# django-rest-swagger Documentation

***Release 0.3.10***

**Marc Gibbons**

July 20, 2016

Contents:

CHAPTER 1

# SWAGGER_SETTINGS

A dictionary containing all configuration of django-rest-swagger.

Example:

```
SWAGGER_SETTINGS = {
    'exclude_url_names': [],
    'exclude_namespaces': [],
    'api_version': '0.1',
    'api_path': '/',
    'relative_paths': False,
    'enabled_methods': [
        'get',
        'post',
        'put',
        'patch',
        'delete'
    ],
    'api_key': '',
    'is_authenticated': False,
    'is_superuser': False,
    'unauthenticated_user': 'django.contrib.auth.models.AnonymousUser',
    'permission_denied_handler': None,
    'resource_access_handler': None,
    'base_path':'helloreverb.com/docs',
    'info': {
        'contact': 'apiteam@wordnik.com',
        'description': 'This is a sample server Petstore server. '
                       'You can find out more about Swagger at '
                       '<a href="http://swagger.wordnik.com">'
                       'http://swagger.wordnik.com</a> '
                       'or on irc.freenode.net, #swagger. '
                       'For this sample, you can use the api key '
                       '"special-key" to test '
                       'the authorization filters',
        'license': 'Apache 2.0',
        'licenseUrl': 'http://www.apache.org/licenses/LICENSE-2.0.html',
        'termsOfServiceUrl': 'http://helloreverb.com/terms/',
        'title': 'Swagger Sample App',
    },
    'doc_expansion': 'none',
}
```

## 1.1 api_version

version of your api.

Defaults to `''`

## 1.2 api_path

path to your api. url protocol and domain is taken from django settings, so do not include those in here.

Defaults to `'/'`

## 1.3 api_key

an api key

Defaults to `''`

## 1.4 base_path

the url to where your main Swagger documentation page will live without the protocol. Optional.

If not provided, it will generate the base_path from the `request.get_host()` method.

## 1.5 doc_expansion

The docExpansion parameter as defined in the Swagger UI spec. Potential values include "none", "list", or "full".

Defaults to `'none'`

## 1.6 enabled_methods

The methods that can be interacted with in the UI

Default: `['get', 'post', 'put', 'patch', 'delete']`

## 1.7 exclude_url_names

list URL names to ignore

Default: `[]`

## 1.8 exclude_namespaces

list URL namespaces to ignore

Default: `[]`

## 1.9 info

**specify the info object per** https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#513-info-object

## 1.10 is_authenticated

set to True to enforce user authentication

Default: `False`

## 1.11 is_superuser

set to True to enforce admin only access

Default: `False`

## 1.12 unauthenticated_user

Sets the class that is used for the user in unauthenticated requests.

set to None to specify no user class

Default: `django.contrib.auth.models.AnonymousUser`

## 1.13 permission_denied_handler

custom handler for permission denied on attempting to access swagger.

Takes a callable or a string that names a callable.

Default: `None`

Example:

```
SWAGGER_SETTINGS = {
    'permission_denied_handler': 'app.views.permission_denied_handler'
}
```

Then in app/views.py:

```
def permission_denied_handler(request):
    from django.http import HttpResponse
    return HttpResponse('you have no permissions!')
```

## 1.14 relative_paths

set to True to make API paths relative to specified `api_path`.

Default: `False`

## 1.15 resource_access_handler

custom handler for delegating access rules to the project.

Takes a callable or a string that names a callable with the following signature:

```
def resource_access_handler(request, resource)
```

**The handler must accept the following arguments:**

***request* (django.http.HttpRequest): The request for documentation, providing the user and any** other relevant details about the user who is making the HTTP request.

***resource* (str): The path to the API endpoint for which to approve or reject authorization. Does not have** leading/trailing slashes.

The handler should return a truthy value when the resource is accessible in the context of the current request.

