drf-spectacular

T. Franzel

May 10, 2021

CONTENTS

Sane and flexible OpenAPI 3 schema generation for Django REST framework.

Documentation is an integral part of API development and OpenAPI 3 is finally here to make that process a easier. By using drf-spectacular with Django REST Framework (DRF), your schema and therefore your documentation & client will always stay close to your API.

drf-spectacular works well out of the box, but also provides you with several easy ways to customize the generated OpenAPI 3 schema. It is explicitly designed to work well for documentation (SwaggerUI, ReDoc) and automatic client generation.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 drf-spectacular

Sane and flexible OpenAPI 3.0 schema generation for Django REST framework.

This project has 3 goals:

1. Extract as much schema information from DRF as possible.
2. Provide flexibility to make the schema usable in the real world (not only toy examples).
3. Generate a schema that works well with the most popular client generators.

The code is a heavily modified fork of the DRF OpenAPI generator, which is/was lacking all of the below listed features.

Features

- Serializers modelled as components. (arbitrary nesting and recursion supported)
- @extend_schema **decorator for customization of APIView, Viewsets, function-based views, and `@action`**
  - additional parameters
  - request/response serializer override (with status codes)
  - polymorphic responses either manually with `PolymorphicProxySerializer` helper or via `rest_polymorphic`'s PolymorphicSerializer)
  - … and more customization options
- Authentication support (DRF natives included, easily extendable)
- Custom serializer class support (easily extendable)
- `MethodSerializerField()` type via type hinting or `@extend_schema_field`
- i18n support
- Tags extraction
- Request/response/parameter examples
- Description extraction from `docstrings`
- Sane fallbacks
- Sane `operation_id` naming (based on path)
- Schema serving with `SpectacularAPIView` (Redoc and Swagger-UI views are also available)

- Optional input/output serializer component split
- **Included support for:**
  - django-polymorphic / django-rest-polymorphic
  - SimpleJWT
  - DjangoOAuthToolkit
  - djangorestframework-jwt (tested fork drf-jwt)
  - dj-rest-auth (maintained fork of django-rest-auth)
  - djangorestframework-camel-case (via postprocessing hook `camelize_serializer_fields`)
  - django-filter

For more information visit the documentation.

1.1.1 License

Provided by T. Franzel, Cashlink Technologies GmbH. Licensed under 3-Clause BSD.

1.1.2 Requirements

- Python >= 3.6
- Django (2.2, 3.1, 3.2)
- Django REST Framework (3.10, 3.11, 3.12)

1.1.3 Installation

Install using `pip`...

```
$ pip install drf-spectacular
```

then add drf-spectacular to installed apps in `settings.py`

```
INSTALLED_APPS = [
    # ALL YOUR APPS
    'drf_spectacular',
]
```

and finally register our spectacular AutoSchema with DRF.

```
REST_FRAMEWORK = {
    # YOUR SETTINGS
    'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
}
```

drf-spectacular ships with sane default settings that should work reasonably well out of the box. It is not necessary to specify any settings, but we comment to specify at least some metadata.

```
SPECTACULAR_SETTINGS = {
    'TITLE': 'Your Project API',
    'DESCRIPTION': 'Your project description',
    'VERSION': '1.0.0',
    # OTHER SETTINGS
}
```

Release management

drf-spectacular deliberately stays below version `1.x.x` to signal that every new version may potentially break you. For production we strongly recommend pinning the version and inspecting a schema diff on update.

With that said, we aim to be extremely defensive w.r.t. breaking API changes. However, we also acknowledge the fact that even slight schema changes may break your toolchain, as any existing bug may somehow also be used as a feature.

We define version increments with the following semantics. *y-stream* increments may contain potentially breaking changes to both API and schema. *z-stream* increments will never break the API and may only contain schema changes that should have a low chance of breaking you.

1.1.4 Take it for a spin

Generate your schema with the CLI:

```
$ ./manage.py spectacular --file schema.yml
$ docker run -p 80:8080 -e SWAGGER_JSON=/schema.yml -v ${PWD}/schema.yml:/schema.yml␣
↪swaggerapi/swagger-ui
```

If you also want to validate your schema add the *–validate* flag. Or serve your schema directly from your API. We also provide convenience wrappers for *swagger-ui* or *redoc*.

```
from drf_spectacular.views import SpectacularAPIView, SpectacularRedocView,␣
↪SpectacularSwaggerView
urlpatterns = [
    # YOUR PATTERNS
    path('api/schema/', SpectacularAPIView.as_view(), name='schema'),
    # Optional UI:
    path('api/schema/swagger-ui/', SpectacularSwaggerView.as_view(url_name='schema'),␣
↪name='swagger-ui'),
    path('api/schema/redoc/', SpectacularRedocView.as_view(url_name='schema'), name=
↪'redoc'),
]
```

1.1.5 Usage

drf-spectacular works pretty well out of the box. You might also want to set some metadata for your API. Just create a SPECTACULAR_SETTINGS dictionary in your `settings.py` and override the defaults. Have a look at the available settings.

The toy examples do not cover your cases? No problem, you can heavily customize how your schema will be rendered.

Customization by using `@extend_schema`

Most customization cases should be covered by the `extend_schema` decorator. We usually get pretty far with specifying `OpenApiParameter` and splitting request/response serializers, but the sky is the limit.

```
from drf_spectacular.utils import extend_schema, OpenApiParameter, OpenApiExample
from drf_spectacular.types import OpenApiTypes

class AlbumViewset(viewset.ModelViewset)
    serializer_class = AlbumSerializer

    @extend_schema(
        request=AlbumCreationSerializer
        responses={201: AlbumSerializer},
    )
    def create(self, request):
        # your non-standard behaviour
        return super().create(request)

    @extend_schema(
        # extra parameters added to the schema
        parameters=[
            OpenApiParameter(name='artist', description='Filter by artist',␣
↪required=False, type=str),
            OpenApiParameter(
                name='release',
                type=OpenApiTypes.DATE,
                location=OpenApiParameter.QUERY,
                description='Filter by release date',
                examples=[
                    OpenApiExample(
                        'Example 1',
                        summary='short optional summary',
                        description='longer description',
                        value='1993-08-23'
                    ),
                    ...
                ],
            ),
        ],
        # override default docstring extraction
        description='More descriptive text',
        # provide Authentication class that deviates from the views default
        auth=None,
        # change the auto-generated operation name
```

(continues on next page)

(continued from previous page)

```
        operation_id=None,
        # or even completely override what AutoSchema would generate. Provide raw Open␣
↪API spec as Dict.
        operation=None,
        # attach request/response examples to the operation.
        examples=[
            OpenApiExample(
                'Example 1',
                description='longer description',
                value=...
            ),
            ...
        ],
    )
    def list(self, request):
        # your non-standard behaviour
        return super().list(request)

    @extend_schema(
        request=AlbumLikeSerializer
        responses={204: None},
    )
    @action(detail=True, methods=['post'])
    def set_password(self, request, pk=None):
        # your action behaviour
```

More customization

Still not satisifed? You want more! We still got you covered. Visit customization for more information.

1.1.6 Testing

Install testing requirements.

```
$ pip install -r requirements.txt
```

Run with runtests.

```
$ ./runtests.py
```

You can also use the excellent tox testing tool to run the tests against all supported versions of Python and Django. Install tox globally, and then simply run:

```
$ tox
```

1.2 Settings

Settings are configurable in `settings.py` in the scope `SPECTACULAR_SETTINGS`. You can override any setting, otherwise the defaults below are used.

```
SPECTACULAR_DEFAULTS: Dict[str, Any] = {
    # A regex specifying the common denominator for all operation paths. If
    # SCHEMA_PATH_PREFIX is set to None, drf-spectacular will attempt to estimate
    # a common prefix. use '' to disable.
    # Mainly used for tag extraction, where paths like '/api/v1/albums' with
    # a SCHEMA_PATH_PREFIX regex '/api/v[0-9]' would yield the tag 'albums'.
    'SCHEMA_PATH_PREFIX': None,
    # Remove matching SCHEMA_PATH_PREFIX from operation path. Usually used in
    # conjunction with appended prefixes in SERVERS.
    'SCHEMA_PATH_PREFIX_TRIM': False,

    'DEFAULT_GENERATOR_CLASS': 'drf_spectacular.generators.SchemaGenerator',

    # Schema generation parameters to influence how components are constructed.
    # Some schema features might not translate well to your target.
    # Demultiplexing/modifying components might help alleviate those issues.
    #
    # Create separate components for PATCH endpoints (without required list)
    'COMPONENT_SPLIT_PATCH': True,
    # Split components into request and response parts where appropriate
    'COMPONENT_SPLIT_REQUEST': False,
    # Aid client generator targets that have trouble with read-only properties.
    'COMPONENT_NO_READ_ONLY_REQUIRED': False,

    # Configuration for serving a schema subset with SpectacularAPIView
    'SERVE_URLCONF': None,
    # complete public schema or a subset based on the requesting user
    'SERVE_PUBLIC': True,
    # include schema enpoint into schema
    'SERVE_INCLUDE_SCHEMA': True,
    # list of authentication/permission classes for spectacular's views.
    'SERVE_PERMISSIONS': ['rest_framework.permissions.AllowAny'],
    # None will default to DRF's AUTHENTICATION_CLASSES
    'SERVE_AUTHENTICATION': None,

    # Dictionary of configurations to pass to the SwaggerUI({ ... })
    # https://swagger.io/docs/open-source-tools/swagger-ui/usage/configuration/
    'SWAGGER_UI_SETTINGS': {
        'deepLinking': True,
    },
    'SWAGGER_UI_DIST': '//unpkg.com/swagger-ui-dist@3.44.0',
    'SWAGGER_UI_FAVICON_HREF': '//unpkg.com/swagger-ui-dist@3.44.0/favicon-32x32.png',

    'REDOC_DIST': '//cdn.jsdelivr.net/npm/redoc@next',

    # Append OpenAPI objects to path and components in addition to the generated objects
    'APPEND_PATHS': {},
```

