Flask restful swagger Documentation

Release 1.0.0

Sobolev Nikita

January 03, 2017

Contents

Contents:

CHAPTER 1

What is flask-restful-swagger?

flask-restful-swagger is a wrapper for flask-restful which enables swagger support.

In essense, you just need to wrap the Api instance and add a few python decorators to get full swagger support.

CHAPTER 2

How to:

Install:

```
pip install flask-restful-swagger
```

(This installs flask-restful as well)

And in your program, where you'd usually just use flask-restful, add just a little bit of sauce and get a swagger spec out.

```
from flask import Flask
from flask.ext.restful import  Api
from flask_restful_swagger import swagger

app = Flask(__name__)

###################################
# Wrap the Api with swagger.docs. It is a thin wrapper around the Api class that adds some swagger sm
api = swagger.docs(Api(app), apiVersion='0.1')
###################################


# You may decorate your operation with @swagger.operation
class Todo(Resource):
    "Describing elephants"
    @swagger.operation(
        notes='some really good notes',
        responseClass=ModelClass.__name__,
        nickname='upload',
        parameters=[
            {
              "name": "body",
              "description": "blueprint object that needs to be added. YAML.",
              "required": True,
              "allowMultiple": False,
              "dataType": ModelClass2.__name__,
              "paramType": "body"
            }
          ],
        responseMessages=[
            {
              "code": 201,
              "message": "Created. The URL of the created blueprint should be in the Location header"
            },
```

```
            {
              "code": 405,
              "message": "Invalid input"
            }
          ]
        )
    def get(self, todo_id):

# Operations not decorated with @swagger.operation do not get added to the swagger docs

class Todo(Resource):
    def options(self, todo_id):
        """
        I'm not visible in the swagger docs
        """
        pass


# Then you add_resource as you usually would

api.add_resource(TodoList, '/todos')
api.add_resource(Todo, '/todos/<string:todo_id>')

# You define models like this:
@swagger.model
class TodoItem:
  "A description ..."
  pass

# Swagger json:
    "models": {
        "TodoItemWithArgs": {
            "description": "A description...",
            "id": "TodoItem",
        },

# If you declare an __init__ method with meaningful arguments then those args could be used to deduce
@swagger.model
class TodoItemWithArgs:
  "A description ..."
  def __init__(self, arg1, arg2, arg3='123'):
    pass

# Swagger json:
    "models": {
        "TodoItemWithArgs": {
            "description": "A description...",
            "id": "TodoItem",
            "properties": {
                "arg1": {
                    "type": "string"
                },
                "arg2": {
                    "type": "string"
                },
                "arg3": {
                    "default": "123",
                    "type": "string"
```

```
                }
            },
            "required": [
                "arg1",
                "arg2"
            ]
        },


# Additionally, if the model class has a `resource_fields` class member then flask-restful-swagger is

@swagger.model
class TodoItemWithResourceFields:
  resource_fields = {
      'a_string': fields.String
  }

# Swagger json:
    "models": {
        "TodoItemWithResourceFields": {
            "id": "TodoItemWithResourceFields",
            "properties": {
                "a_string": {
                    "type": "string"
                },
            }
        }

# And in order to close the loop with flask-restify you'd also need to tell flask-restify to @marshal
# Example:

@marshal_with(TodoItemWithResourceFields.resource_fields)
def get()
  return ...
```

CHAPTER 3

Using @marshal_with

Let us recap usage of @marshal_with. flask-restful has a decorator `@marshal_with`. With the following setup it's possible to define the swagger model types with the same logic as `@marshal_with`.

You have to:

```
# Define your model with resource_fields
@swagger.model
class TodoItemWithResourceFields:
  resource_fields = {
      'a_string': fields.String,
      'a_second_string': fields.String(attribute='a_second_string_field_name')
  }

# And use @marshal_with(YourClass.resource_fields):
@marshal_with(TodoItemWithResourceFields.resource_fields)
def get()
  return ...
```

CHAPTER 4

Running and testing

Now run your flask app

```
python example.py
```

And visit:

```
curl http://localhost:5000/api/spec.json
```

CHAPTER 5

Passing more metadata to swagger

When creating the `swagger.docs` object you may pass additional arguments, such as the following:

```
api_spec_url - where to serve the swagger spec from. Default is /api/spec. This will make the json
available at /api/spec as well as /api/spec.json and will also present a nice interactive
HTML interface at /api/spec.html

apiVersion - passed directly to swagger as the apiVersion attribute. Default: 0.0

basePath - passed directly to swagger as the basePath attribute. Default: 'http://localhost:5000' (do

resourcePath - same as before. default: '/'

produces - same as before, passed directly to swagger. The default is ["application/json"]

swaggerVersion - passed directly to swagger. Default: 1.2

description - description of this API endpoint. Defaults to 'Auto generated API docs by flask-restful
```

CHAPTER 6

Accessing the result json spec and an Interactive HTML interface

Assuming you provided `swagger.docs` with a parameter `api_spec_url='/api/spec'` (or left out in which case the default is '/api/spec') you may access the resulting json at /api/spec.json. You may also access /api/spec.html where you'd find an interactive HTML page that lets you play with the API to some extent.

Here's how this HTML page would look like:

Fig. 6.1: An example /api/spec.html page

CHAPTER 7

Accessing individual endpoints (.help.json)

flask-restful-swagger adds some useful help pages (well, json documents) to each of your resources. This isn't part of the swagger spec, but could be useful anyhow. With each endpoint you register, there's also an automatically registered help endpoint which ends with a .help.json extension. So for example when registering the resource `api.add_resource(TodoList, '/todos')` you may access the actual api through the url `/todos` and you may also access the help page at `/todos.help.json`. This help page spits out the relevant json content only for this endpoint (as opposed to `/api/spec.json` which spits out the entire swagger document, which could be daunting)

Example:

```
### python:

> api.add_resource(TodoList, '/todos')

### Shell:

$ curl localhost:5000/todos.help.json
{
    "description": null,
    "operations": [
        {
            "method": "GET",
            "nickname": "nickname",
            "parameters": [],
            "summary": null
        },
        {
            "method": "POST",
            "nickname": "create",
            "notes": "Creates a new TODO item",
            "parameters": [
                {
                    "allowMultiple": false,
                    "dataType": "TodoItem",
                    "description": "A TODO item",
                    "name": "body",
                    "paramType": "body",
                    "required": true
                }
            ],
            "responseClass": "TodoItem",
            "responseMessages": [
                {
```

```
                    "code": 201,
                    "message": "Created. The URL of the created blueprint should be in the Location h
                },
                {
                    "code": 405,
                    "message": "Invalid input"
                }
            ],
            "summary": null
        }
    ],
    "path": "/todos"
}
```

When registering an endpoint with path parameters (e.g. `/todos/<string:id>`) then the .help url is may be found at the swagger path, e.g. `/todos/{id}.help.json` where {id} is just that - a literal string "{id}"

Example:

```
### Python:
> api.add_resource(Todo, '/todos/<string:todo_id>')

### Shell:
 # You might need to quote and escape to prevent the shell from messing around

curl 'localhost:5000/todos/\{todo_id\}.help.json'
{
    "description": "My TODO API",
    "operations": [
        {
            "method": "DELETE",
            "nickname": "nickname",
            "parameters": [
                {
                    "dataType": "string",
                    "name": "todo_id"
                }
            ],
            "summary": null
        },
        {
            "method": "GET",
            "nickname": "get",
            "notes": "get a todo item by ID",
            "parameters": [
                {
                    "allowMultiple": false,
                    "dataType": "string",
                    "description": "The ID of the TODO item",
                    "name": "todo_id_x",
                    "paramType": "path",
                    "required": true
                }
            ],
            "responseClass": "TodoItemWithResourceFields",
            "summary": "Get a todo task"
        },
        {
            "method": "PUT",
```

```
            "nickname": "nickname",
            "parameters": [
                {
                    "dataType": "string",
                    "name": "todo_id"
                }
            ],
            "summary": null
        }
    ],
    "path": "/todos/{todo_id}"
}
```

CHAPTER 8

Accessing individual endpoints as HTML (.help.html)

Similarly to the `.help.json` URLs we have `.help.html` pages which are static HTML pages to document your APIs. Here's a screenshot to illustrate:

This project is part of the 'Cloudify Cosmo project <https://github.com/CloudifySource/>'__

CHAPTER 9

Internal functions documentation

Core wrapper functions:

flask_restful_swagger.swagger.**docs**(*api*, ***kwargs*)

This function adds endpoints for the swagger. It also handles all the model loading by replacing original *add_resource* with the patched one.

:version changed 1.0.0 The old docs() function before version 1.0.0 had 'camelCase' kwargs, which was not-PEP8, and now it is recommended to use 'snake_case'. But for backward compatibility 'cameCase' is also accepted.

Parameters

- **api** – flask-resful's Api object
- **kwargs** – key-word arguments described in *_docs* function.

Returns flask-resful's Api object passed as *api*.

flask_restful_swagger.swagger.**extract_path_arguments**(*path*)

Extracts a swagger path arguments from the given flask path.

Examples

This /path/<parameter> extracts [{name: 'parameter'}]

And this /<string(length=2):lang_code>/<string:id>/<float:probability> extracts: [{name: 'lang_code', dataType: 'string'}, {name: 'id', dataType: 'string'} {name: 'probability', dataType: 'float'}]

flask_restful_swagger.swagger.**extract_swagger_path**(*path*)

Extracts a swagger type path from the given flask style path. This /path/<parameter> turns into this /path/{parameter} And this /<string(length=2):lang_code>/<string:id>/<float:probability> to this: /{lang_code}/{id}/{probability}

flask_restful_swagger.swagger.**operation**(***kwargs*)

This decorator marks a function as a swagger operation so that we can easily extract attributes from it. It saves the decorator's key-values at the function level so we can later extract them later when add_resource is invoked.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

f

D

E

F

O