Default: `None`

Example:

```
SWAGGER_SETTINGS = {
    'resource_access_handler': 'app.views.resource_access_handler'
}
```

Then in app/views.py:

```
from django.core.urlresolvers import resolve

from .flags import flag_is_active


def resource_access_handler(request, resource):
    """ Callback for resource access. Determines who can see the documentation for which API. """
    # Superusers and staff can see whatever they want
    if request.user.is_superuser or request.user.is_staff:
        return True
    else:
        if isinstance(resource, basestring):
            try:
                resolver_match = resolve('/{}/'.format(resource))
                view = resolver_match.func
            except Exception:
                return False
        else:
            view = resource.callback

        view_attributes = view.func_dict
        feature_flag = view_attributes.get('feature_flag')

        # Hide documentation for disabled features
        if feature_flag and not flag_is_active(request, feature_flag):
            return False
        else:
            return True
```

## 1.16 token_type

Overrides authorization token type.

Default: `'Token'`

CHAPTER 2

# YAML Docstring

Docstring parser powered by YAML syntax

This parser allows you override some parts of automatic method inspection behaviours.

Example:

```
@api_view(["POST"])
def foo_view(request):
    """
    Your docs
    ---
    # YAML (must be separated by `---`)

    type:
      name:
        required: true
        type: string
      url:
        required: false
        type: url
      created_at:
        required: true
        type: string
        format: date-time

    serializer: .serializers.FooSerializer
    omit_serializer: false
    many: true

    parameters_strategy: merge
    omit_parameters:
        - path
    parameters:
        - name: name
          description: Foobar long description goes here
          required: true
          type: string
          paramType: form
        - name: other_foo
          paramType: query
        - name: other_bar
          paramType: query
        - name: avatar
          type: file
    responseMessages:
        - code: 401
          message: Not authenticated

    consumes:
        - application/json
        - application/xml
    produces:
        - application/json
        - application/xml
    """
    ...
```

## 2.1 parameters

Define parameters and their properties in docstrings:

```
parameters:
    - name: some_param
      description: Foobar long description goes here
      required: true
      type: integer
      paramType: form
    - name: other_foo
      paramType: query
    - name: avatar
      type: file
```

For the fields allowed in each parameter, see the Parameter Object fields and the Data Type Fields.

Exceptions: *$ref* is not currently supported.

## 2.2 parameters meta-fields

### 2.2.1 pytype

If you have a Django Rest Framework serializer that you would like to use to populate `type` you can specify it with `pytype`:

```
pytype: .serializers.FooSerializer
```

## 2.3 Overriding parameters

### 2.3.1 parameters_strategy

It is possible to override parameters discovered by method inspector by defining: *parameters_strategy* option to either *merge* or *replace*

To define different strategies for different *paramType*'s use the following syntax:

```
parameters_strategy:
    form: replace
    query: merge
```

By default strategy is set to *merge*

### 2.3.2 omit_parameters

Sometimes the method inspector produces a list of parameters that you might not want to see in SWAGGER form. To handle this situation define *paramTypes* that should be omitted

```
omit_parameters:
    - form
```

## 2.4 Serializers

You can explicitly specify the serializer:

```
serializer: some.package.FooSerializer
```

*serializer* can take a relative path, or no path. Lookup begins in the module of the view:

```
serializer: .package.FooSerializer

serializer: FooSerializer
```

You can specify different serializers for request and response:

```
request_serializer: some.package.FooSerializer
response_serializer: some.package.BarSerializer
```

You can prevent django-rest-swagger from using any serializer:

```
omit_serializer: true
```

## 2.5 type

If your view does not use a serializer at all but instead outputs a simple data type such as JSON you may define a custom response object in the method signature as follows:

```
type:
  name:
    required: true
    type: string
  url:
    required: false
    type: url
```

## 2.6 many

In cases where an API response is a list of objects, it is possible to mark this to django-rest-swagger by overriding `many` to *True*.

```
many: true
```

This overrides the `type` returned to be an array of the resolved API type. ViewSet `list` methods do not require this definition, and are marked as `many` automatically.

## 2.7 responseMessages

To document error codes that your APIView might throw you can define them in `responseMessages`:

```
responseMessages:
    - code: 401
      message: Not authenticated
    - code: 403
      message: Insufficient rights to call this procedure
```

## 2.8 Media Types

To document supported media types as input or output you can define them as `consumes` and/or `produces`, respectively

```
consumes:
    - application/json
    - application/xml
produces:
    - application/json
    - application/xml
```

## 2.9 Different models for reading and writing operations

REST Framework does not output write_only fields in responses and also does not require read_only fields to be provided. It is worth to automatically register 2 separate models for reading and writing operations.

