Release 0.1

Axel Haustant

August 19, 2014

Contents

Flask-RestPlus provide syntaxic suger, helpers and automatically generated Swagger documentation on top of Flask-Restful.

CHAPTER 1

Compatibility

flask-restplus requires Python 2.7+.

CHAPTER 2

Installation

You can install flask-restplus with pip:

```
$ pip install flask-restplus
```

or with easy_install:

```
$ easy_install flask-restplus
```

CHAPTER 3

Documentation

3.1 Quick start

As every other extension, you can initialize it with an application object:

```
from flask import Flask
from flask.ext.restplus import Api

app = Flask(__name__)
api = Api(app)
```

of lazily with the factory pattern:

```
from flask import Flask
from flask.ext.restplus import Api

api = Api()

app = Flask(__name__)
api.init_app(api)
```

With Flask-Restplus, you only import the api instance to route and document your endpoints.

```
from flask import Flask
from flask.ext.restplus import Api, Resource, fields

app = Flask(__name__)
api = Api(app)

@api.route('/somewhere')
class Somewhere(Resource):
    def get(self):
        return {}

    def post(self)
        api.abort(403)
```

3.2 Syntaxic sugar

One of the purpose of Flask-Restplus is to provide some syntaxic sugar of Flask-Restful.

3.2.1 Route with decorator

The Api class has a `route()` decorator used to route API's endpoint.

When with Flask-Restful you were writting :

```
class MyResource(Resource):
    def get(self, id):
        return {}

api.add_resource('/my-resource/<id>', MyResource.as_view('my-resource'))
```

With Flask-Restplus, you can write:

```
@api.route('/my-resource/<id>', endpoint='my-resource')
class MyResource(Resource):
    def get(self, id):
        return {}
```

You can optionnaly provide class-wide documentation:

```
@api.route('/my-resource/<id>', endpoint='my-resource', doc={params:{'id': 'An ID'}})
class MyResource(Resource):
    def get(self, id):
        return {}
```

The namespace object provide the same feature:

```
ns = api.namespcae('ns', 'Some namespace')

@ns.route('/my-resource/<id>', endpoint='my-resource')
class MyResource(Resource):
    def get(self, id):
        return {}
```

3.2.2 `abort` shortcut

You can use the `Api.abort()` method to abort a request. This shortcut always serialize the response in the right format.

```
@api.route('/failure')
class MyResource(Resource):
    def get(self):
        api.abort(403)

    def post(self):
        api.abort(500, 'Some custom message')
```

3.2.3 `parser` shortcut

You can use the `Api.parser()` shortcut to obtain a RequestParser instance.

```
parser = api.parser()
parser.add_argument('param', type=str, help='Some parameter')
```

3.2.4 `marshal` shortcut

You can use the `Api.marshal()` shortcut to serialize your objects.

```
return api.marshal(todos, fields), 201
```

3.3 Documenting your API with Swagger

A Swagger API documentation is automatically generated and available on your API root but you need to provide some details with the `Api.doc()` decorator.

3.3.1 Documenting with the `Api.doc()` decorator

This decorator allows you specify some details about your API. They will be used in the Swagger API declarations.

You can document a class or a method.

```
@api.route('/my-resource/<id>', endpoint='my-resource')
@api.doc(params={'id': 'An ID'})
class MyResource(Resource):
    def get(self, id):
        return {}

    @api.doc(responses={403: 'Not Authorized'})
    def post(self, id):
        api.abort(403)
```

3.3.2 Documenting with the `Api.marshal_with()` decorator

This decorator works like the Flask-Restful `marshal_with` decorator with the difference that it documents the methods. The optionnal parameter `as_list` allows you to specify wether or not the objects are returned as a list.

```
resource_fields = {
    'name': fields.String,
}

@api.route('/my-resource/<id>', endpoint='my-resource')
class MyResource(Resource):
    @api.marshal_with(resource_fields, as_list=True)
    def get(self):
        return get_objects()

    @api.marshal_with(resource_fields)
    def post(self):
        return create_object()
```

The `Api.marshal_list_with()` decorator is strictly equivalent to `Api.marshal_with(fields, as_list=True)`.

```
resource_fields = {
    'name': fields.String,
}

@api.route('/my-resource/<id>', endpoint='my-resource')
```

```
class MyResource(Resource):
    @api.marshal_list_with(resource_fields)
    def get(self):
        return get_objects()

    @api.marshal_with(resource_fields)
    def post(self):
        return create_object()
```

3.3.3 Documenting with the `Api.route()` decorator

You can provide class-wide documentation by using the `Api.route()`'s' `doc` parameter. It accept the same attribute/syntax than the `Api.doc()` decorator.