(continues on next page)

(continued from previous page)

```
    'APPEND_COMPONENTS': {},

    # DISCOURAGED - please don't use this anymore as it has tricky implications that
    # are hard to get right. For authentication, OpenApiAuthenticationExtension are
    # strongly preferred because they are more robust and easy to write.
    # However if used, the list of methods is appended to every endpoint in the schema!
    'SECURITY': [],

    # Postprocessing functions that run at the end of schema generation.
    # must satisfy interface result = hook(generator, request, public, result)
    'POSTPROCESSING_HOOKS': [
        'drf_spectacular.hooks.postprocess_schema_enums'
    ],

    # Preprocessing functions that run before schema generation.
    # must satisfy interface result = hook(endpoints=result) where result
    # is a list of Tuples (path, path_regex, method, callback).
    # Example: 'drf_spectacular.hooks.preprocess_exclude_path_format'
    'PREPROCESSING_HOOKS': [],

    # enum name overrides. dict with keys "YourEnum" and their choice values "field.
↪choices"
    'ENUM_NAME_OVERRIDES': {},
    # Adds "blank" and "null" enum choices where appropriate. disable on client␣
↪generation issues
    'ENUM_ADD_EXPLICIT_BLANK_NULL_CHOICE': True,

    # function that returns a list of all classes that should be excluded from doc␣
↪string extraction
    'GET_LIB_DOC_EXCLUDES': 'drf_spectacular.plumbing.get_lib_doc_excludes',

    # Function that returns a mocked request for view processing. For CLI usage
    # original_request will be None.
    # interface: request = build_mock_request(method, path, view, original_request,␣
↪**kwargs)
    'GET_MOCK_REQUEST': 'drf_spectacular.plumbing.build_mock_request',

    # Camelize names like operationId and path parameter names
    'CAMELIZE_NAMES': False,

    # Determines if and how free-form 'additionalProperties' should be emitted in the␣
↪schema. Some
    # code generator targets are sensitive to this. None disables generic
↪'additionalProperties'.
    # allowed values are 'dict', 'bool', None
    'GENERIC_ADDITIONAL_PROPERTIES': 'dict',

    # Determines whether operation parameters should be sorted alphanumerically or just␣
↪in
    # the order they arrived. Accepts either True, False, or a callable for sort's key␣
↪arg.
    'SORT_OPERATION_PARAMETERS': True,
```

(continues on next page)

(continued from previous page)

```
    # Option for turning off error and warn messages
    'DISABLE_ERRORS_AND_WARNINGS': False,

    # General schema metadata. Refer to spec for valid inputs
    # https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.3.md#openapi-
↪object
    'TITLE': '',
    'DESCRIPTION': '',
    'TOS': None,
    # Optional: MAY contain "name", "url", "email"
    'CONTACT': {},
    # Optional: MUST contain "name", MAY contain URL
    'LICENSE': {},
    # Statically set schema version. May also be an empty string. When used together with
    # view versioning, will become '0.0.0 (v2)' for 'v2' versioned requests.
    # Set VERSION to None if only the request version should be rendered.
    'VERSION': '0.0.0',
    # Optional list of servers.
    # Each entry MUST contain "url", MAY contain "description", "variables"
    'SERVERS': [],
    # Tags defined in the global scope
    'TAGS': [],
    # Optional: MUST contain 'url', may contain "description"
    'EXTERNAL_DOCS': {},

    # Arbitrary specification extensions attached to the schema's info object.
    # https://swagger.io/specification/#specification-extensions
    'EXTENSIONS_INFO': {},

    # Oauth2 related settings. used for example by django-oauth2-toolkit.
    # https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.3.md#oauth-
↪flows-object
    'OAUTH2_FLOWS': [],
    'OAUTH2_AUTHORIZATION_URL': None,
    'OAUTH2_TOKEN_URL': None,
    'OAUTH2_REFRESH_URL': None,
    'OAUTH2_SCOPES': None,
}
```

1.2.1 Django Rest Framework settings

Some of the Django Rest Framework settings also impact the schema generation. Refer to the documentation for the version that you are using.

Settings which effect the processing of requests and data types of responses will usually be effective.

There is explicit use of these settings:

- DEFAULT_SCHEMA_CLASS
- COERCE_DECIMAL_TO_STRING

- UPLOADED_FILES_USE_URL
- URL_FORMAT_OVERRIDE
- FORMAT_SUFFIX_KWARG

The following settings are ignored:

- SCHEMA_COERCE_METHOD_NAMES

The following are known to be effective:

- SCHEMA_COERCE_PATH_PK

1.2.2 Example: SwaggerUI settings

We currently support passing through all basic SwaggerUI configuration parameters. For more customization options (e.g. JS functions), you can modify and override the SwaggerUI template in your project files.

```
SPECTACULAR_SETTINGS = {
    ...
    # available SwaggerUI configuration parameters
    # https://swagger.io/docs/open-source-tools/swagger-ui/usage/configuration/
    "SWAGGER_UI_SETTINGS": {
        "deepLinking": True,
        "persistAuthorization": True,
        "displayOperationId": True,
        ...
    },
    # available SwaggerUI versions: https://github.com/swagger-api/swagger-ui/releases
    "SWAGGER_UI_DIST": "//unpkg.com/swagger-ui-dist@3.35.1", # default
    "SWAGGER_UI_FAVICON_HREF": settings.STATIC_URL + "your_company_favicon.png", #␣
↪default is swagger favicon
    ...
}
```

1.3 Workflow & schema customization

You are not satisfied with your generated schema? Follow these steps in order to get your schema closer to your API.

Note: The warnings emitted by `./manage.py spectacular --file schema.yaml --validate` are intended as an indicator to where *drf-spectacular* discovered issues. Sane fallbacks are used wherever possible and some warnings might not even be relevant to you. The remaining issues can be solved with the following steps.

1.3.1 Step 1: `queryset` and `serializer_class`

Introspection heavily relies on those two attributes. `get_serializer_class()` and `get_serializer()` are also used if available. You can also set those on `APIView`. Even though this is not supported by DRF, *drf-spectacular* will pick them up and use them.

1.3.2 Step 2: `@extend_schema`

Decorate your view functions with the *`@extend_schema`* decorator. There is a multitude of override options, but you only need to override what was not properly discovered in the introspection.

```
class PersonView(viewsets.GenericViewSet):
    @extend_schema(
        parameters=[
          QuerySerializer,  # serializer fields is converted parameters
          OpenApiParameter("nested", QuerySerializer),  # serializer object is converted␣
↪parameter
          OpenApiParameter("queryparam1", OpenApiTypes.UUID, OpenApiParameter.QUERY),
          OpenApiParameter("pk", OpenApiTypes.UUID, OpenApiParameter.PATH), # path␣
↪variable was overridden
        ],
        request=YourRequestSerializer,
        responses=YourResponseSerializer,
        # more customizations
    )
    def retrieve(self, request, pk, *args, **kwargs)
        # your code
```

Note: `responses` can be detailed further by providing a dictionary instead. This could be for example `{201: YourRequestSerializer, ...}` or `{(200, 'application/pdf'): OpenApiTypes.BINARY, ...}`.

Note: For simple responses, you might not go through the hassle of writing an explicit serializer class. In those cases, you can simply specify the request/response with a call to *`inline_serializer`*. This lets you conveniently define the endpoint's schema inline without actually writing a serializer class.

Note: If you want to annotate methods that are provided by the base classes of a view, you have nothing to attach *`@extend_schema`* to. In those instances you can use *`@extend_schema_view`* to conveniently annotate the default implementations.

```
class XViewset(mixins.ListModelMixin, viewsets.GenericViewSet):
    @extend_schema(description='text')
    def list(self, request, *args, **kwargs)
        return super().list(request, *args, **kwargs)
```

is equivalent to

```
@extend_schema_view(
    list=extend_schema(description='text')
)
```

(continues on next page)

(continued from previous page)

```
class XViewset(mixins.ListModelMixin, viewsets.GenericViewSet):
    ...
```

Note: You may also use *@extend_schema* on views to attach annotations to all methods in that view (e.g. tags). Method annotations will take precedence over view annotation.

1.3.3 Step 3: @extend_schema_field and type hints

A custom SerializerField might not get picked up properly. You can inform *drf-spectacular* on what is to be expected with the *@extend_schema_field* decorator. It takes either basic types or a Serializer as argument. In case of basic types (e.g. str int etc.) a type hint is already sufficient.

```
@extend_schema_field(OpenApiTypes.BYTE)  # also takes basic python types
class CustomField(serializers.Field):
    def to_representation(self, value):
        return urlsafe_base64_encode(b'\xf0\xf1\xf2')
```

You can apply it also to the method of a *SerializerMethodField*.

```
class ErrorDetailSerializer(serializers.Serializer):
    field_custom = serializers.SerializerMethodField()

    @extend_schema_field(OpenApiTypes.DATETIME)
    def get_field_custom(self, object):
        return '2020-03-06 20:54:00.104248'
```

1.3.4 Step 4: @extend_schema_serializer

You may also decorate your serializer with *@extend_schema_serializer*. Mainly used for excluding specific fields from the schema or attaching request/response examples. On rare occasions (e.g. envelope serializers), overriding list detection with many=False may come in handy.

```
@extend_schema_serializer(
    exclude_fields=('single',) # schema ignore these fields
    examples = [
         OpenApiExample(
            'Valid example 1',
            summary='short summary',
            description='longer description',
            value={
                'songs': {'top10': True}
                'single': {'top10': True}
            },
            request_only=True, # signal that example only applies to requests
            response_only=False, # signal that example only applies to responses
        ),
    ]
```

(continues on next page)

(continued from previous page)

```
)
class AlbumSerializer(serializers.ModelSerializer):
    songs = SongSerializer(many=True)
    single = SongSerializer(read_only=True)

    class Meta:
        fields = '__all__'
        model = Album
```

1.3.5 Step 5: Extensions

The core purpose of extensions is to make the above customization mechanisms also available for library code. Usually, you cannot easily decorate or modify `View`, `Serializer` or `Field` from libraries. Extensions provide a way to hook into the introspection without actually touching the library.