The discovered serializer will be registered with *Write* or *Read* prefix. Response Class will be automatically adjusted if serializer class was detected by method inspector.

You can also refer to these models in your parameters:

```
parameters:
    - name: CigarSerializer
      type: WriteCigarSerializer
      paramType: body
```

## 2.10 view_mocker

Specify a function to modify (or replace entirely) the view that django-rest-swagger uses to introspect serializer class.

django-rest-swagger passes this function a view object, and expects a view object to be returned, or None, in which case this bit of introspection is skipped.

```
class ViewMockerNeedingAPI(ListCreateAPIView):
    def get_serializer_class(self):
        if self.request.tacos == 'tasty':
            return CommentSerializer
        else:
            return QuerySerializer

    def post(self, request, *args, **kwargs):
        """
        ---
        view_mocker: my_view_mocker
        """
        return super(ViewMockerNeedingAPI, self).post(request, *args, **kwargs)
```

```
def my_view_mocker(view):
    view.request.tacos = 'tasty'
    return view
```

CHAPTER 3

# Miscellaneous

## 3.1 Markdown

django-rest-swagger will parse docstrings as markdown if Markdown is installed.

## 3.2 reStructuredText

django-rest-swagger can be configured to parse docstrings as reStructuredText.

Add to your settings:

```
REST_FRAMEWORK = {
    'VIEW_DESCRIPTION_FUNCTION': 'rest_framework_swagger.views.get_restructuredtext'
}
```

## 3.3 Swagger 'nickname' attribute

By default, django-rest-swagger uses django-rest-framework's get_view_name to resolve the *nickname* attribute of a Swagger operation. You can specify an alternative function for *nickname* resolution using the following setting:

```
REST_FRAMEWORK = {
    'VIEW_NAME_FUNCTION': 'module.path.to.custom.view.name.function'
}
```

This function should use the following signature:

```
view_name(cls, suffix=None)
```

-`cls` The view class providing the operation.

-`suffix` The string name of the class method which is providing the operation.

## 3.4 Swagger 'list' views

django-rest-swagger introspects your views and viewset methods in order to determine the serializer used.

In the majority of cases, the object returned is a single type. However, there are times where multiple serialized objects can be returned, such as in the case of *list* methods.

When you use ViewSets, django-rest-swagger will report that the *list* method on a viewset returns a list of objects.

For other ViewSet methods or function based views, you can also hint to django-rest-swagger that the view response is also a list, rather than a single object. See *many*

CHAPTER 4

# Examples

## 4.1 Basic Example with a ViewSet

Consider the following ViewSet:

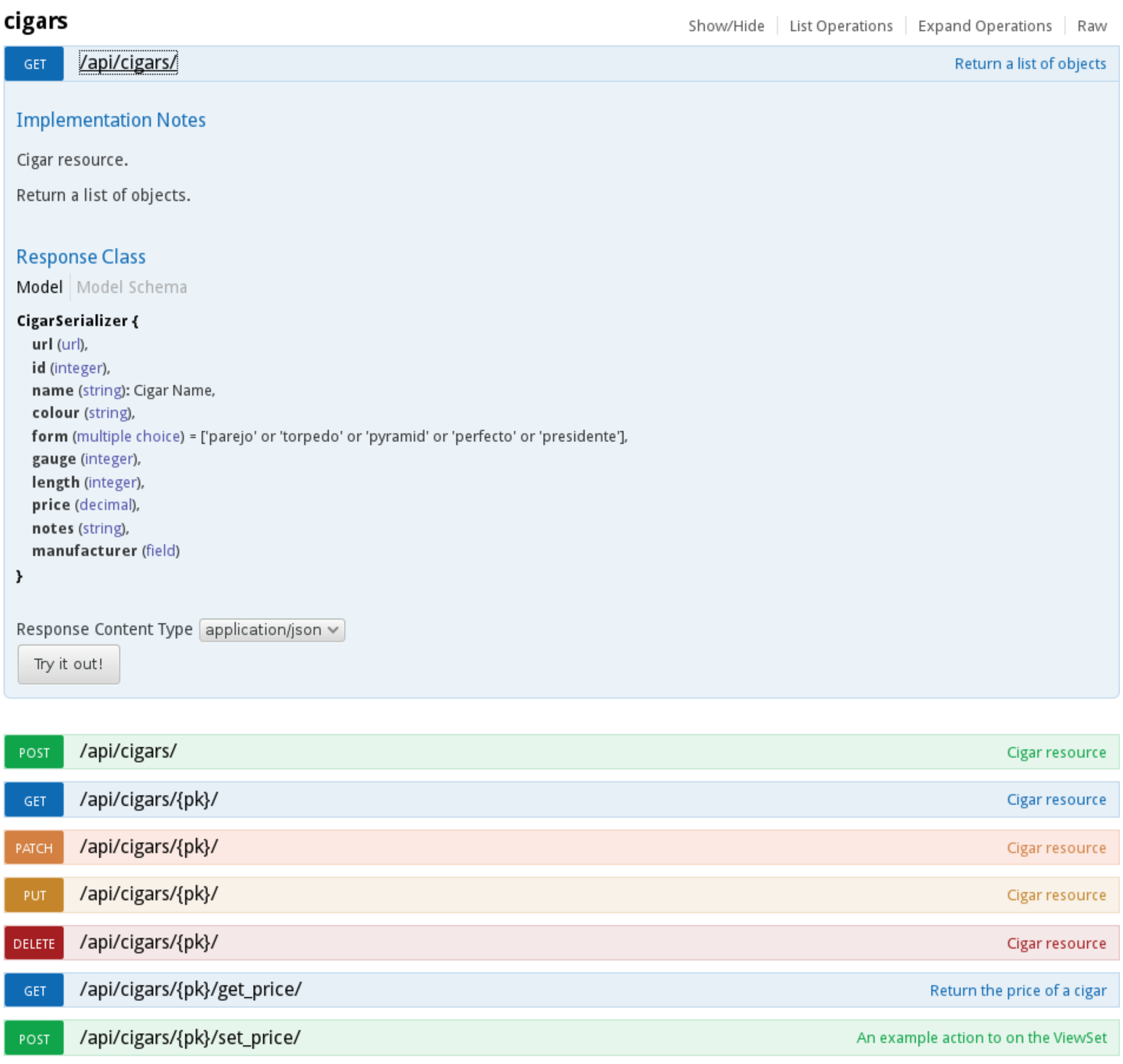
```
class CigarViewSet(viewsets.ModelViewSet):

    """ Cigar resource. """

    serializer_class = CigarSerializer
    model = Cigar
    queryset = Cigar.objects.all()

    def list(self, request, *args, **kwargs):
        """
        Return a list of objects.

        """
        return super(CigarViewSet, self).list(request, *args, **kwargs)

    @action()
    def set_price(self, request, pk):
        """An example action to on the ViewSet."""
        return Response('20$')

    @link()
    def get_price(self, request, pk):
        """Return the price of a cigar."""
        return Response('20$')
```

with supporting model and serializer:

```
class Cigar(models.Model):
    FORM_CHOICES = (
        ('parejo', 'Parejo'),
        ('torpedo', 'Torpedo'),
        ('pyramid', 'Pyramid'),
        ('perfecto', 'Perfecto'),
        ('presidente', 'Presidente'),
    )
    name = models.CharField(max_length=25, help_text='Cigar Name')
    colour = models.CharField(max_length=30, default="Brown")
    form = models.CharField(max_length=20, choices=FORM_CHOICES, default='parejo')
    gauge = models.IntegerField()
    length = models.IntegerField()
    price = models.DecimalField(decimal_places=2, max_digits=5)
    notes = models.TextField()
    manufacturer = models.ForeignKey('Manufacturer')

    def get_absolute_url(self):
        return "/api/cigars/%i/" % self.id
```

```
class CigarSerializer(serializers.ModelSerializer):
    url = fields.URLField(source='get_absolute_url', read_only=True)

    class Meta:
        model = models.Cigar
```

From this code, django-rest-swagger will produce the following swagger docs:



cigars

GET /api/cigars/ Return a list of objects

Implementation Notes

Cigar resource.

