DRF OpenAPI Documentation

Release 1.3.0

Lim H.

Jan 02, 2018

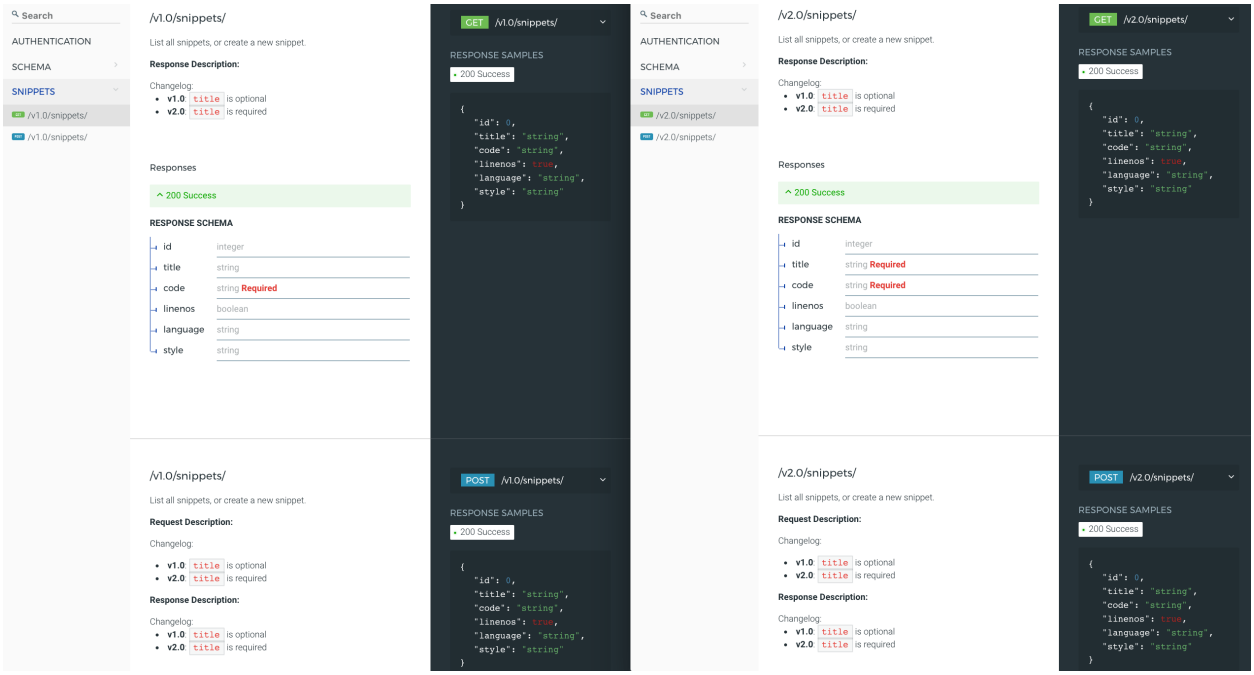
Contents

Contents:

CHAPTER 1

DRF OpenAPI

Generates OpenAPI-compatible schema from API made with Django Rest Framework. Use ReDoc as default interface instead of Swagger. First-class support for API versioning changelog & method-specific schema definition.



Contents

1.1 1. Background

Django Rest Framework has an API schema generation/declaration mechanism provided by coreapi standard. There are a couple of problems with the current ecosystem:

- CoreAPI is not compatible out of the box with OpenAPI which is a much more popular API standard with superior tooling support, i.e. Swagger et. al.
- The OpenAPI codec (compatibility layer) that CoreAPI team provides drops / doesn't support a number of useful OpenAPI features.
- There is no support for versioning or method-specific schema.

1.2 2. Requirements:

This project was born to bridge the gap between DRF and OpenAPI. The high-level requirements are as followed:

- Can be dropped into any existing DRF project without any code change necessary.
- Provide clear disctinction between request schema and response schema.
- Provide a versioning mechanism for each schema. Support defining schema by version range syntax, e.g. `>1.0, <=2.0`
- Support multiple response codes, not just `200`
- All this information should be bound to view methods, not view classes.

It's important to stress the non-intrusiveness requirement, not least because I want to minimize what I will have to change when DRF itself decides to support OpenAPI officially, if at all.

1.3 3. Design

- **Schema are automatically generated from serializers**
  - From here onwards, `schema` and `serializer` are used interchangably
- Versioned schema is supported by extending `VersionedSerializers`.
- Metadata, i.e. versioning, response and request schema, are bound to a view method through the `view_config` decorator.
- Extra schema information such as response status codes and their descriptions are bound to the serializer `Meta` class
- Automatic response validation is optionally provided `view_config(response_serializer=FooSerializer, validate_response=True)`

1.4 4. Constraints

Currently DRF OpenAPI only supports DRF project that has versioning enabled. I have only tested URLPathVersioning but I intend to suppor the full range of versioning scheme supported by DRF.

1.5 5. Examples

Please read the docs for a quickstart.

Also I have recreated the example in DRF tutorial with OpenAPI schema enabled in examples/.