All extensions work on the same principle. You provide a `target_class` (import path string or actual class) and then state what *drf-spectcular* should use instead of what it would normally discover.

Note: Only the first Extension matching the criteria is used. By setting the `priority` attribute on your extension, you can influence the matching order (default `0`). Built-in Extensions have a priority of `-1`. If you subclass built-in Extensions, don't forget to increase the priority.

Replace views with `OpenApiViewExtension`

Many libraries use `@api_view` or `APIView` instead of *ViewSet* or *GenericAPIView*. In those cases, introspection has very little to work with. The purpose of this extension is to augment or switch out the encountered view (only for schema generation). Simply extending the discovered class `class Fixed(self.target_class)` with a `queryset` or `serializer_class` attribute will often solve most issues.

```
class Fix4(OpenApiViewExtension):
    target_class = 'oscarapi.views.checkout.UserAddressDetail'

    def view_replacement(self):
        from oscar.apps.address.models import UserAddress

        class Fixed(self.target_class):
            queryset = UserAddress.objects.none()
        return Fixed
```

Specify authentication with `OpenApiAuthenticationExtension`

Authentication classes that do not have 3rd party support will emit warnings and be ignored. Luckily authentication extensions are very easy to implement. Have a look at the default authentication method extensions. A simple custom HTTP header based authentication could be achieved like this:

```
class MyAuthenticationScheme(OpenApiAuthenticationExtension):
    target_class = 'my_app.MyAuthentication'  # full import path OR class ref
    name = 'MyAuthentication'  # name used in the schema
```

(continues on next page)

(continued from previous page)

```
    def get_security_definition(self, auto_schema):
        return {
            'type': 'apiKey',
            'in': 'header',
            'name': 'api_key',
        }
```

Declare field output with `OpenApiSerializerFieldExtension`

This is mainly targeted to custom *SerializerField*'s that are within library code. This extension is functionally equivalent to *@extend_schema_field*

```
class CategoryFieldFix(OpenApiSerializerFieldExtension):
    target_class = 'oscarapi.serializers.fields.CategoryField'

    def map_serializer_field(self, auto_schema, direction):
        # equivalent to return {'type': 'string'}
        return build_basic_type(OpenApiTypes.STR)
```

Declare serializer magic with `OpenApiSerializerExtension`

This is one of the more involved extension mechanisms. *drf-spectacular* uses those to implement polymorphic serializers. The usage of this extension is rarely necessary because most custom `Serializer` classes stay very close to the default behaviour.

Declare custom/library filters with `OpenApiFilterExtension`

This extension only applies to filter and pagination classes and is rarely used. Built-in support for *django-filters* is realized with this extension. *OpenApiFilterExtension* replaces the filter's native `get_schema_operation_parameters` with your customized version, where you have full access to *drf-spectacular's* more advanced introspection features.

1.3.6 Step 6: Postprocessing hooks

The generated schema is still not to your liking? You are no easy customer, but there is one more thing you can do. Postprocessing hooks run at the very end of schema generation. This is how the choice `Enum` are consolidated into component objects. You can register additional hooks with the `POSTPROCESSING_HOOKS` setting.

```
def custom_postprocessing_hook(result, generator, request, public):
    # your modifications to the schema in parameter result
    return result
```

1.3.7 Step 7: Preprocessing hooks

Preprocessing hooks are applied shortly after collecting all API operations and before the actual schema generation starts. They provide an easy mechanism to alter which operations should be represented in your schema. You can exclude specific operations, prefix paths, introduce or hardcode path parameters or modify view initiation. additional hooks with the `PREPROCESSING_HOOKS` setting.

```python
def custom_preprocessing_hook(endpoints):
    # your modifications to the list of operations that are exposed in the schema
    for (path, path_regex, method, callback) in endpoints:
        pass
    return endpoints
```

Note: A common use case would be the removal of duplicated `{format}`-suffixed operations, for which we already provide the *`drf_spectacular.hooks.preprocess_exclude_path_format`* hook. You can simply enable this hook by adding the import path string to the `PREPROCESSING_HOOKS`.

1.3.8 Congratulations

You should now have no more warnings and a spectacular schema that satisfies all your requirements. If that is not the case, feel free to open an issue and make a suggestion for improvement.

1.4 Client generation

drf-spectacular aims to generate the most accurate schema possible under the constraints of OpenAPI 3.0.3. Unfortunately, sometimes this goal conflicts with generating a good and functional client.

To serve the two main use cases, i.e. documenting the API and generating clients, we opt for getting the most accurate schema first, and then provide settings that allow to resolve potential issues with client generation.

Note: TL;DR - Simply setting `'COMPONENT_SPLIT_REQUEST': True` will most likely yield the best and most accurate client.

Note: *drf-spectacular* generates warnings where it recognizes potential problems. Some warnings are important to having a correct client. Fixing all warning is highly recommended.

Note: For generating clients with CI, we highly recommend using `./manage.py spectacular --file schema.yaml --validate --fail-on-warn` to catch potential problems early on.

1.4.1 Component issues

Most client issues revolve around the construction of components. Some client targets have trouble with `readOnly` and `required` fields like `id`. Even though technically correct, the generated code may not allow creating objects with `id` missing for `POST` requests. Some fields like `FileField` behave very differently on requests and responses and are simply not translatable into a single component.

The most useful setting is `'COMPONENT_SPLIT_REQUEST': True`, where all affected components are split into request and response components. This takes care of almost all `required`, `writeOnly`, `readOnly` issues, and generally delivers code that is easier to understand and harder to misuse.

Sometimes you may only want to fix the `required/readOnly` issue without splitting all components. This can be explicitly addressed with `'COMPONENT_NO_READ_ONLY_REQUIRED': True`. Because this setting waters down the correctness of the schema, we generally recommend using `COMPONENT_SPLIT_REQUEST` instead.

`'COMPONENT_SPLIT_PATCH': True` is already enabled by default as `PATCH` and `POST` requests clash on the `required` property and cannot be adequately modeled with a single component.

Relevant settings:

```
# Split components into request and response parts where appropriate
'COMPONENT_SPLIT_REQUEST': False,
# Aid client generator targets that have trouble with read-only properties.
'COMPONENT_NO_READ_ONLY_REQUIRED': False,
# Create separate components for PATCH endpoints (without required list)
'COMPONENT_SPLIT_PATCH': True,
```

1.4.2 Enum issues

Some generator targets choke on combined enum components or having a `null` choice on a `nullable: true` field. Even though it is the correct way (according to the specification), it sadly breaks some generator targets. Setting `'ENUM_ADD_EXPLICIT_BLANK_NULL_CHOICE': False` will create a less accurate schema that tends to offend fewer generator targets.

For more information please refer to the official documentation and more specifically the specification proposal.

Relevant settings:

```
# Adds "blank" and "null" enum choices where appropriate. disable on client generation␣
↪issues
'ENUM_ADD_EXPLICIT_BLANK_NULL_CHOICE': True,
```

1.4.3 Type issues

Some generator targets behave differently depending on how `additionalProperties` is structured. According to the specification all three variations should yield identical results, which unfortunately is not the case in practice.

Relevant settings:

```
# Determines if and how free-form 'additionalProperties' should be emitted in the schema.␣
↪Some
# code generator targets are sensitive to this. None disables generic
↪'additionalProperties'.
# allowed values are 'dict', 'bool', None
'GENERIC_ADDITIONAL_PROPERTIES': 'dict',
```

1.5 FAQ

1.5.1 I use library/app *XXX* and the generated schema is wrong or broken

Sometimes DRF libraries do not cooperate well with the introspection mechanics. Check the *Extension Blueprints* for already available fixes. If there aren't any, learn how to do easy *Workflow & schema customization*. Feel free to contribute back missing fixes.

If you think this is a bug in *drf-spectacular*, open a issue.

1.5.2 I cannot use `@extend_schema` on library code

You can easily adapt introspection for libraries/apps with the `Extension` mechanism. `Extensions` provide an easy way to attach schema information to code that you cannot modify otherwise. Have a look at *Workflow & schema customization* on how to use `Extensions`

1.5.3 I get an empty schema or endpoints are missing

This is usually due to endpoint permissions or versioning.

In case you use versioning on all endpoints, that might be the intended output. By default the schema will only contain unversioned endpoints. Explicitly specify what version you want to generate.

```
./manage.py spectacular --api-version 'YOUR_VERSION'
```

This will contain unversioned endpoints together with the endpoints for the the specified version.

If that does not help, open an issue.

1.5.4 I expected a different schema

Sometimes views declare one thing (via `serializer_class` and `queryset`) and do a entirely different thing. Usually this is attributed to making a library code flexible under varying situations. In those cases it is best to override what the introspection decuded and state explicitly what is to be expected. Work through the steps in *Workflow & schema customization* to adapt your schema.

1.5.5 I get duplicated operations with a `{format}`-suffix

Your app likely uses DRF's `format_suffix_patterns`. If those operations are undesireable in your schema, you can simply exclude them with an already provided *preprocessing hook*.

1.5.6 I get a lot of warnings

The warnings are emitted to inform you of discovered schema issues. Some usage patterns like `@api_view` or `APIView` provide very little discoverable information on your API. In those cases you can easily augment those endpoints and serializers with additional information. Look at *Workflow & schema customization* options to fill those gaps and make the warnings disappear.

1.5.7 I get warnings regarding my `Enum` or my `Enum` names have a weird suffix

This is because the `Enum` postprocessing hook is activated by default. Enum suffixes like `LanguageCa22Enum` mean that there was a naming collision that got resolved. Other warnings might indicate that you use one and the same choice set under different names.