Return a list of objects.

Response Class

```
CigarSerializer {
  url (url),
  id (integer),
  name (string): Cigar Name,
  colour (string),
  form (multiple choice) = ['parejo' or 'torpedo' or 'pyramid' or 'perfecto' or 'presidente'],
  gauge (integer),
  length (integer),
  price (decimal),
  notes (string),
  manufacturer (field)
}
```

Response Content Type application/json

POST /api/cigars/ Cigar resource

GET /api/cigars/{pk}/ Cigar resource

PATCH /api/cigars/{pk}/ Cigar resource

PUT /api/cigars/{pk}/ Cigar resource

DELETE /api/cigars/{pk}/ Cigar resource

GET /api/cigars/{pk}/get_price/ Return the price of a cigar

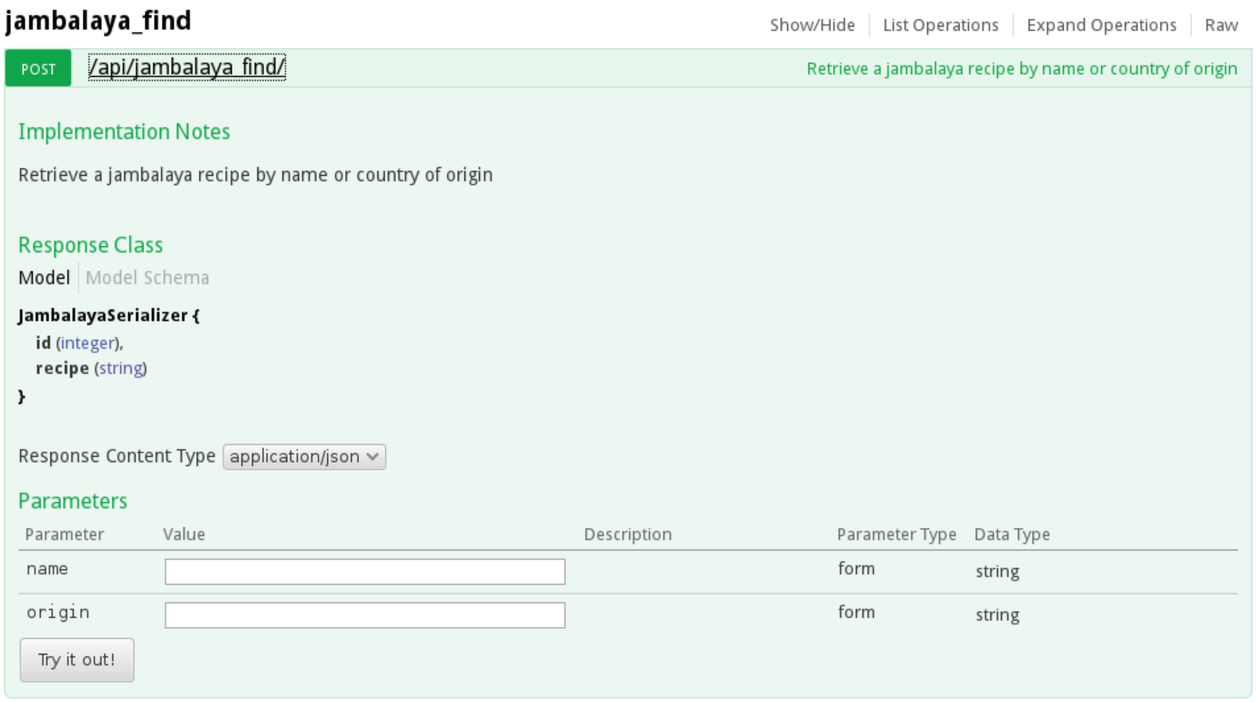
POST /api/cigars/{pk}/set_price/ An example action to on the ViewSet

## 4.2 Function Based Views

django-rest-swagger also supports function based views. Since the serializers used by a function based view are not readily introspect-able, you can use the yaml parser to manually provide them.

This example also illustrates support for markdown.

```
def find_jambalaya(request):
    """
    Retrieve a *jambalaya* recipe by name or country of origin
    ---
    request_serializer: JambalayaQuerySerializer
    response_serializer: JambalayaSerializer
    """
    if request.method == 'POST':
        serializer = JambalayaQuerySerializer(data=request.DATA)
        if serializer.data['name'] is not None:
            j = Jambalaya.objects.filter(recipe__contains='name=%s' % serializer.data['name'])
        else:
            j = Jambalaya.objects.filter(recipe__contains="country=%s" % serializer.data['origin'])
        serializer = JambalayaSerializer(j, many=True)
        return Response(serializer.data)
    else:
        return Response("", status=status.HTTP_400_BAD_REQUEST)
```



jambalaya_find

/api/jambalaya_find/ — Retrieve a jambalaya recipe by name or country of origin

Implementation Notes

Retrieve a jambalaya recipe by name or country of origin

Response Class

Model | Model Schema

```
JambalayaSerializer {
  id (integer),
  recipe (string)
}
```

Response Content Type application/json

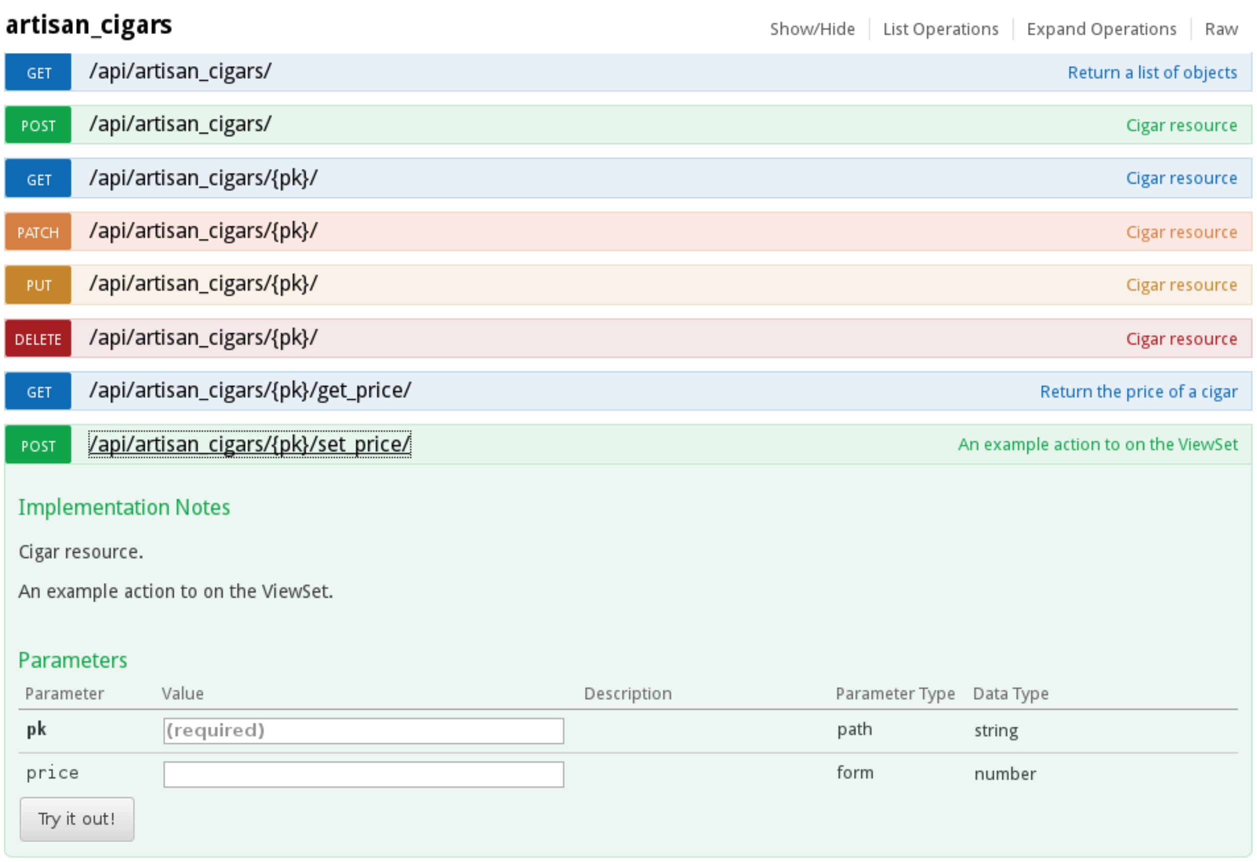
Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| name | | | form | string |
| origin | | | form | string |

## 4.3 YAML in the class docstring

You can put yaml in the class-level docstring of your view. It must be nested in the name of the method of interest:

```
class ArtisanCigarViewSet(viewsets.ModelViewSet):

    """
    Cigar resource.
    ---
    get_price:
        omit_serializer: true
    set_price:
        omit_serializer: true
        parameters_strategy:
            form: replace
        parameters:
            - name: price
              type: number
    """

    serializer_class = CigarSerializer
    model = Cigar
    queryset = Cigar.objects.all()

    def list(self, request, *args, **kwargs):
        """
        Return a list of objects.

        """
        return super(ArtisanCigarViewSet, self).list(request, *args, **kwargs)

    @action()
    def set_price(self, request, pk):
        """An example action to on the ViewSet."""
        return Response('20$')

    @link()
    def get_price(self, request, pk):
        """Return the price of a cigar."""
        return Response('20$')
```



artisan_cigars

Show/Hide | List Operations | Expand Operations | Raw

| | | |
|---|---|---|
| GET | /api/artisan_cigars/ | Return a list of objects |
| POST | /api/artisan_cigars/ | Cigar resource |
| GET | /api/artisan_cigars/{pk}/ | Cigar resource |
| PATCH | /api/artisan_cigars/{pk}/ | Cigar resource |
| PUT | /api/artisan_cigars/{pk}/ | Cigar resource |
| DELETE | /api/artisan_cigars/{pk}/ | Cigar resource |
| GET | /api/artisan_cigars/{pk}/get_price/ | Return the price of a cigar |
| POST | /api/artisan_cigars/{pk}/set_price/ | An example action to on the ViewSet |

Implementation Notes

Cigar resource.

An example action to on the ViewSet.

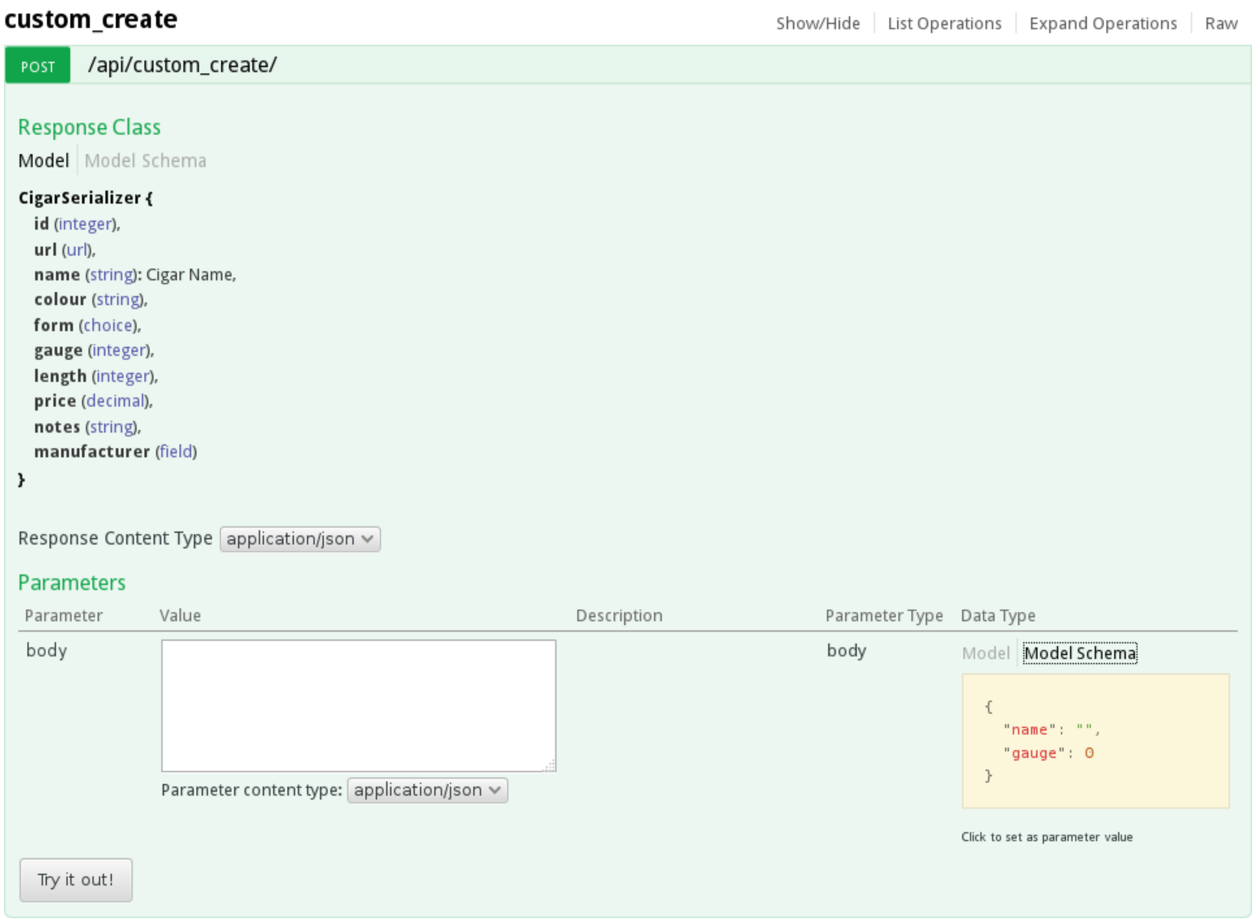
Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| **pk** | (required) | | path | string |
| price | | | form | number |

Try it out!

## 4.4 Raw JSON objects

```python
def create_cigar2(request):
    """
    ---
    response_serializer: CigarSerializer
    parameters:
        - name: body
          pytype: CigarSerializerMinimal
          paramType: body
    """
    in_serializer = CigarSerializerMinimal(data=request.DATA)
    if in_serializer.is_valid():
        cigar = Cigar()
        cigar.name = in_serializer.data['name']
        cigar.gauge = in_serializer.data['gauge']
        cigar.length = 2
        cigar.price = 2
        manufacturer = Manufacturer.objects.first()
        if manufacturer is None:
            manufacturer = Manufacturer()
            manufacturer.name = 'Taco tobacco'
            country = Country.objects.first()
            if country is None:
                country = Country()
                country.name = "Watchacallistan"
                country.save()
            manufacturer.country = country
            manufacturer.save()
        cigar.manufacturer = manufacturer
        cigar.save()
        out_serializer = CigarSerializer(cigar)
        return Response(out_serializer.data,
                        status=status.HTTP_201_CREATED)
    return Response(in_serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```



```
CigarSerializer {
  id (integer),
  url (url),
  name (string): Cigar Name,
  colour (string),
  form (choice),
  gauge (integer),
  length (integer),
  price (decimal),
  notes (string),
  manufacturer (field)
}
```

Response Content Type application/json

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| body | Parameter content type: application/json | | body | Model Model Schema `{ "name": "", "gauge": 0 }` Click to set as parameter value |

CHAPTER 5

# Overview

Django REST Swagger is a library that generates Swagger documentation from your Django Rest Framework API code.

Supports Swagger 1.2.

CHAPTER 6

# Quickstart

1. Add `rest_framework_swagger` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    'rest_framework_swagger',
)
```

2. Include the rest_framework_swagger URLs to a path of your choice

```
patterns = ('',
    ...
    url(r'^docs/', include('rest_framework_swagger.urls')),
)
```

Further configuration can be made via SWAGGER_SETTINGS in your project's *settings.py*.