By example, these two declaration are equivalents:

```
@api.route('/my-resource/<id>', endpoint='my-resource')
@api.doc(params={'id': 'An ID'})
class MyResource(Resource):
    def get(self, id):
        return {}

@api.route('/my-resource/<id>', endpoint='my-resource', doc={params:{'id': 'An ID'}})
class MyResource(Resource):
    def get(self, id):
        return {}
```

3.3.4 Documenting with the `Api.model()` decorator

The `Api.model` decorator allows you to declare the models that your API can serialize.

You can use it either on a fields dictionnary or a `field.Raw` subclass:

```
my_fields = api.model('MyModel', {
    'name': fields.String
})

@api.model('MyField')
class MySpecialField(fields.Raw):
    pass
```

3.3.5 Cascading

Documentation handling is done in cascade. Method documentation override class-wide documentation. Inherited documentation override parent one.

By example, these two declaration are equivalents:

```
@api.route('/my-resource/<id>', endpoint='my-resource')
@api.doc(params={'id': 'An ID'})
class MyResource(Resource):
    def get(self, id):
        return {}
```

```
@api.route('/my-resource/<id>', endpoint='my-resource')
@api.doc(params={'id': 'Class-wide description'})
class MyResource(Resource):
    @api.doc(params={'id': 'An ID'})
    def get(self, id):
        return {}
```

You can also provide method specific documentation from a class decoration. The following example will produce the same documentation than the two previous examples:

```
@api.route('/my-resource/<id>', endpoint='my-resource')
@api.doc(params={'id': 'Class-wide description'})
@api.doc(get={'params': {'id': 'An ID'}})
class MyResource(Resource):
    def get(self, id):
        return {}
```

3.3.6 Overriding the API root view

TODO

3.4 Full example

Here a full example extracted from Flask-Restful and ported to Flask-RestPlus.

```
from flask import Flask
from flask.ext.restplus import Api, Resource, fields

app = Flask(__name__)
api = Api(app, version='1.0', title='Todo API',
    description='A simple TODO API extracted from the original flask-restful example'
)

ns = api.namespace('todos', description='TODO operations')

TODOS = {
    'todo1': {'task': 'build an API'},
    'todo2': {'task': '?????'},
    'todo3': {'task': 'profit!'},
}

todo_fields = api.model('Todo', {
    'task': fields.String
})


def abort_if_todo_doesnt_exist(todo_id):
    if todo_id not in TODOS:
        api.abort(404, "Todo {} doesn't exist".format(todo_id))

parser = api.parser()
parser.add_argument('task', type=str, required=True, help='The task details')


@ns.route('/<string:todo_id>')
```

```
@api.doc(responses={404: 'Todo not found'}, params={'todo_id': 'The Todo ID'})
class Todo(Resource):
    '''Show a single todo item and lets you delete them'''
    @api.doc(notes='todo_id should be in {0}'.format(', '.join(TODOS.keys())))
    @api.marshal_with(todo_fields)
    def get(self, todo_id):
        '''Fetch a given resource'''
        abort_if_todo_doesnt_exist(todo_id)
        return TODOS[todo_id]

    def delete(self, todo_id):
        '''Delete a given resource'''
        abort_if_todo_doesnt_exist(todo_id)
        del TODOS[todo_id]
        return '', 204

    @api.doc(parser=parser)
    @api.marshal_with(todo_fields)
    def put(self, todo_id):
        '''Update a given resource'''
        args = parser.parse_args()
        task = {'task': args['task']}
        TODOS[todo_id] = task
        return task, 201


@ns.route('/')
class TodoList(Resource):
    '''Shows a list of all todos, and lets you POST to add new tasks'''
    @api.marshal_with(todo_fields, as_list=True)
    def get(self):
        '''List all todos'''
        return TODOS

    @api.doc(parser=parser)
    @api.marshal_with(todo_fields)
    def post(self):
        '''Ceate a todo'''
        args = parser.parse_args()
        todo_id = 'todo%d' % (len(TODOS) + 1)
        TODOS[todo_id] = {'task': args['task']}
        return TODOS[todo_id], 201


if __name__ == '__main__':
    app.run(debug=True)
```

You can find full examples in the github repository `examples` folder.

3.5 API

3.5.1 flask.ext.restplus

3.5.2 flask.ext.restplus.fields

CHAPTER 4

Indices and tables

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