aiohttp-apispec

Mar 12, 2019

Contents:

Build and document REST APIs with aiohttp and apispec

`aiohttp-apispec` key features:

- `docs`, `request_schema` and `response_schema` decorators to add swagger spec support out of the box;
- `validation_middleware` middleware to enable validating with marshmallow schemas from those decorators;
- **SwaggerUI** support.

`aiohttp-apispec` api is fully inspired by `flask-apispec` library

CHAPTER 1

Guide

1.1 Usage

1.1.1 Quickstart

```
from aiohttp_apispec import (docs,
                             request_schema,
                             response_schema,
                             setup_aiohttp_apispec)
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')
    bool_field = fields.Bool()


class ResponseSchema(Schema):
    msg = fields.Str()
    data = fields.Dict()


@docs(tags=['mytag'],
      summary='Test method summary',
      description='Test method description')
@request_schema(RequestSchema(strict=True))
@response_schema(ResponseSchema(), 200)
async def index(request):
    return web.json_response({'msg': 'done',
                              'data': {}})
```

(continues on next page)

(continued from previous page)

```
# Class based views are also supported:
class TheView(web.View):
    @docs(
        tags=['mytag'],
        summary='View method summary',
        description='View method description',
    )
    @request_schema(RequestSchema(strict=True))
    def delete(self):
        return web.json_response({
            'msg': 'done',
            'data': {'name': self.request['data']['name']},
        })


app = web.Application()
app.router.add_post('/v1/test', index)
app.router.add_view('/v1/view', TheView)

# init docs with all parameters, usual for ApiSpec
setup_aiohttp_apispec(app=app, title="My Documentation", version="v1")


# find it on 'http://localhost:8080/api/docs/api-docs'
web.run_app(app)
```

1.1.2 Adding validation middleware

```
from aiohttp_apispec import validation_middleware

...

app.middlewares.append(validation_middleware)
```

Now you can access all validated data in route from `request['data']` like so:

```
@docs(tags=['mytag'],
      summary='Test method summary',
      description='Test method description')
@request_schema(RequestSchema(strict=True))
@response_schema(ResponseSchema(), 200)
async def index(request):
    uid = request['data']['id']
    name = request['data']['name']
    return web.json_response(
        {'msg': 'done',
         'data': {'info': f'name - {name}, id - {uid}'}}
     )
```

You can change `Request`'s `'data'` param to another with `request_data_name` argument of `setup_aiohttp_apispec` function:

```
setup_aiohttp_apispec(app=app,
                      request_data_name='validated_data',
```

(continues on next page)

(continued from previous page)

```
                        title='My Documentation',
                        version='v1',
                        url='/api/docs/api-docs')

...

@request_schema(RequestSchema(strict=True))
async def index(request):
    uid = request['validated_data']['id']
    ...
```

If you want to catch validation errors you should write your own middleware and catch `web.HTTPClientError`, `json.JSONDecodeError` and so on. Like this:

```
@web.middleware
async def my_middleware(request, handler):
    try:
        return await handler(request)
    except web.HTTPClientError:
        return web.json_response(status=400)

app.middlewares.extend([
    my_middleware,  # Catch exception by your own, format it and respond to client
    validation_middleware,
])
```

1.1.3 Build swagger web client

`aiohttp-apispec` adds `swagger_dict` parameter to aiohttp web application after initialization (with `setup_aiohttp_apispec` function). So you can use it easily with `aiohttp_swagger` library:

```
from aiohttp_apispec import setup_aiohttp_apispec
from aiohttp_swagger import setup_swagger


def create_app(app):
    setup_aiohttp_apispec(app)

    async def swagger(app):
        setup_swagger(
            app=app, swagger_url='/api/doc', swagger_info=app['swagger_dict']
        )
    app.on_startup.append(swagger)
    # now we can access swagger client on '/api/doc' url
    ...
    return app
```

Now we can access swagger client on `/api/doc` url

1.2 Installation

```
$ pip install -U aiohttp-apispec
```

1.3 API Reference

aiohttp_apispec.**setup_aiohttp_apispec**(*app: aiohttp.web_app.Application*, ***, *title: str = 'API documentation'*, *version: str = '0.0.1'*, *url: str = '/api/docs/swagger.json'*, *request_data_name: str = 'data'*, *swagger_path: str = None*, *static_path: str = '/static/swagger'*, ***kwargs*) → None

aiohttp-apispec extension.