The naming mechanism will handle all encountered issues automatically, but also notify you of potential problems. You can resolve (or silence) enum issues by adding an entry to the `ENUM_NAME_OVERRIDES` setting. Values can take the form of choices (list of tuples), value lists (list of strings), or import strings. Django's `models.Choices` and Python's `Enum` classes are supported as well.

```
'ENUM_NAME_OVERRIDES': {
    # variable containing list of tuples, e.g. [('US', 'US'), ('RU', 'RU'),]
    'LanguageEnum': language_choices,
    # dedicated Enum or models.Choices class
    'CountryEnum': 'import_path.enums.CountryEnum',
    # choices is an attribute of class CurrencyContainer containing a list of tuples
    'CurrencyEnum': 'import_path.CurrencyContainer.choices',
}
```

1.5.8 My endpoints use different serializers depending on the situation

Welcome to the real world! Use *`@extend_schema`* in combination with *`PolymorphicProxySerializer`* like so:

```
class PersonView(viewsets.GenericViewSet):
    @extend_schema(responses={
        200: PolymorphicProxySerializer(
                component_name='Person',
                # on 200 either a legal or a natural person is returned
                serializers=[LegalPersonSerializer, NaturalPersonSerializer],
                resource_type_field_name='type',
        ),
        500: YourOptionalErrorSerializer,
    })
    def retrieve(self, request, *args, **kwargs)
        pass
```

1.5.9 My authentication method is not supported

You can easily specify a custom authentication with `OpenApiAuthenticationExtension`. Have a look at *Workflow & schema customization* on how to use `Extensions`

1.5.10 How can I i18n/internationalize my schema and UI?

You can use the Django internationalization as you would normally do. The workflow is as one would expect: `USE_I18N=True`, settings the languages, `makemessages`, and `compilemessages`.

The CLI tool accepts a language parameter (`./manage.py spectacular --lang="de-de"`) for offline generation. The schema view as well as the UI views accept a `lang` query parameter for explicitly requesting a language (`example.com/api/schema?lang=de`). If i18n is enabled and there is no query parameter provided, the `ACCEPT_LANGUAGE` header is used. Otherwise the translation falls back to the default language.

```python
from django.utils.translation import gettext_lazy as _

class PersonView(viewsets.GenericViewSet):
    __doc__ = _("""
    More lengthy explanation of the view
    """)

    @extend_schema(summary=_('Main endpoint for creating person'))
    def retrieve(self, request, *args, **kwargs)
        pass
```

1.5.11 FileField (ImageField) is not handled properly in the schema

In contrast to most other fields, `FileField` behaves differently for requests and responses. This duality is impossible to represent in a single component schema.

For these cases, there is an option to split components into request and response parts by setting `COMPONENT_SPLIT_REQUEST = True`. Note that this influences the whole schema, not just components with `FileFields`.

Also consider explicitly setting `parser_classes = [parsers.MultiPartParser]` (or any file compatible parser) on your *View* or write a custom *get_parser_classes*. These fields do not work with the default `JsonParser` and that fact should be represented in the schema.

1.5.12 I'm using `@action(detail=False)` but the response schema is not a list

`detail=True/False` only specifies whether the action should be routed at `x/{id}/action` or `x/action`. The `detail` parameter in itself makes no statement about the action's response. Also note that the default for underspecified endpoints is a non-list response. To signal a listed response, you can use `@extend_schema(response=XSerializer(many=True))`.

1.5.13 Using `@extend_schema` on `APIView` has no effect

`@extend_schema` needs to be applied to the entrypoint method of the view. For views derived from `Viewset`, these are methods like `retrieve`, `list`, `create`. For `APIView` based views, these are `get`, `post`, `create`. This confusion commonly occurs while using convenience classes like `ListAPIView`. `ListAPIView` does in fact have a `list` method (via mixin), but the actual entrypoint is still the `get` method, and the `list` call is proxied through the entrypoint.

1.5.14 Where should i put my extensions? / my extensions are not detected

The extensions register themselves automatically. Just be sure that the python interpreter sees them at least once. To that end, we suggest creating a `PROJECT/schema.py` file and importing it in your `PROJECT/__init__.py` (same directory as `settings.py` and `urls.py`) with `import PROJECT.schema`. Please do not import the file in `settings.py` as this may potentially lead to cyclic import issues.

1.6 Extension Blueprints

Blueprints are a collection of schema fixes for Django and REST Framework apps. Some libraries/apps do not play well with *drf-spectacular*'s automatic introspection. With extensions you can manually provide the necessary information to generate a better schema.

There is no blueprint for the app you are looking for? No problem, you can easily write extensions yourself. Take the blueprints here as examples and have a look at *Workflow & schema customization*. Feel free to contribute new ones or fixes with a PR. Blueprint files can be found here.

Note: Simply copy&paste the snippets into your codebase. The extensions register themselves automatically. Just be sure that the python interpreter sees them at least once. To that end, we suggest creating a `PROJECT/schema.py` file and importing it in your `PROJECT/__init__.py` (same directory as `settings.py` and `urls.py`) with `import PROJECT.schema`. Please do not import the file in `settings.py` as this may potentially lead to cyclic import issues. Now you are all set.

1.6.1 dj-stripe

Stripe Models for Django: dj-stripe

```python
from djstripe.contrib.rest_framework.serializers import (
    CreateSubscriptionSerializer, SubscriptionSerializer
)

from drf_spectacular.extensions import OpenApiViewExtension
from drf_spectacular.utils import extend_schema


class FixDjstripeSubscriptionRestView(OpenApiViewExtension):
    target_class = 'djstripe.contrib.rest_framework.views.SubscriptionRestView'

    def view_replacement(self):
        class Fixed(self.target_class):
            serializer_class = SubscriptionSerializer
```

(continues on next page)

(continued from previous page)

```
            @extend_schema(
                request=CreateSubscriptionSerializer,
                responses=CreateSubscriptionSerializer
            )
            def post(self, request, *args, **kwargs):
                pass

        return Fixed
```

1.6.2 django-oscar-api

RESTful API for django-oscar: django-oscar-api

```
from rest_framework import serializers

from drf_spectacular.extensions import (
    OpenApiSerializerExtension, OpenApiSerializerFieldExtension, OpenApiViewExtension
)
from drf_spectacular.plumbing import build_basic_type
from drf_spectacular.types import OpenApiTypes
from drf_spectacular.utils import OpenApiParameter, extend_schema, extend_schema_field


class Fix1(OpenApiViewExtension):
    target_class = 'oscarapi.views.root.api_root'

    def view_replacement(self):
        return extend_schema(responses=OpenApiTypes.OBJECT)(self.target_class)


class Fix2(OpenApiViewExtension):
    target_class = 'oscarapi.views.product.ProductAvailability'

    def view_replacement(self):
        from oscarapi.serializers.product import AvailabilitySerializer

        class Fixed(self.target_class):
            serializer_class = AvailabilitySerializer
        return Fixed


class Fix3(OpenApiViewExtension):
    target_class = 'oscarapi.views.product.ProductPrice'

    def view_replacement(self):
        from oscarapi.serializers.checkout import PriceSerializer

        class Fixed(self.target_class):
            serializer_class = PriceSerializer
        return Fixed
```

(continues on next page)

(continued from previous page)

```
class Fix4(OpenApiViewExtension):
    target_class = 'oscarapi.views.checkout.UserAddressDetail'

    def view_replacement(self):
        from oscar.apps.address.models import UserAddress

        class Fixed(self.target_class):
            queryset = UserAddress.objects.none()
        return Fixed


class Fix5(OpenApiViewExtension):
    target_class = 'oscarapi.views.product.CategoryList'

    def view_replacement(self):
        class Fixed(self.target_class):
            @extend_schema(parameters=[
                OpenApiParameter(name='breadcrumbs', type=OpenApiTypes.STR,␣
↪location=OpenApiParameter.PATH)
            ])
            def get(self, request, *args, **kwargs):
                pass

        return Fixed


class Fix6(OpenApiSerializerExtension):
    target_class = 'oscarapi.serializers.checkout.OrderSerializer'

    def map_serializer(self, auto_schema, direction):
        from oscarapi.serializers.checkout import OrderOfferDiscountSerializer,␣
↪OrderVoucherOfferSerializer

        class Fixed(self.target_class):
            @extend_schema_field(OrderOfferDiscountSerializer(many=True))
            def get_offer_discounts(self):
                pass

            @extend_schema_field(OpenApiTypes.URI)
            def get_payment_url(self):
                pass

            @extend_schema_field(OrderVoucherOfferSerializer(many=True))
            def get_voucher_discounts(self):
                pass

        return auto_schema._map_serializer(Fixed, direction)


class Fix7(OpenApiSerializerFieldExtension):
    target_class = 'oscarapi.serializers.fields.CategoryField'
```

(continues on next page)

(continued from previous page)

```python
    def map_serializer_field(self, auto_schema, direction):
        return build_basic_type(OpenApiTypes.STR)


class Fix8(OpenApiSerializerFieldExtension):
    target_class = 'oscarapi.serializers.fields.AttributeValueField'

    def map_serializer_field(self, auto_schema, direction):
        return {
            'oneOf': [
                build_basic_type(OpenApiTypes.STR),
            ]
        }


class Fix9(OpenApiSerializerExtension):
    target_class = 'oscarapi.serializers.basket.BasketSerializer'

    def map_serializer(self, auto_schema, direction):
        class Fixed(self.target_class):
            is_tax_known = serializers.SerializerMethodField()

            def get_is_tax_known(self) -> bool:
                pass

        return auto_schema._map_serializer(Fixed, direction)


class Fix10(Fix9):
    target_class = 'oscarapi.serializers.basket.BasketLineSerializer'
```

1.6.3 djangorestframework-api-key

Since djangorestframework-api-key has no entry in `authentication_classes`, *drf-spectacular* cannot pick up this library. To alleviate this shortcoming, you can manually add the appropriate security scheme.

