restit Documentation

Release 0.3.2

Erik Tuerke

Apr 24, 2022

Contents

Welcome to the **RestIt** documentation.

Python HTTP REST library including OOP-readiness and Open-API generation

CHAPTER 1

Installation

CHAPTER 2

Getting Started

2.1 A Minimal Application

To get started with RestIt we can use the following code snippet:

```
from restit import Request, Resource, Response, RestItApp
from restit.decorator import path


@path("/")
class IndexResource(Resource):
    def get(self, request: Request) -> Response:
        return Response("Hello from index.")


app = RestItApp(resources=[IndexResource()])


if __name__ == "__main__":
    # start a development server on http://127.0.0.1:5000
    app.start_development_server()
```

One of the key aspects of *REST* and the *RestIt* library are *Resources*. Since a resource is identified with an *URI*, in our Python code we assign it using the `path()` decorator.

2.2 Swagger/OpenApi Documentation

To get your HTTP app serving an *OpenApi* documentation, you have to create an instance of `OpenApiDocumentation` and pass it to your `RestItApp` constructor.

```
open_api_documentation = OpenApiDocumentation(
    info=InfoObject(title="My HTTP API", version="1.0.0"), path="/api"
```

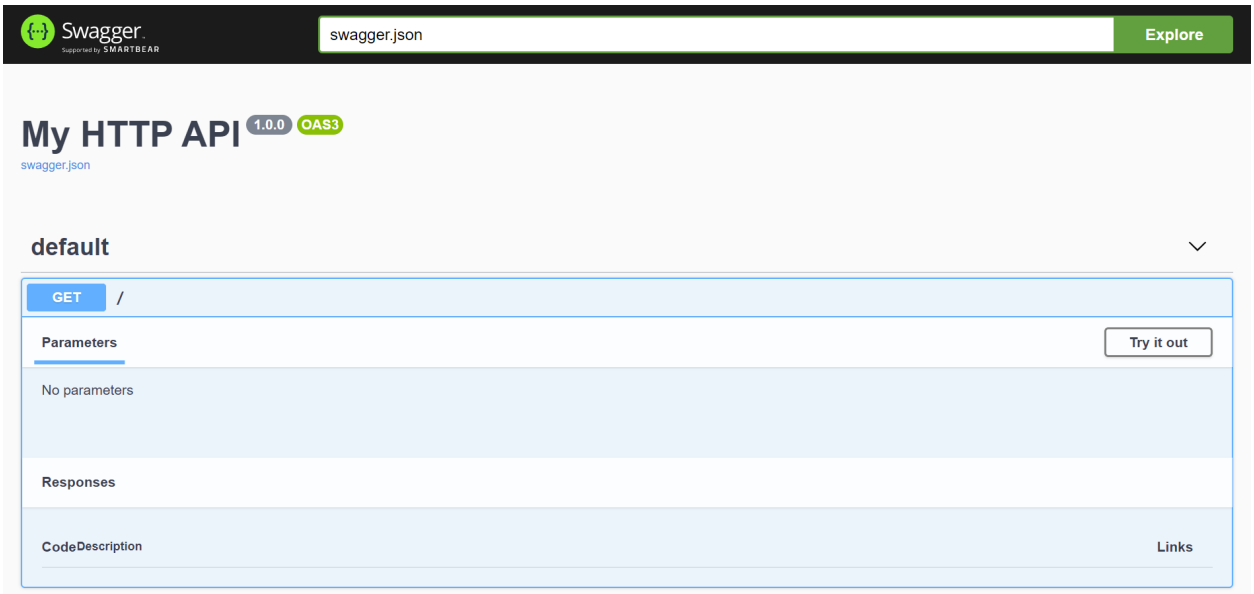
(continues on next page)

(continued from previous page)

```
)

app = RestItApp(
    resources=[IndexResource()], open_api_documentation=open_api_documentation
)
```

Once you start your development server and navigate to `http://127.0.0.1:5000/api/` you will see a minimal *OpenApi* documentation.



Note: Since we did not yet provide any information about our API we do not see too much in the *OpenApi* documentation yet.

2.3 Describe your REST API

The following decorators

2.3.1 Request Method Description

A description for the request method is always a good starting point and so we are adding a simple doc string to our `get` method:

```
@path("/")
class IndexResource(Resource):
    def get(self, request: Request) -> Response:
        """This is a super get method.

        It takes a request and responds with a text.
        """
        return Response("Hello from index.")
```

The doc string then will be used to generate the summary and description fields.

The first line will always be treated as the *summary* and the following lines as the *description*.

Note: The doc string of the resource class will also be recognized and added to the PathItemObject, but for some reason it might not be appear in the *OpenApi* documentation.

2.3.2 Describing the Path Parameters

Imagine you want to add a resource with a parameter in the *URL* - a so called *path parameter*. So for instance, we want to serve the *URL* `/users/:id`:

```
from marshmallow import fields

...


@path("/users/:id")
@path_parameter("id", "The user id", fields.Integer())
class UserResource(Resource):
    def get(self, request: Request) -> Response:
        """Get user information"""
        return Response({"id": request.path_parameters["id"]})
```

Though our *HTTP* service would also consider the path parameter `id` here without the *`path_parameter()`* decorator, we add it because we want to:

1. Hand more information about that parameter to the *OpenApi* documentation
2. Use marshmallow for validation and deserialization here

So in our *OpenApi* documentation we will see the description and the type of our path parameter, but we will also get the path parameter `id` as an *int* in our request method. And we will also get a *400 BadRequest* response status, if the incoming path parameter can not be deserialized (in our example, because someone is passing a `id` of type string).

Note: As an alternative syntax you can also register path parameters the following way:

```
@path("/users/:id", path_parameters=[PathParameter("id", "The user id", fields.
↪Integer())])
class UserResource(Resource):
    ...
```

2.3.3 Describing the Query Parameters

So now imagine we want to add a query parameter that controls whether to send the address information or not. Lets call it `address_info`:

```
@path("/users/:id")
@path_parameter("id", "The user id", fields.Integer())
class UserResource(Resource):

    @query_parameter("address_info", "Send address information", fields.
↪Boolean(default=False))
```

(continues on next page)

(continued from previous page)

```
    def get(self, request: Request) -> Response:
        """Get user information"""

        if request.query_parameters["address_info"]:
            # collect address information here

        return Response({"name": ...})
```

An example *URL* can be:

- `/users/1?address_info=true`
- `/users/1?address_info=false`
- `/users/1` (which here defaults to *false*)

2.3.4 Describing the Request Body

If you expect a response body with an incoming request, you can specify that with the *request_body()* decorator.

First we need to define our schema:

```
from marshmallow import Schema, fields


class MyRequestSchema(Schema):
    """This is my example request schema"""
    string_fields = fields.String()
    string_fields.__doc__ = "A field holding a string value"
    integer_field = fields.Integer()
    integer_field.__doc__ = "A field holding an integer value"
```

Now we can use that schema to describe our expected request body:

```
@path("/orders)
class MyResource(Resource):

    @request_body({"application/json": MyRequestSchema()}, "My request body␣
↪description")
    def post(self, request: Request) -> Response:
        request_body = request.deserialized_body

        ...
```

As you can see, you can access the request body with the `deserialized_body` property.

2.3.5 Response Details

2.4 Exception Mapping

2.5 Hyperlink Generation

CHAPTER 3

RestIt API Reference

This part of the documentation covers all the interfaces of RestIt.

3.1 Restit Application

class restit.**RestItApp**(*resources: List[restit.resource.Resource] = None*, *namespaces: List[restit.namespace.Namespace] = None*, *debug: bool = False*, *raise_exceptions: bool = False*, *open_api_documentation: restit.open_api.open_api_documentation.OpenApiDocumentation = None*)

This class represents your *REST* application and is used to glue everything together.

Since it is a WSGI-Application, its instance can be passed to servers like Gunicorn.

Parameters

- **resources** (*List[Resource]*) – A list of *Resource* instances
- **namespaces** (*List[Namespace]*) – A list of `Namespace` instances
- **debug** (*bool*) – If set to *True*, you will get a detailed *HTML* stacktrace if an error is raised inside your application
- **raise_exceptions** (*bool*) – If set to *True*, exceptions will not cause error responses but will raise an error
- **open_api_documentation** (`OpenApiDocumentation`) – An instance of `OpenApiDocumentation`. If not set, no OpenApi documentation will be generated.

register_resources(*resources: List[restit.resource.Resource]*)

Register an instance of *Resource* to your application.

A list of resource instances can also be set in the constructor.

set_open_api_documentation(*open_api_documentation: restit.open_api.open_api_documentation.OpenApiDocumenta*

Set an instance of `OpenApiDocumentation`.

If not set, no OpenApi documenation will be generated.

Can also be set in the constructor.

start_development_server(*host: str = None*, *port: int = 5000*, *blocking: bool = True*) → int
This function starts a development server

> **Warning:** Do not use the development server in production!

Parameters

- **host** (*str*) – The host name, defaults to 127.0.0.1
- **port** (*int*) – The port number. If set to 0, the OS will assign a free port number for you. The port number will then be returned by that function, defaults to 5000
- **blocking** (*bool*) – If set to *True*, the function will block. Otherwise, the server will run in a thread and can be stopped by calling *stop_development_server()*.

Returns The port the development server is running on

Return type int

start_development_server_in_context(*host: str = None*, *port: int = 5000*) → int
Starts a development server in a context.

Example:

```
import requests

from restit import RestitApp, Request, Response, Resource, request_mapping


@request_mapping("/path")
class MyResource(Resource):
    def get(self, request: Request) -> Response:
        return Response("Hello")


my_restit_app = RestitApp(resources=[MyResource()])

with my_restit_app.start_development_server_in_context(port=0) as port:
    response = requests.get(f"http://127.0.0.1:{port}/path")
    assert response.status_code == 200
    assert response.text == "Hello"

# here the development server has stopped
```

Parameters

- **host** (*str*) – The host name, defaults to 127.0.0.1
- **port** – The port number. If set to 0, the OS will assign a free port number for you. The port number will then be returned by that function, defaults to 5000

Returns The port the development server is running on

Return type int

stop_development_server() → None
Stops the development server if started in non blocking mode.

3.2 Resource Related

class restit.**Resource**

Base class you have to inherit from when implementing your resource.

It provides the interface for all HTTP request methods and implicit implementation for the OPTIONS method.

If an *HTTP* method is not implemented but requested by the client, a `MethodNotAllowed` error will be raised leading to a 405 response status code on the client.

Note: You have to map your resource with an *URI* using the `request_mapping()` decorator.

---

restit.decorator.**path**(*path: str*, *path_parameters: List[restit._path_parameter.PathParameter] = None*)

Maps a resource *URI* path to your resource.

The path can contain multiple path parameters.

Example:

```
@path("/users/:id")
class UserResource(Resource):
    ...
```

Or:

```
@path("/orders/:year/:month/:id")
class OrdersResource(Resource):
    ...
```

Note: A specific request mapping path will always win. So given the following two paths:

- `/orders/api`
- `/orders/:id`

The incoming path `/orders/api` will always map to the resource with the `/orders/api` path.

Setting OpenApi **properties**

There are two ways of setting the path parameter properties for the *OpenApi* documentation.

1. Using the *`path_parameter()`* decorator
2. Passing a list of *`PathParameter`* instances to the `path_parameter` parameter of the *`path()`* decorator

Example for 2.:

```
from marshmallow import fields

...

@path(
    "/users/:id",
    [
        PathParameter("id", "The user id", fields.Integer())
    ]
```

(continues on next page)

(continued from previous page)

```
)
class UserResource(Resource):
    ...
```

As you can see, we are using the marshmallow library here.

Note: If you do not describe the path parameters with either *path_parameter()* or inside the *path()* decorator, they won´t show up in the *OpenApi* documentation and the type is considered to be string.

Parameters

- **path** (*str*) – The *URI* path to the resource
- **path_parameters** (*List[PathParameter]*) – Optional list of path parameter properties used to generate the OpenApi documentation. Can also be set using the `path_parameter()` decorator.

restit.decorator.**path_parameter**(*name: str, description: str, field_type: marshmallow.fields.Field = <fields.String(dump_default=<marshmallow.missing>, attribute=None, validate=None, required=False, load_only=False, dump_only=False, load_default=<marshmallow.missing>, allow_none=False, error_messages={'required': 'Missing data for required field.', 'null': 'Field may not be null.', 'validator_failed': 'Invalid value.', 'invalid': 'Not a valid string.', 'invalid_utf8': 'Not a valid utf-8 string.'})>*)

Decorator to describe the path parameters.

Besides describing the path parameters in the *path()* decorator function, you can add this decorator to your resource class.

Example:

```
@path("/orders/:year/:month/:id")
@path_parameter("year", "The year of the order", fields.Integer())
@path_parameter("month", "The month of the order", fields.Integer())
@path_parameter("id", "The order id", fields.Integer())
class OrdersResource(Resource):
    ...
```

Note: If you do not describe the path parameters with either *path_parameter()* or inside the *path()* decorator, they won´t show up in the *OpenApi* documentation and the type is considered to be string.

Parameters

- **name** (*str*) – The path parameter name
- **description** (*str*) – The description of the path parameter
- **field_type** (*marshmallow.fields.Field*) – The type of the path parameter

class restit._path_parameter.**PathParameter**(*name: str, description: str, field_type: Union[marshmallow.fields.Field, type]*)

Class that holds the path parameter properties and is used in the `path()` decorator.

Parameters

- **name** (*str*) – The path parameter name
- **description** (*str*) – The description of the path parameter
- **field_type** (*marshmallow.fields.Field*) – The type of the path parameter

restit.decorator.**query_parameter**(*name: str, description: str, field_type: marshmallow.fields.Field = <fields.String(dump_default=<marshmallow.missing>, attribute=None, validate=None, required=False, load_only=False, dump_only=False, load_default=<marshmallow.missing>, allow_none=False, error_messages={'required': 'Missing data for required field.', 'null': 'Field may not be null.', 'validator_failed': 'Invalid value.', 'invalid': 'Not a valid string.', 'invalid_utf8': 'Not a valid utf-8 string.'})>*)

restit.decorator.**request_body**(*content_types: Dict[str, Union[marshmallow.schema.Schema, marshmallow.fields.Field]], description: str, required: bool = True, validation_error_class=<class 'restit.exception.client_errors_4xx.UnprocessableEntity'>*)

Describe an expected request body for a method

Parameters

- **content_types** (*Dict[str, Union[marshmallow.Schema, marshmallow.fields.Field]*) – A dictionary that maps a *Content-Type* with a *marshmallow* schema or a field.
- **description** (*str*) – The description of the request body
- **required** (*bool*) – Is the request body required?
- **validation_error_class** (*An instance of type Exception*) – An error class either that should be raised if either the schema validation fails or the request body is missing.

3.3 OpenApi Documentation

class restit.open_api.**OpenApiDocumentation**(*info: restit.open_api.info_object.InfoObject, path: str = '/api'*)

Class that that is responsible for creating the OpenApi documentation.

If you want to create a *OpenApi* documentation, you have to instantiate this class and pass it your `RestItApp`.

Example:

```
from restit import RestItApp
from restit.open_api import OpenApiDocumentation, InfoObject, ContactObject,␣
↪LicenseObject

open_api_documentation = OpenApiDocumentation(
    info=InfoObject(
```

(continues on next page)

(continued from previous page)

```
        title="First OpenApi Test",
        description="Super description",
        version="1.2.3",
        contact=ContactObject("API Support", "http://www.example.com/support",
↪"support@example.com"),
        license=LicenseObject("Apache 2.0", "https://www.apache.org/licenses/
↪LICENSE-2.0.html"),
        terms_of_service="http://example.com/terms/"
    ),
    path="/some/custom/api/path"
)

restit_app = RestItApp(resource=[...], open_api_documentation=open_api_
↪documentation)

...
```

Once your app is running, you can access `http://<host>:<port>/some/custom/api/path/` to see your API documentation.

Parameters

- **info** (InfoObject) – Metadata about the API
- **path** (*str*) – The path where the API is served

generate_spec

Generate the OpenApi specification as a dictionary

Note: Only use this function if you want to generate the API specification outside your app.

Returns The generated specification

Return type dict

register_resource(*resource: restit.resource.Resource*)

Register a resource that should be documented.

Note: Only use this function if you want to generate the API specification outside your app.

Parameters **resource** (Resource) – The resource that should be registered

class restit.open_api.**InfoObject**(*title: str*, *version: str*, *description: str = None*, *terms_of_service: str = None*, *contact: restit.open_api.contact_object.ContactObject = None*, *license: restit.open_api.license_object.LicenseObject = None*)

Holds the OpenApi documentation InfoObject.

The object provides metadata about the API. The metadata MAY be used by the clients if needed, and MAY be presented in editing or documentation generation tools for convenience.

Parameters

- **title** (*str*) – The title of the API
- **version** (*str*) – The version of the OpenAPI document

- **description** (*str*) – A short description of the API. CommonMark syntax MAY be used for rich text representation
- **terms_of_service** (*str*) – A URL to the Terms of Service for the API. MUST be in the format of a URL
- **contact** (ContactObject) – The contact information for the exposed API
- **license** (LicenseObject) – The license information for the exposed API

class restit.open_api.**LicenseObject**(*name: str*, *url: str = None*)
License information for the exposed API.

Parameters

- **name** (*str*) – The license name used for the API
- **url** (*str*) – A URL to the license used for the API. MUST be in the format of a URL

class restit.open_api.**ContactObject**(*name: str*, *url: str*, *email: str*)
Contact information for the exposed API.

Parameters

- **name** (*str*) – The identifying name of the contact person/organization
- **url** (*str*) – The URL pointing to the contact information. MUST be in the format of a URL
- **email** (*str*) – The email address of the contact person/organization. MUST be in the format of an email address

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index