1.6 6. License

MIT

CHAPTER 2

Installation

2.1 Stable release

To install DRF OpenAPI, run this command in your terminal:

```
$ pip install drf_openapi
```

This is the preferred method to install DRF OpenAPI, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for DRF OpenAPI can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/limdauto/drf_openapi
```

Or download the tarball:

```
$ curl  -OL https://github.com/limdauto/drf_openapi/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

3.1 1. Quickstart

```
# in settings.py
INSTALLED_APPS = [
    ...
    'drf_openapi'
]
REST_FRAMEWORK = {
    'DEFAULT_VERSIONING_CLASS': 'rest_framework.versioning.URLPathVersioning'
}

# in urls.py
urlpatterns += [url(f'{API_PREFIX}/', include('drf_openapi.urls'))]
```

And voila! Your API documentation will be available at `/<API_PREFIX>/schema`

3.2 2. Add schema to a view method

DRF OpenAPI support the separation of response schema and request schema on a per view method basis through the use of a *view_config* decorator

```
from drf_openapi.utils import view_config

class SnippetList(APIView):
"""
List all snippets, or create a new snippet.
"""

    @view_config(response_serializer=SnippetSerializer)
    def get(self, request, version, format=None):
        snippets = Snippet.objects.all()
```

```
        res = self.response_serializer(snippets, many=True)
        res.is_valid(raise_exception=True)
        return Response(res.validated_data)

    @view_config(request_serializer=SnippetSerializer, response_
↪serializer=SnippetSerializer)
    def post(self, request, version, format=None):
        req = self.request_serializer(data=request.data)
        req.is_valid(raise_exception=True)
        req.save()
        res = self.response_serializer(req.data)
        res.is_valid(raise_exception=True)
        return Response(res.validated_data, status=status.HTTP_201_CREATED)
```

3.3 3. Add version to schema

DRF OpenAPI support schema versioning through versioning the serializers that the schema are generated from. To make a serializer version-specific, extends `VersionedSerializers`

```
from drf_openapi.entities import VersionedSerializers
from rest_framework import serializers


class SnippetSerializerV1(serializers.Serializer):
    title = serializers.CharField(required=False, allow_blank=True, max_length=100)


class SnippetSerializerV2(SnippetSerializerV1):
    title = serializers.CharField(required=True, max_length=100)


class SnippetSerializer(VersionedSerializers):
    """
    Changelog:

    * **v1.0**: `title` is optional
    * **v2.0**: `title` is required
    """

    VERSION_MAP = (
        ('>=1.0, <2.0', SnippetSerializerV1),
        ('>=2.0', SnippetSerializerV2),
    )
```

That's it. The `view_config` decorator will be able to correctly determined what serializer to use based on the request version at run time.

3.4 4. Add response status code to schema

By default, the response serializer's fields and docstring, if specified, are associated with the `200` status code. Support for error status codes is provided through the use of `Meta` class in the serializer.

```
from rest_framework.status import HTTP_400_BAD_REQUEST

class SnippetSerializerV1(serializers.Serializer):
    title = serializers.CharField(required=False, allow_blank=True, max_length=100)

    class Meta:
        error_status_codes = {
            HTTP_400_BAD_REQUEST: 'Bad Request'
        }
```

In later iteration, I will add support for sample error response.

3.5 5. Customization of the API View

You can customize the API View that renders the schema documentation by subclassing it. It's important to note that it is just a DRF APIView so it inherits all attributes available in an APIView. Therefore, if you want to customize the permissions to allow public access to your API documentation for example, which by default is staff-only IsAdminUser, you can do the following

```
# in your.project.views
from rest_framework import permissions
from drf_openapi.views import SchemaView

class MySchemaView(SchemaView):
    permission_classes = (permissions.AllowAny,)

# in your.project.urls
from your.project.views import MySchemaView
url('schema/$', MySchemaView.as_view(title='My Awesome API'), name='api_schema')
```

Take a look at the example project to see the default URL handler in action.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/limdauto/drf_openapi/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

DRF OpenAPI could always use more documentation, whether as part of the official DRF OpenAPI docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/limdauto/drf_openapi/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *drf_openapi* for local development.

1. Fork the *drf_openapi* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/drf_openapi.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv drf_openapi
$ cd drf_openapi/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 drf_openapi tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/limdauto/drf_openapi/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_drf_openapi
```

CHAPTER 5

Credits

5.1 Development Lead

- Lim H. <limdauto@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2017-07-01)

- First release on PyPI.

6.2 0.7.0 (2017-07-28)

- Implement `VersionedSerializer`
- Implement `view_config`
- Make the library an installable Django app

6.3 0.8.0 (2017-07-28)

- Some minor fixes to make sure it works on generic project
- Add examples

6.4 0.8.1 (2017-07-28)

- Fix bug when parsing empty docstring of the serializer

6.5 0.9.0 (2017-07-28)

- Rename base `VersionedSerializer` into `VersionedSerializers`

6.6 0.9.1 (2017-07-28)

- Fix import issue after renaming

6.7 0.9.3 (2017-08-05)

- Add support for different response status codes (Issue 27)

6.8 0.9.5 (2017-08-12)

- Add Python 2.7 compatibility (thanks tuffnatty)
- Add support for ModelViewSet (thanks tuffnatty)

6.9 0.9.6 (2017-08-12)

- Fix type display for child of ListSerializer/ListField (Issue 28)

6.10 0.9.7 (2017-09-12)

- Improve permission for schema view (Issue 31)

6.11 0.9.8 (2017-10-01)

- Turn schema view into a class-based view for easier customization

6.12 0.9.9 (2017-10-01)

- Another fix for ListSerializer/ListField (Issue 28)

6.13 1.0.1 (2017-12-14)

- Fix DRF 3.7 compatibility issue
- Added (werwty) as a maintainer

6.14 1.1.0 (2017-12-14)

- Fix viewset that doesn't have pagination_class (Issue 84) and (Issue 92)

6.15 1.2.0 (2017-12-20)

- Make serializer_class optional (Issue 57)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search