Note: Usage of the `SECURITY` setting is discouraged, unless there are special circumstances like here for example. For almost all cases `OpenApiAuthenticationExtension` is strongly preferred, because `SECURITY` will get appended to every endpoint in the schema regardless of effectiveness.

```python
SPECTACULAR_SETTINGS = {
    "APPEND_COMPONENTS": {
        "securitySchemes": {
            "ApiKeyAuth": {
                "type": "apiKey",
                "in": "header",
                "name": "Authorization"
            }
```

(continues on next page)

(continued from previous page)

```
        }
    },
    "SECURITY": [{"ApiKeyAuth": [], }],
     ...
}
```

1.7 From *drf-yasg* to OpenAPI 3

drf-yasg is an excellent library and the most popular choice for generating OpenAPI 2.0 / Swagger schemas with DRF. Unfortunately, it currently does not provide OpenAPI 3 support. Migration from *drf-yasg* to *drf-spectacular* requires only minor modifications.

- `@swagger_auto_schema` is largely equivalent to *`@extend_schema`*.
- `manual_parameters` is called `parameters`
- `openapi.Parameter` is roughly equivalent to *`OpenApiParameter`*.
- `@swagger_serializer_method` is equivalent to *`@extend_schema_field`*.
- `ref_name` on Serializer `Meta` classes is supported (excluding inlining with `ref_name=None`)
- `swagger_fake_view` is available as attribute on views to signal schema generation
- `Response` is largely identical to *`OpenApiResponse`*.

1.8 License

(continues on next page)

(continued from previous page)

1.9 Changelog

1.9.1 0.16.0 (2021-05-10)

- add redoc dist setting
- bugfix mock request asymmetry #370 #250
- refactor urlpattern simplification #373 #168
- include relation PKs into SCHEMA_COERCE_PATH_PK handling #251
- allow PolymorphicProxySerializer to be simple ‘oneOf’
- bugfix incorrect PolymorphicProxySerializer warning on extend_schema_field #263
- add break-out option for SerializerFieldExtension
- Modify urls for nested routers [Matthias Erll]

Breaking changes / important additions:

- Revamped handling of mocked requests. Now `GET_MOCK_REQUEST` is always called, not just for offline schema generation. In case there is a real request available, we carry over headers and authetication. If you use your own implementation, you may want to inspect the new default implementation.
- NamespaceVersioning: switched path variable substitution from regex to custom state machine due to parethesis counting issue.
- Improved implicit support for drf-nested-routers
- Added some convenience options for plain `oneOf` to PolymorphicProxySerializer
- This release should have minimal impact on the generated schema. We opt for a y-stream release due to potentially breaking changes when a user-provided `GET_MOCK_REQUEST` is used.

1.9.2 0.15.1 (2021-04-08)

- bugfix prefix estimation with RE special char literals in path #358

Breaking changes / important additions:

- minor release to fix newly introduced default prefix estimation.

1.9.3 0.15.0 (2021-04-03)

- fix boundaries for decimals coerced to strings #335
- improve util type hints
- add convenience response wrapper OpenApiResponse #345 #272 #116
- adapt for dj-rest-auth upstream changes in iMerica/dj-rest-auth#227
- Fixed traversing of 'Optional' type annotations [Luke Plant]
- prevent pagination on error responses. #277
- fix SCHEMA_PATH_PREFIX_TRIM ^/ pitfall & remove unused old URL mounting
- slighly improve #332 for django-filter range filters
- introduce non-redundant title field. #191 #286
- improve schema version string handling including variations #303
- bugfix ENUM_NAME_OVERRIDES for categorized choices #339
- improve SCHEMA_PATH_PREFIX handling, add auto-detect default, introduce prefix trimming #336
- add support for all django-filters RangeFilter [Jules Waldhart]
- Added default value for missing attribute [Matthias Erll]
- Fix map_renderers where format is None [Matthias Erll]

Breaking changes / important additions:

- explicitly set responses via `@extend_schema` will not get paginated/listed anymore for non `2XX` status codes.
- New default `None` for `SCHEMA_PATH_PREFIX` will attempt to determine a reasonable prefix. Previous behavior is restored with `''`
- Added `OpenApiResponses` to gain access to response object descriptions.

1.9.4 0.14.0 (2021-03-09)

- Fixed bug with *cached_property* non-Model objects not being traversed [Luke Plant]
- Fixed issue #314 - include information about view/serializer in warnings. [Luke Plant]
- bugfix forward/reverse model traversal #323
- fix nested serializer detection & smarter metadata extraction #319
- add drf-yasg compatibility feature 'swagger_fake_view' #321
- fix django-filter through model edge case & catch exceptions #320
- refactor/bugfix PATCH & Serializer(partial=True) behaviour.
- bugfix django-filter custom filter class resolution #317
- bugfix django-filter for Django 2.2 AutoField
- improved/restructured resolution priority in django-filter extension #317 #234
- handle Decimals for YAML #316
- remove deprecated django-filter backend solution
- update swagger-ui version

- bugfix [] case and lint #312
- discriminate None and typing.Any usage #315
- fix multi-step source relation field resolution, again. #274 #296
- Add any type for OpenApiTypes [Andre da Silva]
- improve Extension usage documentation #307
- restructure request body for extend_schema #266 #279
- bugfix multipart boundary showing up in Accept header
- bugfix: use get_parsers() and get_renderers() #266
- Fix for better support of PEP 563 compatible annotations. [Luke Plant]
- Add document authentication [gongul]
- Do not override query params [Fabricio Aguiar]
- New setting for enabling/disabling error/warn messages [Fabricio Aguiar]
- bugfix response headers without body #297
- issue #296 [Luis Saavedra]
- Fixes #283 – implement response header parameters [Sergei Maertens]
- Added feature test for response headers [Sergei Maertens]
- robustify django-filter enum sorting #295

Breaking changes / important additions:

- *drf-spectacular*'s custom `DjangoFilterBackend` removed after previous deprecation. Just use the original class again.
- `django-filter` extension received a significant refactoring so your schema may have several changes, hopefully positive ones.
- Added response headers feature
- Extended `@extend_schema(request=X)`, where `X` may now also be a `Dict[content_type, serializer_etc]`
- Updated Swagger UI version
- Fixed several model traveral issues that may lead to PK changes in the schema
- Added *drf-yasg's* `swagger_fake_view`

1.9.5 0.13.2 (2021-02-11)

- add setting for operation parameter sorting #281
- bugfix/generalize Union hint extraction #284
- bugfix functools.partial methods in django-filters #290
- bugfix django-filter method filter #290
- Check serialzer help_text field is passed to the query description [Jorge Rodríguez-Flores Esparza]
- QUERY Parameters from serializer ignore description in SwaggerUI [Jorge Rodríguez-Flores Esparza]
- README.rst encoding change [gongul]

- Add support for SCOPES_BACKEND_CLASS setting from django-oauth-toolkit [diesieben07]
- use source instead of field_name for model field detection #274 [diesieben07]
- bugfix parameter removal from custom AutoSchema #212
- add specification extension option to info section #165
- add default to OpenApiParameter #271
- show violating view for easier fixing #278
- fix readonly related fields generating incorrect schema #274 [diesieben07]
- bugfix save parameter removal #212

1.9.6 0.13.1 (2021-01-21)

- bugfix/handle more django-filter cases #263
- bugfix missing meta on extend_serializer_field, raw schema, and breakout
- expose explode and style for OpenApiParameter #267
- Only generate mock request if there is no actual request [Matthias Erll]
- Update blueprints.rst [takizuka]
- bugfix enum substitution for enumed arrays (multiple choice)
- Update README.rst [Chad Ramos]
- Create new mock request on each operation [Matthias Erll]

1.9.7 0.13.0 (2021-01-13)

- add setting for additionalProperties handling #238
- bugfix path param extraction for PrimaryKeyRelatedField #258
- use injected django-filter help_text #234
- robustify normalization of tyes #257
- bugfix PATCH split serializer disparity #249
- django-filter description bugfix #234
- bugfix unsupported http verbs #244
- bugfix assert on methods in django-filter #252 #234 #241
- Regression: Filterset defined as method (and from a @property) are not supported [Nicolas Delaby]
- bugfix view-level AutoSchema noneffective with extend_schema #241
- bugfix incorrect warning on paginated actions #233

Breaking changes:

- several small improvements that should not have a big impact. this is a y-stream release mainly due to schema changes that may occur with `django-filter`.

1.9.8 0.12.0 (2020-12-19)

- add exclusion for discovered parameters #212
- bugfix incorrect collision warning #233
- introduce filter extensions #234
- revert Swagger UI view to single request and alternative #211 #173
- bugfix Simple JWT token refresh #232
- bugfix simple JWT serializer schema #232
- Fix enum postprocessor to allow 0 as possible value [Vikas]
- bugfix/restore optional default parameter value #226
- Include QuerySerializer in documentation [KimSoungRyoul]
- support OAS3.0 ExampleObject to @extend_schema & @extend_schema_serializer #115 [KimSoungRyoul]
- add explicit double and int32 types. #214
- added type extension for int64 format support [Peter Dreuw]
- fix TokenAuthentication handling of keyword #205
- Allow callable limit_value in schema [Serkan Hosca]
- @extend_schema responses param now accepts tuples with media type #201
- bugfix List hint extraction with non-basic sub types #207

Breaking changes:

- reverted back to `0.10.0` Swagger UI behavior as default. Users relying on stricter CSP should use `SpectacularSwaggerSplitView`
- `tokenAuth` slightly changed to properly model correct `Authorization` header
- a lot of minor improvements that may slightly alter the schema

1.9.9 0.11.1 (2020-11-15)

- bugfix hint extraction on @cached_property #198
- add support for basic TypedDict hints #184
- improve type hint resolution #199
- add option to disable Null/Blank enum choice feature #185
- bugfix return code for Viewset create methods #196
- honor SCHEMA_COERCE_PATH_PK on path param type resolution #194
- bugfix absolute schema URL to relative in UI #193

Breaking changes:

- return code for `create` on `ViewSet` changed from `200` to `201`. Some generator targets are picky, others don't care.