Usage:

```
from aiohttp_apispec import docs, request_schema, setup_aiohttp_apispec
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')
    bool_field = fields.Bool()


@docs(tags=['mytag'],
      summary='Test method summary',
      description='Test method description')
@request_schema(RequestSchema)
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})


app = web.Application()
app.router.add_post('/v1/test', index)

# init docs with all parameters, usual for ApiSpec
setup_aiohttp_apispec(app=app,
                      title='My Documentation',
                      version='v1',
                      url='/api/docs/api-docs')

# now we can find it on 'http://localhost:8080/api/docs/api-docs'
web.run_app(app)
```

Parameters

- **app** (*Application*) – aiohttp web app
- **title** (*str*) – API title
- **version** (*str*) – API version
- **url** (*str*) – url for swagger spec in JSON format
- **request_data_name** (*str*) – name of the key in Request object where validated data will be placed by validation_middleware (`'data'` by default)
- **swagger_path** (*str*) – experimental SwaggerUI support (starting from v1.1.0). By default it is None (disabled)
- **static_path** (*str*) – path for static files used by SwaggerUI (if it is enabled with `swagger_path`)

- **kwargs** – any apispec.APISpec kwargs

aiohttp_apispec.**docs**(***kwargs*)

Annotate the decorated view function with the specified Swagger attributes.

Usage:

```
from aiohttp import web

@docs(tags=['my_tag'],
      summary='Test method summary',
      description='Test method description',
      parameters=[{
              'in': 'header',
              'name': 'X-Request-ID',
              'schema': {'type': 'string', 'format': 'uuid'},
              'required': 'true'
          }]
      )
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})
```

aiohttp_apispec.**request_schema**(*schema*, *locations=None*, ***kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request[‘data’] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse

aiohttp_apispec.**response_schema**(*schema*, *code=200*, *required=False*, *description=None*)

Add response info into the swagger spec

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields
```

(continues on next page)

(continued from previous page)

```
class ResponseSchema(Schema):
    msg = fields.Str()
    data = fields.Dict()

@response_schema(ResponseSchema(), 200)
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})
```

Parameters

- **description** (*str*) – response description
- **required** (*bool*) –
- **schema** – Schema class or instance
- **code** (*int*) – HTTP response code

aiohttp_apispec.**use_kwargs**(*schema*, *locations=None*, ***kwargs*)

Add request info into the swagger spec and prepare injection keyword arguments from the specified webargs arguments into the decorated view function in request['data'] for validation_middleware validation middleware.

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class RequestSchema(Schema):
    id = fields.Int()
    name = fields.Str(description='name')

@request_schema(RequestSchema(strict=True))
async def index(request):
    # aiohttp_apispec_middleware should be used for it
    data = request['data']
    return web.json_response({'name': data['name'],
                              'id': data['id']})
```

Parameters

- **schema** – Schema class or instance
- **locations** – Default request locations to parse

aiohttp_apispec.**marshal_with**(*schema*, *code=200*, *required=False*, *description=None*)

Add response info into the swagger spec

Usage:

```
from aiohttp import web
from marshmallow import Schema, fields


class ResponseSchema(Schema):
    msg = fields.Str()
    data = fields.Dict()
```

(continues on next page)

(continued from previous page)

```
@response_schema(ResponseSchema(), 200)
async def index(request):
    return web.json_response({'msg': 'done', 'data': {}})
```

Parameters

- **description** (*str*) – response description
- **required** (*bool*) –
- **schema** – Schema class or instance
- **code** (*int*) – HTTP response code

aiohttp_apispec.**validation_middleware**(*request: aiohttp.web_request.Request*, *handler*) → aiohttp.web_response.Response

Validation middleware for aiohttp web app

Usage:

```
app.middlewares.append(validation_middleware)
```

Python Module Index

a

Index