1.9.10 0.11.0 (2020-11-06)

- Remove unnecessary view permission from action [Vikas]
- Fix security definition for IsAuthenticatedOrReadOnly permission [Vikas]
- introduce convenience decorator @schema_extend_view #182
- bugfix override behaviour of extend_schema with methods and views
- move some plumbing to drainage to make importable without cirular import issues
- bugfix naming for ListSerializer with pagination #183
- cleanup trailing whitespace in docstrings
- normalize regex in pattern, remove ECMA-incompatible URL pattern #175
- remove Swagger UI inline script for stricter CSP #173
- fixed typo [Sebastian Pabst]
- add the PASSWORD format to types.py [Sebastian Pabst]
- docs(settings): fix favicon example [Max Wittig]

Breaking changes:

- `@extend_schema` override mechanics are now consistent. may affect schema only if used on both view and view method
- otherwise mainly small improvement/fixes that should have minimal impact on the schema.

1.9.11 0.10.0 (2020-10-20)

- bugfix non-effective multi-usage of view extension.
- improve resolvable enum collisions with split components
- Update README.rst [Jose Luis da Cruz Junior]
- fix regular expression in detype_pattern [Ruslan Ibragimov]
- improve enum naming with resolvable collisions
- improve handling of discouraged SECURITY setting (fixes #48 fixes #136)
- instance check with ViewSetMixin instead of GenericViewSet [SoungRyoul Kim]
- support swagger-ui-settings [SoungRyoul Kim]
- Change Settings variable, allow override of default swagger settings and remove unnecessary line [Nix]
- Fix whitspace issues in code [Nix]
- Allow Swagger-UI configuration through settings Closes #162 [Nix]
- extend django_filters test case #155
- add enum postprocessing handling of blank and null #135
- rest-auth improvements
- test_rest_auth: Add test schema transforms [John Vandenberg]
- tests: Allow transformers on expected schemas [John Vandenberg]
- Improve schema difference test harness [John Vandenberg]

- Add rest-auth tests [John Vandenberg]
- contrib: Add rest-auth support [John Vandenberg]

Breaking changes:

- enum naming collision resolution changed in cleanly resolvable situations.
- enums gained `null` and `blank` cases, which are modeled through `oneOf` for deduplication
- SECURITY setting is now additive instead of being the mostly overridden default

1.9.12 0.9.14 (2020-10-04)

- improve client generation for paginated listings
- update pinned swagger-ui version #160
- Hot fix for AcceptVersioningHeader support [Nicolas Delaby]
- bugfix module string includes with urlpatterns #157
- add expressive error in case of misconfiguration #156
- fix django-filter related resolution. improve test #150 #151
- improve follow_field_source for reverse resolution and model leafs #150
- add ref if list field child is serializer [Matt Shirley]
- add customization option for mock request generation #135

Breaking changes:

- paginated list response is now wrapped in its own component

1.9.13 0.9.13 (2020-09-13)

- bugfix filter parameter application on non-list views #147
- improved support for django-filter
- add mocked request for view processing. #81 #141
- Use sha256 to hash lists [David Davis]
- change empty operation name on API prefix-cut to "root"
- bugfix lost "missing hint" warning and incorrect empty fallback
- add operationId collision resolution #137
- bugfix leaking path var names in operationId #137
- add config for camelizing names #138
- bugfix parameterized patterns for namespace versioning #145
- Add support for Accept header versioning [Krzysztof Socha]
- support for DictField child type (#142) and models.JSONField (Django>=3.1)
- add convenience inline_serializer for extend_schema #139
- remove multipleOf due to schema violation #131

Breaking changes:

- `operationId` changed for endpoints using the DRF's `FORMAT` path feature.
- `operationId` changed where there were path variables leaking into the name.

1.9.14 0.9.12 (2020-07-22)

- Temporarily pin the swagger-ui unpkg URL to 3.30.0 [Mohamed Abdulaziz]
- Add *deepLinking* parameter [p.alekseev]
- added preprocessing hooks for operation list modification/filtering #93
- Document effective DRF settings [John Vandenberg]
- add format query parameter #110
- improve assert messages #126
- more graceful handling of magic fields #126
- allow for field child on ListSerializer. #120
- Fix sorting of endpoints with params [John Vandenberg]
- Emit enum of possible format suffixes [John Vandenberg]
- i18n #109
- bugfix INSTALLED_APP retrieval #114
- emit import warning for extensions with installed apps #114

Breaking changes:

- `drf_spectacular.hooks.postprocess_schema_enums` moved from `blumbing` to `hooks` for consistency. Only relevant if `POSTPROCESSING_HOOKS` is explicitly set by user.
- preprocessing hooks are currently experimental and may change on the next release.

1.9.15 0.9.11 (2020-07-08)

- extend instead of replace extra parameters #111
- add client generator helper settings for readOnly
- bugfix format param: path params must be required=True
- bugfix DRF docstring excludes and configuration #107
- bugfix operations with urlpattern override #92
- decrease built-in extension priority and improve doc #106
- add option to hide serializer fields #100
- allow None on @extend_schema request/response
- bugfix json spec violation on "required :[]" for COMPONENT_SPLIT_REQUEST

Breaking changes:

- `@extend_schema(parameters=...)` is extending instead of replacing for custom `AutoSchema`
- path parameter are now always `required=True` as required by specification

1.9.16 0.9.10 (2020-06-23)

- bugfix cyclic import in plumbing. #104
- add upstream test target with contrib allowed to fail
- preparations for django 3.1 and DRF 3.12
- improve tox targets for unreleased upstream

1.9.17 0.9.9 (2020-06-20)

- added explicit URL option to UI views. #103
- improve auth extension doc #99
- bugfix attr typo with Token auth extension #99
- improve docstring extraction #96
- Manual polymorphic [Jair Henrique]
- Add summary field to extend_schema #97 [lilisha100]
- reduce minimal package requirements
- extend sdist with tests & doc
- bugfix nested RO/WO serializer on COMPONENT_SPLIT_REQUEST
- add pytest option –skip-missing-contrib #87
- Save test files in temporary folder [Jair Henrique]
- Setup isort library [Jair Henrique]

1.9.18 0.9.8 (2020-06-07)

- bugfix read-only many2many relation processing #79
- Implement OrderedDict representer for yaml dumper [Jair Henrique]
- bugfix UI permissions #84
- fix abc import #82
- add duration field #78

1.9.19 0.9.7 (2020-06-05)

- put contrib code in packages named files
- improve djangorestframework-camel-case support #73
- Add support to djangorestframework-camel-case [Jair Henrique]
- ENUM_NAME_OVERRIDES accepts import string for easier handling #70
- honor versioning on schema UIs #71
- improve enum naming mechanism. #63 #70
- provide global enum naming. #70

- refactor choice field
- remove unused sorter setting
- improve FileField, add test and documentation. #69
- Fix file fields [John Vandenberg]
- allow for functions on models beside properties. #68
- replace removed DRF compat function

Breaking changes:

- Enum naming conflicts are now resolved explicitly. how to resolve conflicts
- Choice fields may be rendered slightly different
- Swagger UI and Redoc views now honor versioned requests
- Contrib package code moved. each package has its own file now

1.9.20 0.9.6 (2020-05-23)

- overhaul documentation #52
- improve serializer field mapping (nullbool & time)
- remove duplicate and misplaced description. #61
- extract serializer docstring
- Recognise ListModelMixin as a list [John Vandenberg]
- bugfix component sorting to include enums. #60
- bugfix fail on missing readOnly flag
- Fix incorrect parameter cutting [p.alekseev]

1.9.21 0.9.5 (2020-05-20)

- add optional serializer component split
- improve SerializerField meta extraction
- improve serializer directionality
- add mypy static analysis
- make all readonly fields required for output. #54
- make yaml multi-line strings nicer
- alphanumeric component sorting.
- generalize postprocessing hooks
- extension override through priority attr

Breaking changes:

- Schemas are funtionally identical, but component sorting changed slightly.
- All `read_only` fields are required by default
- `SerializerFieldExtension` gained direction parameter

1.9.22 0.9.4 (2020-05-13)

- robustify serializer resolution & enum postprocessing
- expose api_version to command. robustify version matching. #22
- add versioning support #22
- robustify urlconf wrapping. resolver does not like lists
- explicit override for non-list serializers on ViewSet list #49
- improve model field mapping via DRF init logic
- bugfix enum substitution with additional field parameters.
- Fix getting default parameter for *MultipleChoiceField* [p.alekseev]
- bugfix model path traversal via intermediate property
- try to be more graceful with unknown custom model fields. #33

Breaking changes:

- If URL or namespace versioning is set in views, it is automatically used for generation. Schemas might shrink because of that. Explicit usage of `--api-version="XXX"` should yield the old result.
- Some warnings might change, as the field/view introspection tries to go deeper.

1.9.23 0.9.3 (2020-05-07)

- Add (partial) support for drf-yasg's serializer ref_name #27
- Add thin wrappers for redoc and swagger-ui. #19
- Simplify serializer naming override #27
- Handle drf type error for yaml. #41
- Tox.ini: Add {posargs} [John Vandenberg]
- add djangorestframework-jwt auth handler [John Vandenberg]
- Docs: example of a manual configuration to use a apiKey in securitySchemes [Jelmer Draaijer]
- Introduce view override extension
- Consolidate extensions
- Parse path parameter type hints from url. closes #34
- Consolidate duplicate warnings/add error #28
- Prevent warning for DRF format suffix param
- Improve ACCEPT header handling #42

Breaking changes:

- all extension base classes moved to `drf_spectacular.extensions`

1.9.24 0.9.2 (2020-04-27)

- Fix incorrect PK access through id. #25.
- Enable attr settings on SpectacularAPIView #35.
- Bugfix @api_view annotation and tests.
- Fix exception/add support for explicit ListSerializer #29.
- Introduce custom serializer field extension mechanic. enables tackling #31
- Improve serializer estimation with educated guesses. #28.
- Bugfix import error and incorrect warning #26.
- Improve scope parsing for oauth2. #26.
- Postprocessing enums to components
- Handle decimal coersion. closes #24.
- Improvement: patched serializer variation only on request.
- Add serializer directionality.
- End the bucket brigade / cleaner interface.
- Add poly serializer warning.
- Bugfix: add serialization for default values.
- Bugfix reverse access collision from schema to view.

Breaking changes:

- internal interface changed (method & path removed)
- fewer PatchedSerializers emitted
- Enums are no longer inlined

1.9.25 0.9.1 (2020-04-09)

- Bugfix missing openapi schema spec json in package
- Add multi-method action decoration support.
- rest-polymorphic str loading prep.
- Improve list view detection.
- Bugfix: response codes must be string. closes #17.

1.9.26 0.9.0 (2020-03-29)

- Add missing related serializer fields #15.
- Bugfix properties with $ref component. closes #16.
- Bugfix polymorphic resource_type lookup. closes #14.
- Generalize plugin system.
- Support `required` parameter for body. [p.alekseev]
- Improve serializer retrieval.
- Add query serializer support #10.
- Custom serializer parsing with plugins.
- Refactor auth plugin system. support for DjangoOAuthToolkit & SimpleJWT.
- Bugfix extra components.

Breaking changes:

- removed *to_schema()* from *OpenApiParameter*. Handled in `AutoSchema` now.

1.9.27 0.8.8 (2020-03-21)

- Documentation.
- Schema serving with `SpectacularAPIView` (configureable)
- Add generator stats and `--fail-on-warn` command option.
- Schema validation with `--validation` against OpenAPI JSON specification
- Added various settings.
- Bugfix/add support for basic type responses (parity with requests)
- Bugfix required in parameters. failed schema validation.
- Add validation against OpenAPI schema specification.
- Improve parameter resolution, warnings and tests.
- Allow default parameter override. (e.g. `id`)
- Fix queryset function call. [p.g.alekseev]
- Supporting enum values in params. [p.g.alekseev]
- Allow `@extend_schema` request basic type annotation.
- Add support for typing Optional[*]
- Bugfix: handle proxy models where pk is a OnetoOne relation.
- Warn on duplicate serializer names.
- Added explicit exclude flag for operation.
- Bugfix: PrimaryKeyRelatedField(read_only=True) failing to find type.
- Change operation sorting to alphanumeric with option (#6)
- Robustify serializer field support for `@extend_schema_field`.

- Enable field serializers support. [p.g.alekseev]
- Adding custom tags support [p.g.alekseev]
- Document extend_schema.
- Allow operation hiding.
- Catch unknown model traversals. custom fields can be tricky.
- Improve model field mapping. extend field tests.
- Add deprecated method to extend_schema decorator. [p.g.alekseev]

Breaking changes:

- `@extend_schema` renamed `extra_parameters` -> `parameters`
- `ExtraParameter` renamed to `OpenApiParameter`

1.9.28 0.8.5 (2020-03-08)

- Generalize `PolymorphicResponse` into `PolymorphicProxySerializer`.
- Type dict is resolved as object.
- Simplify hint resolution.
- Allow `@extend_schema_field` for custom serializer fields.

1.9.29 0.8.4 (2020-03-06)

- `@extend_schema_field` accepts Serializers and OpenApiTypes
- Generalize query parameter.
- Bugfix serializer init.
- Fix unused get_request_serializer.
- Refactor and robustify typing system.
- Helper scripts for swagger and generator.
- Fix license.

1.9.30 0.8.3 (2020-03-02)

- Fix parameter type resolution.
- Remove empty parameters.
- Improved assert message.

1.9.31 0.8.2 (2020-03-02)

- Working release.
- Bugfix wrong call & remove yaml aliases.

1.9.32 0.8.1 (2020-03-01)

- Initial published version.

1.10 Package overview

1.10.1 drf_spectacular.utils

`class drf_spectacular.utils.`**`OpenApiExample`**`(`*name: str*, *value: Optional[Any] = None*, *external_value: str = ''*, *summary: str = ''*, *description: str = ''*, *request_only: bool = False*, *response_only: bool = False*, *media_type: str = 'application/json'*, *status_codes: Optional[List[str]] = None*`)`

Bases: *`drf_spectacular.utils.OpenApiSchemaBase`*

https://swagger.io/specification/

- **Specification**
  - **Schema**
    - ExampleObject

`class drf_spectacular.utils.`**`OpenApiParameter`**`(`*name: str*, *type: Any = <class 'str'>*, *location: str = 'query'*, *required: bool = False*, *description: str = ''*, *enum: Optional[List[Any]] = None*, *deprecated: bool = False*, *style: Optional[str] = None*, *explode: Optional[bool] = None*, *default: Optional[Any] = None*, *examples: Optional[List[drf_spectacular.utils.OpenApiExample]] = None*, *exclude: bool = False*, *response: Union[bool, List[Union[int, str]]] = False*`)`

Bases: *`drf_spectacular.utils.OpenApiSchemaBase`*

`COOKIE = 'cookie'`

`HEADER = 'header'`

`PATH = 'path'`

`QUERY = 'query'`

`class drf_spectacular.utils.`**`OpenApiResponse`**`(`*response: Optional[Any] = None*, *description: Optional[str] = None*, *examples: Optional[List[*drf_spectacular.utils.OpenApiExample*]] = None*`)`

Bases: *`drf_spectacular.utils.OpenApiSchemaBase`*

Helper class to bundle a response object (`Serializer`, `OpenApiType`, raw schema, etc) together with a response object description and/or examples. Examples can alternatively be provided via @extend_schema.

`class drf_spectacular.utils.`**`OpenApiSchemaBase`**

Bases: `object`

`class drf_spectacular.utils.`**`PolymorphicProxySerializer`**(*component_name: str*, *serializers: Union[List[Union[rest_framework.serializers.Serializer, Type[rest_framework.serializers.Serializer]]], Dict[str, Union[rest_framework.serializers.Serializer, Type[rest_framework.serializers.Serializer]]]]*, *resource_type_field_name: Optional[str]*)

Bases: `object`

This class is to be used with `@extend_schema` to signal a request/response might be polymorphic (accepts/returns data possibly from different serializers). Usage usually looks like this:

```
@extend_schema(
    request=PolymorphicProxySerializer(
        component_name='MetaPerson',
        serializers=[
            LegalPersonSerializer, NaturalPersonSerializer,
        ],
        resource_type_field_name='person_type',
    )
)
def create(self, request, *args, **kwargs):
    return Response(...)
```

Beware that this is not a real serializer and therefore is not derived from serializers.Serializer. It *cannot* be used in views as *serializer_class* or as field in a actual serializer. You likely want to handle this in the view method.

Also make sure that each sub-serializer has a field named after the value of `resource_type_field_name` (discriminator field). Generated clients will likely depend on the existence of this field. Setting `resource_type_field_name` to `None` will remove the discriminator altogether. This may be useful in certain situations, but will most likely break client generation.

It is **strongly** recommended to pass the `Serializers` as **list**, and by that let *drf-spectacular* retrieve the field and handle the mapping automatically. In special circumstances, the field may not available when drf-spectacular processes the serializer. In those cases you can explicitly state the mapping with `{'legal': LegalPersonSerializer, ...}`, but it is then your responsibility to have a valid mapping.

`drf_spectacular.utils.`**`extend_schema`**(*operation_id: Optional[str] = None*, *parameters: Optional[List[drf_spectacular.utils.OpenApiParameter]] = None*, *request: Any = <class 'rest_framework.fields.empty'>*, *responses: Any = <class 'rest_framework.fields.empty'>*, *auth: Optional[List[str]] = None*, *description: Optional[str] = None*, *summary: Optional[str] = None*, *deprecated: Optional[bool] = None*, *tags: Optional[List[str]] = None*, *exclude: bool = False*, *operation: Optional[Dict] = None*, *methods: Optional[List[str]] = None*, *versions: Optional[List[str]] = None*, *examples: Optional[List[drf_spectacular.utils.OpenApiExample]] = None*)

decorator mainly for the "view" method kind. partially or completely overrides what would be generated by drf-spectacular.

Parameters

- **`operation_id`** – replaces the auto-generated operation_id. make sure there are no naming collisions.

- **`parameters`** – list of additional or replacement parameters added to the auto-discovered fields.
- **`responses`** – replaces the discovered Serializer. Takes a variety of inputs that can be used individually or combined
  - `Serializer` class
  - `Serializer` instance (e.g. `Serializer(many=True)` for listings)
  - `dict` with status codes as keys and *Serializers* as values.
  - `dict` with tuple (status_code, media_type) as keys and *Serializers* as values.
  - basic types or instances of `OpenApiTypes`
  - *`OpenApiResponse`* for bundling any of the other choices together with either a dedicated response description and/or examples.
  - *`PolymorphicProxySerializer`* for signaling that the operation may yield data from different serializers depending on the circumstances.
- **`request`** – replaces the discovered `Serializer`. Takes a variety of inputs
  - `Serializer` class/instance
  - basic types or instances of `OpenApiTypes`
  - *`PolymorphicProxySerializer`* for signaling that the operation accepts a set of different types of objects.
  - `dict` with media_type as keys and one of the above as values.
- **`auth`** – replace discovered auth with explicit list of auth methods
- **`description`** – replaces discovered doc strings
- **`summary`** – an optional short summary of the description
- **`deprecated`** – mark operation as deprecated
- **`tags`** – override default list of tags
- **`exclude`** – set True to exclude operation from schema
- **`operation`** – manually override what auto-discovery would generate. you must provide a OpenAPI3-compliant dictionary that gets directly translated to YAML.
- **`methods`** – scope extend_schema to specific methods. matches all by default.
- **`versions`** – scope extend_schema to specific API version. matches all by default.
- **`examples`** – attach request/response examples to the operation

Returns

`drf_spectacular.utils.`**`extend_schema_field`**(*field: Union[*drf_spectacular.types.OpenApiTypes, drf_spectacular.utils.PolymorphicProxySerializer, *rest_framework.serializers.Serializer, Type[rest_framework.serializers.Serializer], Dict]*, *component_name: Optional[str] = None*)

Decorator for the "field" kind. Can be used with `SerializerMethodField` (annotate the actual method) or with custom `serializers.Field` implementations.

If your custom serializer field base class is already the desired type, decoration is not necessary. To override the discovered base class type, you can decorate your custom field class.

Always takes precedence over other mechanisms (e.g. type hints, auto-discovery).

Parameters

- **`field`** – accepts a `Serializer`, *`OpenApiTypes`* or raw `dict`
- **`component_name`** – signals that the field should be broken out as separate component

`drf_spectacular.utils.`**`extend_schema_serializer`**(*many: Optional[bool] = None*, *exclude_fields: Optional[List[str]] = None*, *examples: Optional[List[*drf_spectacular.utils.OpenApiExample*]] = None*)

Decorator for the "serializer" kind. Intended for overriding default serializer behaviour that cannot be influenced through *.extend_schema*.

Parameters

- **`many`** – override how serializer is initialized. Mainly used to coerce the list view detection heuristic to acknowledge a non-list serializer.
- **`exclude_fields`** – fields to ignore while processing the serializer. only affects the schema. fields will still be exposed through the API.
- **`examples`** – define example data to serializer.

`drf_spectacular.utils.`**`extend_schema_view`**(***kwargs*)

Convenience decorator for the "view" kind. Intended for annotating derived view methods that are are not directly present in the view (usually methods like `list` or `retrieve`). Spares you from overriding methods like `list`, only to perform a super call in the body so that you have have something to attach `@extend_schema` to.

Parameters **`kwargs`** – method names as argument names and `extend_schema()` calls as values

`drf_spectacular.utils.`**`inline_serializer`**(*name: str*, *fields: Dict[str, rest_framework.fields.Field]*, ***kwargs*) → rest_framework.serializers.Serializer

A helper function to create an inline serializer. Primary use is with *@extend_schema*, where one needs an implicit one-off serializer that is not reflected in an actual class.

Parameters

- **`name`** – name of the
- **`fields`** – dict with field names as keys and serializer fields as values
- **`kwargs`** – optional kwargs for serializer initialization

1.10.2 drf_spectacular.types

`class drf_spectacular.types.`**`OpenApiTypes`**(*value*)

Basic types known to the OpenApi specification or at least common format extension of it.

- Use BYTE for base64 encoded data wrapped in a string
- Use BINARY for raw binary data
- Use OBJECT for arbitrary free-form object (usually a dict)

`ANY = 25`

`BINARY = 7`

`BOOL = 4`

`BYTE = 6`

DATE = 19

DATETIME = 18

DECIMAL = 17

DOUBLE = 3

DURATION = 21

EMAIL = 22

FLOAT = 2

HOSTNAME = 16

INT = 9

INT32 = 10

INT64 = 11

IP4 = 14

IP6 = 15

NONE = 24

NUMBER = 1

OBJECT = 23

PASSWORD = 8

STR = 5

TIME = 20

URI = 13

UUID = 12

1.10.3 drf_spectacular.views

class `drf_spectacular.views.`**SpectacularAPIView**(***kwargs*)

Bases: `rest_framework.views.APIView`

OpenApi3 schema for this API. Format can be selected via content negotiation.

- YAML: application/vnd.oai.openapi
- JSON: application/vnd.oai.openapi+json

api_version = None

authentication_classes = [<class 'rest_framework.authentication.SessionAuthentication'>, <class 'rest_framework.authentication.BasicAuthentication'>]

generator_class

alias of `drf_spectacular.generators.SchemaGenerator`

get(*request*, **args*, ***kwargs*)

permission_classes = [<class 'rest_framework.permissions.AllowAny'>]

renderer_classes = [<class 'drf_spectacular.renderers.OpenApiYamlRenderer'>, <class 'drf_spectacular.renderers.OpenApiYamlRenderer2'>, <class 'drf_spectacular.renderers.OpenApiJsonRenderer'>, <class 'drf_spectacular.renderers.OpenApiJsonRenderer2'>]

serve_public = True

urlconf = None

class drf_spectacular.views.**SpectacularJSONAPIView**(***kwargs*)
Bases: *drf_spectacular.views.SpectacularAPIView*

renderer_classes = [<class 'drf_spectacular.renderers.OpenApiJsonRenderer'>, <class 'drf_spectacular.renderers.OpenApiJsonRenderer2'>]

class drf_spectacular.views.**SpectacularRedocView**(***kwargs*)
Bases: rest_framework.views.APIView

authentication_classes = [<class 'rest_framework.authentication.SessionAuthentication'>, <class 'rest_framework.authentication.BasicAuthentication'>]

get(*request*, **args*, ***kwargs*)

permission_classes = [<class 'rest_framework.permissions.AllowAny'>]

renderer_classes = [<class 'rest_framework.renderers.TemplateHTMLRenderer'>]

template_name = 'drf_spectacular/redoc.html'

url = None

url_name = 'schema'

class drf_spectacular.views.**SpectacularSwaggerSplitView**(***kwargs*)
Bases: *drf_spectacular.views.SpectacularSwaggerView*

Alternate Swagger UI implementation that separates the html request from the javascript request to cater to web servers with stricter CSP policies.

get(*request*, **args*, ***kwargs*)

url_self = None

class drf_spectacular.views.**SpectacularSwaggerView**(***kwargs*)
Bases: rest_framework.views.APIView

authentication_classes = [<class 'rest_framework.authentication.SessionAuthentication'>, <class 'rest_framework.authentication.BasicAuthentication'>]

get(*request*, **args*, ***kwargs*)

permission_classes = [<class 'rest_framework.permissions.AllowAny'>]

renderer_classes = [<class 'rest_framework.renderers.TemplateHTMLRenderer'>]

template_name = 'drf_spectacular/swagger_ui.html'

template_name_js = 'drf_spectacular/swagger_ui.js'

url = None

url_name = 'schema'

class drf_spectacular.views.**SpectacularYAMLAPIView**(***kwargs*)
Bases: *drf_spectacular.views.SpectacularAPIView*

renderer_classes = [<class 'drf_spectacular.renderers.OpenApiYamlRenderer'>, <class 'drf_spectacular.renderers.OpenApiYamlRenderer2'>]

1.10.4 drf_spectacular.extensions

class drf_spectacular.extensions.**OpenApiAuthenticationExtension**(*target*)
Bases: drf_spectacular.plumbing.OpenApiGeneratorExtension[*OpenApiAuthenticationExtension*]

abstract get_security_definition(*auto_schema*)

get_security_requirement(*auto_schema*)

name: str

class drf_spectacular.extensions.**OpenApiFilterExtension**(*target*)
Bases: drf_spectacular.plumbing.OpenApiGeneratorExtension[*OpenApiFilterExtension*]

abstract get_schema_operation_parameters(*auto_schema*, **args*, ***kwargs*)

class drf_spectacular.extensions.**OpenApiSerializerExtension**(*target*)
Bases: drf_spectacular.plumbing.OpenApiGeneratorExtension[*OpenApiSerializerExtension*]

get_name() → Optional[str]
return str for overriding default name extraction

map_serializer(*auto_schema*, *direction*)
override for customized serializer mapping

class drf_spectacular.extensions.**OpenApiSerializerFieldExtension**(*target*)
Bases: drf_spectacular.plumbing.OpenApiGeneratorExtension[*OpenApiSerializerFieldExtension*]

get_name() → Optional[str]
return str for breaking out field schema into separate component

abstract map_serializer_field(*auto_schema*, *direction*)

class drf_spectacular.extensions.**OpenApiViewExtension**(*target*)
Bases: drf_spectacular.plumbing.OpenApiGeneratorExtension[*OpenApiViewExtension*]

abstract view_replacement()

1.10.5 drf_spectacular.hooks

drf_spectacular.hooks.**postprocess_schema_enums**(*result*, *generator*, ***kwargs*)
simple replacement of Enum/Choices that globally share the same name and have the same choices. Aids client generation to not generate a separate enum for every occurrence. only takes effect when replacement is guaranteed to be correct.

drf_spectacular.hooks.**preprocess_exclude_path_format**(*endpoints*, ***kwargs*)
preprocessing hook that filters out {format} suffixed paths, in case format_suffix_patterns is used and {format} path params are unwanted.

1.10.6 drf_spectacular.openapi

class drf_spectacular.openapi.**AutoSchema**

Bases: `rest_framework.schemas.inspectors.ViewInspector`

get_auth()

Obtains authentication classes and permissions from view. If authentication is known, resolve security requirement for endpoint and security definition for the component section. For custom authentication subclass `OpenApiAuthenticationExtension`.

get_description()

override this for custom behaviour

get_examples()

get_operation(*path*, *path_regex*, *path_prefix*, *method*, *registry: drf_spectacular.plumbing.ComponentRegistry*)

get_operation_id()

override this for custom behaviour

get_override_parameters()

override this for custom behaviour

get_request_serializer()

override this for custom behaviour

get_response_serializers()

override this for custom behaviour

get_summary()

override this for custom behaviour

get_tags() → List[str]

override this for custom behaviour

is_deprecated()

override this for custom behaviour

map_parsers()

map_renderers(*attribute*)

method_mapping = {'delete': 'destroy', 'get': 'retrieve', 'patch': 'partial_update', 'post': 'create', 'put': 'update'}

resolve_serializer(*serializer*, *direction*) → drf_spectacular.plumbing.ResolvedComponent

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX