# drf-spectacular

T. Franzel

Apr 15, 2023

# 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 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)
- `SerializerMethodField()` type via type hinting or `@extend_schema_field`
- i18n support
- Tags extraction
- Request/response/parameter examples
- Description extraction from `docstrings`
- Vendor specification extensions (`x-*`) in info, operations, parameters, components, and security schemes
- 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
- Callback operations
- **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
  - drf-nested-routers
  - djangorestframework-recursive
  - djangorestframework-dataclasses
  - django-rest-framework-gis

For more information visit the documentation.

### 1.1.1 License

Provided by T. Franzel. Licensed under 3-Clause BSD.

### 1.1.2 Requirements

- Python >= 3.6
- Django (2.2, 3.2, 4.0, 4.1, 4.2)
- Django REST Framework (3.10.3, 3.11, 3.12, 3.13, 3.14)

### 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 recommend to specify at least some metadata.

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

### Self-contained UI installation

Certain environments have no direct access to the internet and as such are unable to retrieve Swagger UI or Redoc from CDNs. drf-spectacular-sidecar provides these static files as a separate optional package. Usage is as follows:

```
$ pip install drf-spectacular[sidecar]
```

```
INSTALLED_APPS = [
    # ALL YOUR APPS
    'drf_spectacular',
    'drf_spectacular_sidecar',  # required for Django collectstatic discovery
]
SPECTACULAR_SETTINGS = {
    'SWAGGER_UI_DIST': 'SIDECAR',  # shorthand to use the sidecar instead
    'SWAGGER_UI_FAVICON_HREF': 'SIDECAR',
    'REDOC_DIST': 'SIDECAR',
    # 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 --color --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
```

(continues on next page)

(continued from previous page)

```
        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
        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},
        methods=["POST"]
    )
    @extend_schema(description='Override a specific method', methods=["GET"])
    @action(detail=True, methods=['post', 'get'])
    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,
    # Insert a manual path prefix to the operation path, e.g. '/service/backend'.
    # Use this for example to align paths when the API is mounted as a sub-resource
    # behind a proxy and Django is not aware of that. Alternatively, prefixes can
    # also specified via SERVERS, but this makes the operation path more explicit.
    'SCHEMA_PATH_PREFIX_INSERT': '',
    # Coercion of {pk} to {id} is controlled by SCHEMA_COERCE_PATH_PK. Additionally,
    # some libraries (e.g. drf-nested-routers) use "_pk" suffixed path variables.
    # This setting globally coerces path variables like "{user_pk}" to "{user_id}".
    'SCHEMA_COERCE_PATH_PK_SUFFIX': False,

    # 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.
    'DEFAULT_GENERATOR_CLASS': 'drf_spectacular.generators.SchemaGenerator',
    # Create separate components for PATCH endpoints (without required list)
    'COMPONENT_SPLIT_PATCH': True,
```

(continues on next page)

(continued from previous page)

```
    # Split components into request and response parts where appropriate
    # This setting is highly recommended to achieve the most accurate API
    # description, however it comes at the cost of having more components.
    'COMPONENT_SPLIT_REQUEST': False,
    # Aid client generator targets that have trouble with read-only properties.
    'COMPONENT_NO_READ_ONLY_REQUIRED': False,

    # Adds "minLength: 1" to fields that do not allow blank strings. Deactivated
    # by default because serializers do not strictly enforce this on responses and
    # so "minLength: 1" may not always accurately describe API behavior.
    # Gets implicitly enabled by COMPONENT_SPLIT_REQUEST, because this can be
    # accurately modeled when request and response components are separated.
    'ENFORCE_NON_BLANK_FIELDS': 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 endpoint 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 general configuration to pass to the SwaggerUI({ ... })
    # https://swagger.io/docs/open-source-tools/swagger-ui/usage/configuration/
    # The settings are serialized with json.dumps(). If you need customized JS, use a
    # string instead. The string must then contain valid JS and is passed unchanged.
    'SWAGGER_UI_SETTINGS': {
        'deepLinking': True,
    },
    # Initialize SwaggerUI with additional OAuth2 configuration.
    # https://swagger.io/docs/open-source-tools/swagger-ui/usage/oauth2/
    'SWAGGER_UI_OAUTH2_CONFIG': {},

    # Dictionary of general configuration to pass to the Redoc.init({ ... })
    # https://github.com/Redocly/redoc#redoc-options-object
    # The settings are serialized with json.dumps(). If you need customized JS, use a
    # string instead. The string must then contain valid JS and is passed unchanged.
    'REDOC_UI_SETTINGS': {},

    # CDNs for swagger and redoc. You can change the version or even host your
    # own depending on your requirements. For self-hosting, have a look at
    # the sidecar option in the README.
    'SWAGGER_UI_DIST': 'https://cdn.jsdelivr.net/npm/swagger-ui-dist@latest',
    'SWAGGER_UI_FAVICON_HREF': 'https://cdn.jsdelivr.net/npm/swagger-ui-dist@latest/
↪favicon-32x32.png',
    'REDOC_DIST': 'https://cdn.jsdelivr.net/npm/redoc@latest',

    # 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': [],

    # Determines how operations should be sorted. If you intend to do sorting with a
    # PREPROCESSING_HOOKS, be sure to disable this setting. If configured, the sorting
    # is applied after the PREPROCESSING_HOOKS. Accepts either
    # True (drf-spectacular's alpha-sorter), False, or a callable for sort's key arg.
    'SORT_OPERATIONS': True,

    # enum name overrides. dict with keys "YourEnum" and their choice values "field.
↪choices"
    # e.g. {'SomeEnum': ['A', 'B'], 'OtherEnum': 'import.path.to.choices'}
    'ENUM_NAME_OVERRIDES': {},
    # Adds "blank" and "null" enum choices where appropriate. disable on client␣
↪generation issues
    'ENUM_ADD_EXPLICIT_BLANK_NULL_CHOICE': True,
    # Add/Append a list of (``choice value`` - choice name) to the enum description␣
↪string.
    'ENUM_GENERATE_CHOICE_DESCRIPTION': 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
    # Camelization of the operation schema itself requires the addition of
    # 'drf_spectacular.contrib.djangorestframework_camel_case.camelize_serializer_fields'
    # to POSTPROCESSING_HOOKS. Please note that the hook depends on
    # ``djangorestframework_camel_case``, while CAMELIZE_NAMES itself does not.
```

(continues on next page)

(continued from previous page)

```
    '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',

    # Path converter schema overrides (e.g. <int:foo>). Can be used to either modify␣
↪default
    # behavior or provide a schema for custom converters registered with register_
↪converter(...).
    # Takes converter labels as keys and either basic python types, OpenApiType, or raw␣
↪schemas
    # as values. Example: {'aint': OpenApiTypes.INT, 'bint': str, 'cint': {'type': ...}}
    'PATH_CONVERTER_OVERRIDES': {},

    # 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,

    # @extend_schema allows to specify status codes besides 200. This functionality is␣
↪usually used
    # to describe error responses, which rarely make use of list mechanics. Therefore,␣
↪we suppress
    # listing (pagination and filtering) on non-2XX status codes by default. Toggle this␣
↪to enable
    # list responses with ListSerializers/many=True irrespective of the status code.
    'ENABLE_LIST_MECHANICS_ON_NON_2XX': False,

    # This setting allows you to deviate from the default manager by accessing a␣
↪different model
    # property. We use "objects" by default for compatibility reasons. Using "_default_
↪manager"
    # will likely fix most issues, though you are free to choose any name.
    "DEFAULT_QUERY_MANAGER": 'objects',

    # Controls which authentication methods are exposed in the schema. If not None, will␣
↪hide
    # authentication classes that are not contained in the whitelist. Use full import␣
↪paths
    # like ['rest_framework.authentication.TokenAuthentication', ...].
    # Empty list ([]) will hide all authentication methods. The default None will show␣
↪all.
    'AUTHENTICATION_WHITELIST': None,
    # Controls which parsers are exposed in the schema. Works analog to AUTHENTICATION_
↪WHITELIST.
    # List of allowed parsers or None to allow all.
    'PARSER_WHITELIST': None,
```

(continues on next page)

(continued from previous page)

```
    # Controls which renderers are exposed in the schema. Works analog to AUTHENTICATION_
↪WHITELIST.
    # rest_framework.renderers.BrowsableAPIRenderer is ignored by default if whitelist␣
↪is None
    'RENDERER_WHITELIST': None,

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

    # Runs exemplary schema generation and emits warnings as part of "./manage.py check -
↪-deploy"
    'ENABLE_DJANGO_DEPLOY_CHECK': True,

    # General schema metadata. Refer to spec for valid inputs
    # https://spec.openapis.org/oas/v3.0.3#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"
    # e.g. [{'url': 'https://example.com/v1', 'description': 'Text'}, ...]
    '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': {},

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

    # Oauth2 related settings. used for example by django-oauth2-toolkit.
    # https://spec.openapis.org/oas/v3.0.3#oauthFlowsObject
    '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. CSS, JS functions), you can extend or 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 are converted to parameters
          OpenApiParameter("nested", QuerySerializer),  # serializer object is converted␣
↪to a 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: YourResponseSerializer, ...}` 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')
)
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',
```

(continues on next page)

(continued from previous page)

```
            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
        ),
    ]
)
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.

---

**Important:** The extensions register themselves automatically. Just be sure that the Python interpreter sees them at least once.

It is good practice to collect your extensions in `YOUR_MAIN_APP_NAME/schema.py` and importing that file in your `YOUR_MAIN_APP_NAME/apps.py`. Every proper Django app will already have an auto-generated `apps.py` file. Although not strictly necessary, doing the import in `ready()` is the most robust approach. It will make sure your environment (e.g. settings) is properly set up prior to loading.

```
# your_main_app_name/apps.py
class YourMainAppNameConfig(AppConfig):
    default_auto_field = "django.db.models.BigAutoField"
    name = "your_main_app_name"

    def ready(self):
        import your_main_app_name.schema  # noqa: E402
```

---

**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

    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.

In case your `Serializer` makes use of a custom `ListSerializer` (i.e. a custom `to_representation()`), you can write a dedicated extensions for that. This is usually the case when `many=True` does not result in a plain list, but rather in augmented object with additional fields (e.g. envelopes).

#### Declare custom/library filters with `OpenApiFilterExtension`

This extension only applies to filter and pagination classes and is rarely used. Built-in support for *django-filter* 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 hooks with the `POSTPROCESSING_HOOKS` setting.

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

**Note:** Please note that setting `POSTPROCESSING_HOOKS` will override the default. If you intend to keep the `Enum` hook, be sure to add `'drf_spectacular.hooks.postprocess_schema_enums'` back into the list.

### 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.

```
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 an issue.

### 1.5.2 My Swagger UI and/or Redoc page is blank

Chances are high that you are using django-csp. Take a look inside your browser console and confirm that you have `Content Security Policy` errors. By default, `django-csp` usually breaks our UIs for 2 reasons: external assets and inline scripts.

Using the sidecar will mitigate the remote asset loading violation by serving the asset from your `self`. Alternatively, you can also adapt `CSP_DEFAULT_SRC` to allow for those CDN assets instead.

### Solution for Swagger UI:

```
# Option: SIDECAR
SPECTACULAR_SETTINGS = {
     ...
    'SWAGGER_UI_DIST': 'SIDECAR',
    'SWAGGER_UI_FAVICON_HREF': 'SIDECAR',
}
CSP_DEFAULT_SRC = ("'self'", "'unsafe-inline'")
CSP_IMG_SRC = ("'self'", "data:")

# Option: CDN
CSP_DEFAULT_SRC = ("'self'", "'unsafe-inline'", "cdn.jsdelivr.net")
CSP_IMG_SRC = ("'self'", "data:", "cdn.jsdelivr.net")
```

**Note:** Depending on how paranoid you are, you may avoid having to use `unsafe-inline` by using `SpectacularSwaggerSplitView` instead, which does a separate request for the script. Note however that some URL rewriting deployments will break it. Use this option only if you really need to.

### Solution for Redoc:

```
# Option: SIDECAR
SPECTACULAR_SETTINGS = {
     ...
    'REDOC_DIST': 'SIDECAR',
}
# Option: CDN
CSP_DEFAULT_SRC = ("'self'", "cdn.jsdelivr.net")

# required for both CDN and SIDECAR
CSP_WORKER_SRC = ("'self'", "blob:")
CSP_IMG_SRC = ("'self'", "data:", "cdn.redoc.ly")
CSP_STYLE_SRC = ("'self'", "'unsafe-inline'", "fonts.googleapis.com")
CSP_FONT_SRC = ("'self'", "fonts.gstatic.com")
```

### 1.5.3 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*

### 1.5.4 I get an empty schema or endpoints are missing

This is usually due to versioning (or more rarely due to permissions).

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.

For the schema views you can either set a versioning class (implicit versioning via the request) or explicitly override the version with `SpectacularAPIView.as_view(api_version='YOUR_VERSION')`.

### 1.5.5 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 decided and state explicitly what is to be expected. Work through the steps in *Workflow & schema customization* to adapt your schema.

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

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

### 1.5.7 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.

### 1.5.8 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, which attempts to find a name for a set of enum choices.

The naming mechanism uses the name of the field and possibly the name of the component, followed by a suffix if necessary if there are clashes (if there are two enum fields with the same name but different set of choices). This will handle all encountered issues automatically, but also notify you of potential problems, of two kinds:

- multiple names being produced for the same set of values, due to different field names (e.g. if you have a single currency enum used by distinct fields named like `payment_currency` and `preferred_currency`, the naming mechanism will by default treat this as two different enums but emit a warning).

- clashes that result in a suffix being needed, as above.

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. The key is a string that you choose as a name to give to this set of values.

For example:

```
SPECTACULAR_SETTINGS = {
    ...
    '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',
    }
}
```

If you have multiple semantically distinct enums that happen to have the same set of values, and you want different names for them, this mechanism won't work.

### 1.5.9 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
```

### 1.5.10 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*

### 1.5.11 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.

```
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
```

### 1.5.12 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.

### 1.5.13 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(responses=XSerializer(many=True))`.

### 1.5.14 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.

### 1.5.15 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`.

### 1.5.16 My `@action` is erroneously paginated or has filter parameters that I do not want

This usually happens when `@extend_schema(responses=XSerializer(many=True))` is used. Actions inherit filter and pagination classes from their `ViewSet`. If the response is then marked as a list, the `pagination_class` kicks in. Since actions are handled manually by the user, this behavior is usually not immediately obvious. To make your intentions clear to *drf-spectacular*, you need to clear the offending classes in the action decorator, e.g. setting `pagination_class=None`.

Users of *django-filter* might also see unwanted query parameters. Since the same mechanics apply here too, you can remove those parameters by resetting the filter backends with `@action(...,filter_backends=[])`.

```
class XViewset(viewsets.ModelViewSet):
    queryset = SimpleModel.objects.all()
    pagination_class = pagination.LimitOffsetPagination

    @extend_schema(responses=SimpleSerializer(many=True))
    @action(methods=['GET'], detail=False, pagination_class=None)
    def custom_action(self):
        pass
```

### 1.5.17 How do I wrap my responses? / My endpoints are wrapped in a generic envelope

This non-native behavior can be conveniently modeled with a simple helper function. You simply need to wrap the actual serializer with your envelope serializer and provide it to `@extend_schema`.

Here is an example on how to build an `enveloper` helper function. In this example, the actual serializer is put into the `data` field, while `status` is some arbitrary envelope field. Adapt to your specific requirements.

```
def enveloper(serializer_class, many):
    component_name = 'Enveloped{}{}'.format(
        serializer_class.__name__.replace("Serializer", ""),
        "List" if many else "",
    )

    @extend_schema_serializer(many=False, component_name=component_name)
    class EnvelopeSerializer(serializers.Serializer):
        status = serializers.BooleanField()  # some arbitrary envelope field
        data = serializer_class(many=many)  # the enveloping part

    return EnvelopeSerializer


class XViewset(GenericViewSet):
    @extend_schema(responses=enveloper(XSerializer, True))
```

(continues on next page)

(continued from previous page)

```
    def list(self, request, *args, **kwargs):
        ...
```

### 1.5.18 How can I have multiple SpectacularAPIView with differing settings

First, define your base settings in `settings.py` with `SPECTACULAR_SETTINGS`. Then, if you need another schema with different settings, you can provide scoped overrides by providing a `custom_settings` argument. `custom_settings` expects a `dict` and only allows keys that represent valid setting names.

Beware that using this mechanic is not thread-safe at the moment.

Also note that overriding `SERVE_*` or `DEFAULT_GENERATOR_CLASS` in `custom_settings` is not allowed. `SpectacularAPIView` has dedicated arguments for overriding these settings.

```
urlpatterns = [
    path('api/schema/', SpectacularAPIView.as_view(),
    path('api/schema-custom/', SpectacularAPIView.as_view(
        custom_settings={
            'TITLE': 'your custom title',
            'SCHEMA_PATH_PREFIX': 'your custom regex',
            ...
        }
    ), name='schema-custom'),
]
```

### 1.5.19 How to correctly annotate function-based views that use @api_view()

DRF provides a convenient way to write function-based views. `@api_view()` in essence wraps a regular function and implicitly converts it to a `APIView` class. For single-method cases, simply use *@extend_schema* just as you would with a normal view method.

```
@extend_schema(request=XSerializer, responses=XSerializer)
@api_view(['POST'])
def view_func(request, format=None):
    return ...
```

For functions that provide multiple methods, its advisable to use *@extend_schema_view* and break down each case separately.

```
@extend_schema_view(
    get=extend_schema(description='get desc', responses=XSerializer),
    post=extend_schema(description='post desc', request=None, responses=OpenApiTypes.
↪UUID),
)
@api_view(['GET', 'POST'])
def view_func(request, format=None):
    return ...
```

### 1.5.20 My `get_queryset()` depends on some attributes not available at schema generation time

In certain situations we need to call `get_serializer`, which in turn calls `get_queryset`. If your `get_queryset` (or `get_serializer_class`) depends on attributes not available at schema generation time (e.g. `request.user.is_authenticated`), you need to provide a fallback that allows us to call that method. While the schema is generated, you can check for the view attribute `swagger_fake_view` and simply return an empty queryset of the correct model.

```
class XViewset(viewsets.ModelViewset):
    ...

    def get_queryset(self):
        if getattr(self, 'swagger_fake_view', False):  # drf-yasg comp
            return YourModel.objects.none()
        # your usual logic
```

### 1.5.21 How to serve in-memory generated files or files in general outside `FileField`

DRF provides a convenient `FileField` for storing files persistently within a `Model`. `drf-spectacular` handles these correctly by default. But to serve binary files that are *generated in-memory*, follow the following recipe. This example uses the method recommended by Django for treating a `Response` as a file and sets up an appropriate `Renderer` that will handle the client `Accept` header for this response content type. `responses=bytes` expresses that the response is a binary blob without further details on its structure.

```
from django.http import HttpResponse
from rest_framework.renderers import BaseRenderer


class BinaryRenderer(BaseRenderer):
    media_type = "application/octet-stream"
    format = "bin"


class FileViewSet(RetrieveModelMixin, GenericViewSet):
    ...
    renderer_classes = [BinaryRenderer]

    @extend_schema(responses=bytes)
    def retrieve(self, request, *args, **kwargs):
        export_data = b"..."
        return HttpResponse(
            export_data,
            content_type=BinaryRenderer.media_type,
            headers={
                "Content-Disposition": "attachment; filename=out.bin",
            },
        )
```

## 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`. 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

            @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

```python
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


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
```

(continues on next page)

(continued from previous page)

```
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'

    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': [
```

(continues on next page)

(continued from previous page)

```
                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.

---

```
SPECTACULAR_SETTINGS = {
    "APPEND_COMPONENTS": {
        "securitySchemes": {
            "ApiKeyAuth": {
                "type": "apiKey",
                "in": "header",
                "name": "Authorization"
            }
        }
    },
    "SECURITY": [{"ApiKeyAuth": [], }],
     ...
}
```

### 1.6.4 Polymorphic models

Using polymorphic models/serializers unfortunately yields flat serializers due to the way the serializers are constructed. This means the polymorphic serializers have no inheritance hierarchy that represents common functionality. These extensions retroactively build a hierarchy by rolling up the "common denominator" fields into the base components, and importing those into the sub-components via `allOf`. This results in components that better represent the structure of the underlying serializers/models from which they originated.

The components work perfectly fine without this extension, but in some cases generated client code has a hard time with the disjunctive nature of the unmodified components. This blueprint is designed to fix that issue.

```python
from drf_spectacular.contrib.rest_polymorphic import PolymorphicSerializerExtension
from drf_spectacular.plumbing import ResolvedComponent
from drf_spectacular.serializers import PolymorphicProxySerializerExtension
from drf_spectacular.settings import spectacular_settings


class RollupMixin:
    """
    This is a schema helper that pulls the "common denominator" fields from child
    components into their parent component. It only applies to PolymorphicSerializer
    as well as PolymorphicProxySerializer, where there is an (implicit) inheritance␣
↪hierarchy.

    The actual functionality is realized via extensions defined below.
    """
    def map_serializer(self, auto_schema, direction):
        schema = super().map_serializer(auto_schema, direction)

        if isinstance(self, PolymorphicProxySerializerExtension):
            sub_serializers = self.target.serializers
        else:
            sub_serializers = [
                self.target._get_serializer_from_model_or_instance(sub_model)
                for sub_model in self.target.model_serializer_mapping
            ]

        resolved_sub_serializers = [
            auto_schema.resolve_serializer(sub, direction) for sub in sub_serializers
        ]
        # this will only be generated on return of map_serializer so mock it for now
        mocked_component = ResolvedComponent(
            name=auto_schema._get_serializer_name(self.target, direction),
            type=ResolvedComponent.SCHEMA,
            object=self.target,
            schema=schema
        )

        # hack for recursive models. at the time of extension execution, not all sub
        # serializer schema have been generated, so no rollup is possible.
        # by registering a local variable scoped postproc hook, we delay this
        # execution to the end where all schemas are present.
        def postprocessing_rollup_hook(generator, result, **kwargs):
            rollup_properties(mocked_component, resolved_sub_serializers)
```

(continues on next page)

(continued from previous page)

```
            result['components'] = generator.registry.build({})
            return result

        # register postproc hook. must run before enum postproc due to rebuilding the␣
↪registry
        spectacular_settings.POSTPROCESSING_HOOKS.insert(0, postprocessing_rollup_hook)
        # and do nothing for now
        return schema


def rollup_properties(component, resolved_sub_serializers):
    # rollup already happened (spectacular bug and normally not needed)
    if any('allOf' in r.schema for r in resolved_sub_serializers):
        return

    all_field_sets = [
        set(list(r.schema['properties'])) for r in resolved_sub_serializers
    ]
    common_fields = all_field_sets[0].intersection(*all_field_sets[1:])
    common_schema = {
        'properties': {},
        'required': set(),
    }

    # substitute sub serializers' common fields with base class
    for r in resolved_sub_serializers:
        for cf in sorted(common_fields):
            if cf in r.schema['properties']:
                common_schema['properties'][cf] = r.schema['properties'][cf]
                del r.schema['properties'][cf]
                if cf in r.schema.get('required', []):
                    common_schema['required'].add(cf)
        r.schema = {'allOf': [component.ref, r.schema]}

    # modify regular schema for field rollup
    del component.schema['oneOf']
    component.schema['properties'] = common_schema['properties']
    if common_schema['required']:
        component.schema['required'] = sorted(common_schema['required'])


class PolymorphicRollupSerializerExtension(RollupMixin, PolymorphicSerializerExtension):
    priority = 1


class PolymorphicProxyRollupSerializerExtension(RollupMixin,␣
↪PolymorphicProxySerializerExtension):
    priority = 1
```

### 1.6.5 RapiDoc

RapiDoc is documentation tool that can be used as an alternate to Redoc or Swagger UI.

```
from rest_framework.renderers import TemplateHTMLRenderer
from rest_framework.response import Response
from rest_framework.reverse import reverse
from rest_framework.views import APIView

from drf_spectacular.plumbing import get_relative_url, set_query_parameters
from drf_spectacular.settings import spectacular_settings
from drf_spectacular.utils import extend_schema
from drf_spectacular.views import AUTHENTICATION_CLASSES


class SpectacularRapiDocView(APIView):
    renderer_classes = [TemplateHTMLRenderer]
    permission_classes = spectacular_settings.SERVE_PERMISSIONS
    authentication_classes = AUTHENTICATION_CLASSES
    url_name = 'schema'
    url = None
    template_name = 'rapidoc.html'
    title = spectacular_settings.TITLE

    @extend_schema(exclude=True)
    def get(self, request, *args, **kwargs):
        schema_url = self.url or get_relative_url(reverse(self.url_name,␣
↪request=request))
        schema_url = set_query_parameters(schema_url, lang=request.GET.get('lang'))
        return Response(
            data={
                'title': self.title,
                'dist': 'https://cdn.jsdelivr.net/npm/rapidoc@latest',
                'schema_url': schema_url,
            },
            template_name=self.template_name,
        )
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ title|default:"RapiDoc" }}</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script type="module" src="{{ dist }}/dist/rapidoc-min.js"></script>
  </head>
  <body>
    <rapi-doc spec-url="{{ schema_url }}"></rapi-doc>
  </body>
</html>
```

### 1.6.6 Elements

Elements is another documentation tool that can be used as an alternate to Redoc or Swagger UI.

```python
from rest_framework.renderers import TemplateHTMLRenderer
from rest_framework.response import Response
from rest_framework.reverse import reverse
from rest_framework.views import APIView

from drf_spectacular.plumbing import get_relative_url, set_query_parameters
from drf_spectacular.settings import spectacular_settings
from drf_spectacular.utils import extend_schema
from drf_spectacular.views import AUTHENTICATION_CLASSES


class SpectacularElementsView(APIView):
    renderer_classes = [TemplateHTMLRenderer]
    permission_classes = spectacular_settings.SERVE_PERMISSIONS
    authentication_classes = AUTHENTICATION_CLASSES
    url_name = 'schema'
    url = None
    template_name = 'elements.html'
    title = spectacular_settings.TITLE

    @extend_schema(exclude=True)
    def get(self, request, *args, **kwargs):
        schema_url = self.url or get_relative_url(reverse(self.url_name,␣
↪request=request))
        schema_url = set_query_parameters(schema_url, lang=request.GET.get('lang'),␣
↪version=request.GET.get('version'))
        return Response(
            data={
                'title': self.title,
                'js_dist': 'https://unpkg.com/@stoplight/elements/web-components.min.js',
                'css_dist': 'https://unpkg.com/@stoplight/elements/styles.min.css',
                'schema_url': self._get_schema_url(request),
            },
            template_name=self.template_name
        )
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{ title|default:"Elements" }}</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no
↪">
    <script src="{{ js_dist }}"></script>
    <link rel="stylesheet" href="{{ css_dist }}">
  </head>
  <body>
    <elements-api apiDescriptionUrl="{{ schema_url }}" router="hash" />
  </body>
```

(continues on next page)

(continued from previous page)

```
</html>
```

### 1.6.7 drf-rw-serializers

drf-rw-serializers provides generic views, viewsets and mixins that extend the Django REST Framework ones adding separated serializers for read and write operations.

*drf-spectacular* requires just a small `AutoSchema` augmentation to make it aware of `drf-rw-serializers`. Remember to replace the `AutoSchema` in `DEFAULT_SCHEMA_CLASS`.

```
from drf_rw_serializers.generics import GenericAPIView as RWGenericAPIView

from drf_spectacular.openapi import AutoSchema


class CustomAutoSchema(AutoSchema):
    """ Utilize custom drf_rw_serializers methods for directional serializers """

    def get_request_serializer(self):
        if isinstance(self.view, RWGenericAPIView):
            return self.view.get_write_serializer()
        return self._get_serializer()

    def get_response_serializers(self):
        if isinstance(self.view, RWGenericAPIView):
            return self.view.get_read_serializer()
        return self._get_serializer()
```

### 1.6.8 drf-extra-fields Base64FileField

drf-extra-fields provides a `Base64FileField` and `Base64ImageField` that automatically represent binary files as base64 encoded strings. This is a useful way to embed files within a larger JSON API and keep all data within the same tree and served with a single request or response.

Because requests to these fields require a base64 encoded string and responses can be either a URI or base64 contents (if `represent_as_base64=True`) custom schema generation logic is required as this differs from the default DRF `FileField`.

```
from drf_spectacular.extensions import OpenApiSerializerFieldExtension
from drf_spectacular.openapi import AutoSchema
from drf_spectacular.plumbing import append_meta
from drf_spectacular.plumbing import build_basic_type
from drf_spectacular.types import OpenApiTypes
from drf_spectacular.utils import Direction


class Base64FileFieldSchema(OpenApiSerializerFieldExtension):
    target_class = "drf_extra_fields.fields.Base64FileField"

    def map_serializer_field(self, auto_schema, direction):
        if direction == "request":
```

(continues on next page)

(continued from previous page)

```
            return build_basic_type(OpenApiTypes.BYTE)
        elif direction == "response":
            if self.target.represent_in_base64:
                return build_basic_type(OpenApiTypes.BYTE)
            else:
                return build_basic_type(OpenApiTypes.URI)


class Base64ImageFieldSchema(Base64FileFieldSchema):
    target_class = "drf_extra_fields.fields.Base64ImageField"


class PresentablePrimaryKeyRelatedFieldSchema(OpenApiSerializerFieldExtension):
    target_class = 'drf_extra_fields.relations.PresentablePrimaryKeyRelatedField'

    def map_serializer_field(self, auto_schema: AutoSchema, direction: Direction):
        if direction == 'request':
            return build_basic_type(OpenApiTypes.INT)

        meta = auto_schema._get_serializer_field_meta(self.target, direction)
        schema = auto_schema.resolve_serializer(
            self.target.presentation_serializer(
                context=self.target.context, **self.target.presentation_serializer_
↪kwargs,
            ),
            direction,
        ).ref
        return append_meta(schema, meta)
```

### 1.6.9 django-auth-adfs

django-auth-adfs provides "a Django authentication backend for Microsoft ADFS and Azure AD". The blueprint works for the Azure AD configuration guide (see: https://django-auth-adfs.readthedocs.io/en/latest/azure_ad_config_guide.html).

```
from drf_spectacular.extensions import OpenApiAuthenticationExtension
from drf_spectacular.plumbing import build_bearer_security_scheme_object

class AdfsAccessTokenAuthenticationScheme(OpenApiAuthenticationExtension):
    target_class = 'django_auth_adfs.rest_framework.AdfsAccessTokenAuthentication'
    name = 'jwtAuth'

    def get_security_definition(self, auto_schema):
        return build_bearer_security_scheme_object(header_name='AUTHORIZATION',
                                                   token_prefix='Bearer',
                                                   bearer_format='JWT')
```

### 1.6.10 django-parler-rest

django-parler-rest integration for translation package django-parler.

```
from django.conf import settings

from drf_spectacular.extensions import OpenApiSerializerFieldExtension
from drf_spectacular.plumbing import build_object_type


class TranslationsFieldFix(OpenApiSerializerFieldExtension):
    target_class = 'parler_rest.fields.TranslatedFieldsField'

    def map_serializer_field(self, auto_schema, direction):
        # Obtain auto-generated sub-serializer from parler_rest
        # Contains the fields wrapped in parler.models.TranslatedFields()
        translation_serializer = self.target.serializer_class
        # resolve translation sub-serializer into reusable component.
        translation_component = auto_schema.resolve_serializer(
            translation_serializer, direction
        )
        # advertise each language provided in PARLER_LANGUAGES
        return build_object_type(
            properties={
                i['code']: translation_component.ref for i in settings.PARLER_
↪LANGUAGES[None]
            }
        )
```

## 1.7 From *drf-yasg* to OpenAPI 3

drf-yasg is an excellent library and the most popular choice for generating OpenAPI 2.0 (formerly known as Swagger 2.0) schemas with Django REST Framework. Unfortunately, it currently does not provide support for OpenAPI 3.x. Migration from *drf-yasg* to *drf-spectacular* requires some modifications, the complexity of which depends on what features are being used.

---

**Note:** In contrast to *drf-yasg*, we don't package Redoc & Swagger UI but serve them via hyperlinked CDNs instead. If you want or need to serve those files yourself, you can do that with the optional drf-spectacular-sidecar package. See *installation instructions* for further details.

---

### 1.7.1 Decorators

- `@swagger_auto_schema` is largely equivalent to *`@extend_schema`*.
  - `operation_description` argument is called `description`
  - `operation_summary` argument is called `summary`
  - `manual_parameters` and `query_serializer` arguments are merged into a single `parameters` argument
  - `security` argument is called `auth`

  - `request_body` arguments is called `request`
    - Use `None` instead of `drf_yasg.utils.no_body`
  - `method` argument doesn't exist, use `methods` instead (also supported by *drf-yasg*)
  - `auto_schema` has no equivalent.
  - `extra_overrides` has no equivalent.
  - `field_inspectors` has no equivalent.
  - `filter_inspectors` has no equivalent.
  - `paginator_inspectors` has no equivalent.
  - Additional arguments are also available: `exclude`, `operation`, `versions`, `examples`.
- `@swagger_serializer_method` is equivalent to *`@extend_schema_field`*.
  - `component_name` can be provided to break the field out as a separate component.
- *`@extend_schema_serializer`* is available for overriding behavior of serializers.
- Instead of using `@method_decorator`, use *`@extend_schema_view`*.
- Instead of using `swagger_schema_field`, use *`@extend_schema_field`* or *`@extend_schema_serializer`*.

### 1.7.2 Helper Classes

- `Parameter` is roughly equivalent to *`OpenApiParameter`*.
  - `in_` argument is called `location`.
  - `schema` argument should be passed as `type`.
  - `format` argument is merged into `type` argument by using *`OpenApiTypes`*.
  - setting the `many` argument to `True` causes the argument to take an array of values, and generates a schema similar to using the drf_yasg `Items` class on the `items` property. The type of the items in the array are defined by the `type` argument.
- `Response` is largely identical to *`OpenApiResponse`*.
  - `schema` argument is called `response`
  - Order of arguments differs, so use keyword arguments.
- *`OpenApiExample`* is available for providing `examples` to *`@extend_schema`*.
- `Schema` is not required and can be eliminated. Use a plain `dict` instead.

### 1.7.3 Types & Formats

In place of separate `drf_yasg.openapi.TYPE_*` and `drf_yasg.openapi.FORMAT_*` constants, `drf-spectacular` provides the *`OpenApiTypes`* enum:

- `TYPE_BOOLEAN` is called *`BOOL`*, but you can use `bool`.
- `TYPE_FILE` should be replaced by *`BINARY`*
- `TYPE_INTEGER` is called *`INT`*, but you can use `int`.
- `TYPE_INTEGER` with `FORMAT_INT32` is called *`INT32`*
- `TYPE_INTEGER` with `FORMAT_INT64` is called *`INT64`*

- TYPE_NUMBER is called *NUMBER*
- TYPE_NUMBER with FORMAT_FLOAT is called *FLOAT*, but you can use float.
- TYPE_NUMBER with FORMAT_DOUBLE is called *DOUBLE* (or *DECIMAL*, but you can use Decimal)
- TYPE_OBJECT is called *OBJECT*, but you can use dict.
- TYPE_STRING is called *STR*, but you can use str.
- TYPE_STRING with FORMAT_BASE64 is called *BYTE* (which is base64 encoded).
- TYPE_STRING with FORMAT_BINARY is called *BINARY*, but you can use bytes.
- TYPE_STRING with FORMAT_DATETIME is called *DATETIME*, but you can use datetime.datetime.
- TYPE_STRING with FORMAT_DATE is called *DATE*, but you can use datetime.date.
- TYPE_STRING with FORMAT_EMAIL is called *EMAIL*
- TYPE_STRING with FORMAT_IPV4 is called *IP4*, but you can use ipaddress.IPv4Address.
- TYPE_STRING with FORMAT_IPV6 is called *IP6*, but you can use ipaddress.IPv6Address.
- TYPE_STRING with FORMAT_PASSWORD is called *PASSWORD*
- TYPE_STRING with FORMAT_URI is called *URI*
- TYPE_STRING with FORMAT_UUID is called *UUID*, but you can use uuid.UUID.
- TYPE_STRING with FORMAT_SLUG has no direct equivalent. Use *STR* or str instead.
- TYPE_ARRAY is handled by providing *OpenApiParameter* with `many=True` as a parameter. There is no need to set the `items` property on the parameter - the presence of `many=True` turns the parameter into an array parameter.
- The following additional types are also available:
  - *ANY* for which you can use `typing.Any`.
  - *DURATION* for which you can use datetime.timedelta.
  - *HOSTNAME*
  - *IDN_EMAIL*
  - *IDN_HOSTNAME*
  - *IRI_REF*
  - *IRI*
  - *JSON_PTR_REL*
  - *JSON_PTR*
  - *NONE* for which you can use None.
  - *REGEX*
  - *TIME* for which you can use datetime.time.
  - *URI_REF*
  - *URI_TPL*

### 1.7.4 Parameter Location

`drf_yasg.openapi.IN_*` constants are roughtly equivalent to constants defined on the *OpenApiParameter* class:

- `IN_PATH` is called *PATH*
- `IN_QUERY` is called *QUERY*
- `IN_HEADER` is called *HEADER*
- `IN_BODY` and `IN_FORM` have no direct equivalent. Instead you can use `@extend_schema(request={"<media-type>": ...})`.
- *COOKIE* is also available.

### 1.7.5 Docstring Parsing

*drf-yasg* has some special handling for docstrings that is not supported by *drf-spectacular*.

It attempts to split the first line from the rest of the docstring to use as the operation summary, and the remainder is used as the operation description. *drf-spectacular* uses the entire docstring as the description. Use the `summary` and `description` arguments of *@extend_schema* instead. Optionally, the docstring can still be used to populate the operation description.

```
# Supported by drf-yasg:
class UserViewSet(ViewSet):
    def list(self, request):
        """
        List all the users.

        Return a list of all usernames in the system.
        """
        ...

# Updated for drf-spectacular using decorator for description:
class UserViewSet(ViewSet):
    @extend_schema(
        summary="List all the users.",
        description="Return a list of all usernames in the system.",
    )
    def list(self, request):
        ...

# Updated for drf-spectacular using docstring for description:
class UserViewSet(ViewSet):
    @extend_schema(summary="List all the users.")
    def list(self, request):
        """Return a list of all usernames in the system."""
        ...
```

In addition, *drf-yasg* also supports named sections, but these are not supported by *drf-spectacular*. Again, use the `summary` and `description` arguments of *@extend_schema* instead:

```
# Supported by drf-yasg:
class UserViewSet(ViewSet):
    """
```

(continues on next page)

(continued from previous page)

```
    list:
        List all the users.

        Return a list of all usernames in the system.

    retrieve:
        Retrieve user

        Get details of a specific user
    """
    ...

# Updated for drf-spectacular using decorator for description:
@extend_schema_view(
    list=extend_schema(
        summary="List all the users.",
        description="Return a list of all usernames in the system.",
    ),
    retrieve=extend_schema(
        summary="Retrieve user",
        description="Get details of a specific user",
    ),
)
class UserViewSet(ViewSet):
    ...
```

### 1.7.6 Authentication

In *drf-yasg* it was necessary to manually describe authentication schemes.

In *drf-spectacular* there is support for auto-generating the security definitions for a number of authentication classes built in to DRF as well as other popular third-party packages. *OpenApiAuthenticationExtension* is available to help tie in custom authentication clasees – see the *customization guide*.

### 1.7.7 Compatibility

For compatibility, the following features of *drf-yasg* have been implemented:

- `ref_name` on `Serializer` `Meta` classes is supported (excluding inlining with `ref_name=None`)
  - See drf-yasg's documentation for further details.
  - The equivalent in `drf-spectacular` is `@extend_schema_serializer(component_name="...")`
- `swagger_fake_view` is available as attribute on views to signal schema generation

## 1.8 Contributing to drf-spectacular

As an open source project, drf-spectacular welcomes any form of contribution. The project was initially forked off DRF's schema generator and has since then been continually receiving improvements from the community. Your contribution matters even if it is only a small one.

Contributions come in different shapes and sizes.

- Documentation improvements, clarifications & fixing typos
- Creating issues for feature requests & bug reports
- Creating pull requests for features and bug fixes
- Questions that highlight inconsistencies or workflow issues
- Adding blueprints for not officially supported 3rd party tools.

### 1.8.1 Issues

Generating schemas is a complicated business and the devil often lies in the details. A concise description with examples goes a long way towards getting a good understanding of the issue at hand. If possible/applicable please include

- A concise description
- Example code that produces the issue
- Generated (partial) schema with the issue
- Stacktraces if an error occurred
- drf-spectacular/Django/DRF versions

### 1.8.2 Pull requests

drf-spectacular prides itself on having very high code coverage and an extensive test suite. It is really well tested, which enables us to maintain quality, reliability, and consistency.

- The git history is important to the project. Please make minimally invasive changes where possible. On receiving feedback, we prefer having a small set of amended commits. Consider using `git commit --amend` and `git push --force` for updating your PR.
- If you have a non-trivial PR please consider getting early feedback. We don't want to waste anyone's time.
- We have great tooling around tests. Have a look into test_regressions.py for inspiration. The tests are mainly structured in feature units but in doubt small things go into the regressions.
- We use tox to make sure drf-spectacular works for a range of Django/DRF versions. Your PR must pass the whole test suite to get merged. Local testing with `./runtests.py` usually suffices. You don't need to install a bunch of python versions. The github actions for PRs will take care of the rest.

A quick cheat sheet to get you rolling

```
$ # fork the repo on github
$ git clone https://github.com/YOURGITHUBNAME/drf-spectacular
$ cd drf-spectacular
$ python -m venv venv
$ source venv/bin/activate
```

(continues on next page)

(continued from previous page)

```
(venv) $ pip install -r requirements.txt
(venv) $ ./runtests.py  # runs tests (pytest) & linting (isort, flake8, mypy)
```

With that out of the way, we hope to hear from you soon.

## 1.9 Changelog

### 1.9.1 0.26.2 (2023-04-15)

- fix jwt cookie name settings not being recognised (#972) [Nix Siow]
- Add OpenApiRequest for encoding options #714 #965

Breaking changes / important additions:

- small bugfix release that also contains the new `OpenApiRequest` feature

### 1.9.2 0.26.1 (2023-03-18)

- reorder typed polymorphic fields #958
- Fix test warnings [Dmitry Gribanov]
- Fix PolymorphicSerializer type field handling #885 #958
- Add PresentablePrimarKeyRelatedField schema for drf-exrta-fields blueprint [Đào Minh Ht]
- bugfix KeyError for disabled ENUM_GENERATE_CHOICE_DESCRIPTION #952

Breaking changes / important additions:

- small bugfix release that addresses a issue when turning off choice description generation
- improve/bugfix `PolymorphicSerializer` type field handling

### 1.9.3 0.26.0 (2023-03-04)

- honor djangorestframework_camel_case settings “ignore_keys” and “ignore_fields” #945
- If available, use docstrings from properties for field descriptions (#954)
- Don’t let validators override values already set in the schema (#911) [StopMotionCuber]
- add test and another case to #901
- add enum key/value list to description string #337 #403 #105 #563
- Add option to provide a callable for PolymorphicProxySerializer.serializers [Glenn Matthews]
- consolidate sort fix for enum sorting
- add testcase to #950. ensure raw schema dict remains unmodified
- Don’t edit the orignal django-filters schema. [Will Giddens]
- Fix typos and grammer errors in FAQ doc page. [Foad Lind]
- fix OpenApiResponse nested example defaults #875

- mitigate `runtests.py` fail when GDAL library is not installed #945 #821 #775 #777
- bugfix SlugRelatedField with a model property target #943
- suppress erroneous warning for optional extensions #940
- fix whitelist mechanics (enables deny all) #923
- mitigate many=True with default array value #936
- fix dj-rest-auth>=3.0.0 breaking changes #937
- Update plumbing.py
- add swagger UI template blocks for customization [Jan Lis]
- Add support for drf ReturnList and ReturnDict hint [zengqiu]
- add example/test for DynamicFieldsModelSerializer #375 #912
- adapt test schema for dj-rest-auth 2.2.6
- clarify docs for postproc hook mechanics #908
- Add test for custom serializer field pagination #904
- fix: let use a default value for foreignkey model field [Frederic de Zorzi]

Breaking changes / important additions:

- A lot of bug fixes and a few feature additions.
- We now render a descriptive `Enum` key/value list into the description by default. Opt-out with new setting `ENUM_GENERATE_CHOICE_DESCRIPTION`.
- Beware that we now extract more docstrings. Check your schema diff on update whether you are now leaking unintended information.
- The `whitelist` mechanics changed slightly on what is considered default behavior.
- Fix a breaking change in `dj-rest-auth>=3.0.0`
- It should not be possible to run the tests without installing system libraries like GDAL for the contrib tests

### 1.9.4 0.25.1 (2022-12-16)

- Fix warning source line performance regression #889 #897
- improve warning for transient @api_view objects #889
- adapt package arg due to setuptools deprecation #786
- utilize queryset for SlugRelatedField #897

Breaking changes / important additions:

- Bugfix release that addresses a performance regression in `SpectacularApiView` and an oversight in the now stricter handling of `SlugRelatedField`

### 1.9.5 0.25.0 (2022-12-13)

- Fix missing description for ManyRelatedField and tested for SlugField (#895) [StopMotionCuber]
- Simplify hashable_values #833
- Add custom settings to CLI (view parity) #892
- fix function misnomer #891
- improve trace messages / warnings & add color #866
- Treat SlugRelatedField analog to PrimaryKeyRelatedField #854
- Include filename in call to _get_sidecar_url [Justin Spencer]
- add django-parler blueprint #887
- add a view to handle SwaggerUI oauth callbacks (#882) [Finn-Thorben Sell]
- improve documentation
- Introduce setting DEFAULT_QUERY_MANAGER to allow other managers for querset retrieval
- fix flake8 6.0.0 breaking change
- fix example list detection (symmetry with schema) #872
- Use direct view methods for getting serializer instances [Numerlor]
- name overrides for rest_framework_dataclasses #839
- decouple TypedDict class from Py version #861 #654
- bugfix djangorestframework_camel_case #861
- bugfix djangorestframework_camel_case nested object handling #861
- Utils: Replace `List[]` with `Sequence[]`, because of Mypy note 'List is invariant. Consider using Sequence instead.' [Hans Aarne Liblik]
- Fixed minor typos [Conrad]
- Removing blank and null keys when generating the overridden choices hash to match the hash generation logic in the enum post processor hook [Trent Holliday]
- fix test fixture overlap #826
- specifiy min patch release for DRF (fixes #812)
- Preserve context in `get_list_serializer`. [Brady Dean]
- Allow field extensions to return None from map_serializer_field [Andrew Backer]

Breaking changes / important additions:

- Officially set the lower bound for DRF version to `3.10.3`
- Refactored the CLI warning system for better code navigation / orientation, GUI support and color!
- Some minor mechanics changes, several overall improvements, feature additions, and a few bugfixes.

### 1.9.6 0.24.2 (2022-09-26)

- robustify extension class loading #821
- fix regression due to GIS import for django-filter #821

Breaking changes / important additions:

- Hotfix release to mitigate optional GDAL import errors for django-filter.

### 1.9.7 0.24.1 (2022-09-23)

- bugfix GeometryFilter for GIS and django-filter #814
- NullBooleanField comment and add 3.14.0 to test suite #818
- fix: #816 NullBooleanField does not exist in DRF >= 3.14.0 [Laurent Tramoy]
- fix GIS source lookup with hops #813
- Add blueprint for Stoplight Elements docs UI [Alex Burgel]
- fix OpenApiParameter enum and pattern for many=True #808

Breaking changes / important additions:

- Hotfix release to mitigate removal of `NullBooleanField` in DRF 3.14.0
- Small fixes to OpenApiParameter and `django-filter`

### 1.9.8 0.24.0 (2022-09-14)

- fix yaml serialization error on Django SafeString #802
- mitigate DRF bug in ObtainAuthToken < 3.12.0 #796
- add FAQ entry for django-csp errors #173 #797
- bugfix TokenMatchesOASRequirements #469 and JWTCookieAuthentication #626
- add custom redoc settings option
- fix error with PrimaryKeyRelatedField on non-ModelSerializer #353
- provide context to serializer for @extend_schema use-cases #699
- add example value hint to doc #788
- fix packages= so top_level.txt is correct [anthony sottile]
- Adding documentation for the OpenApiParameter 'many' argument [Paul Wayper]
- Extend OpenApiSerializerExtension interface. #392 #705
- Include context with request when instantiating serializers [Mike Hansen]

Breaking changes / important additions:

- Some minor gaps closed in the extension interface and serializer context initialization. It is a y-stream release, because there remains a small chance of change for users that sport non-standard customizations.

### 1.9.9 0.23.1 (2022-07-26)

- improve CAMELIZE_NAMES doc #774
- move import into build_geo_schema function [bidaya0]

Breaking changes / important additions:

- Hotfix release to mitigate unwanted import of optional GIS features that depend on GDAL. GDAL is **not** a new requirement.

### 1.9.10 0.23.0 (2022-07-25)

- fix infinite recursion when accessing missing attributes in generator stats [Oleg Hoefling]
- fix list pagination when examples are provided [topher235]
- accept integer status codes in OpenApiExample [Nicholas Guriev]
- Missing ":" in example documentation [Josué Millán Zamora]
- Flip direction for callbacks serializers [Justas]
- grammar fix [Kojo Idrissa]
- fix sidecar for alternate staticfile storages #718
- add support for `rest_framework_gis`
- add mechanism to handle custom ListSerializers with extensions
- Update based on review [johnthagen]
- Hyphenate in-memory [johnthagen]
- Add FAQ entry for how to serve in-memory generated files [johnthagen]
- add pattern to OpenApiParameter #738
- Add test that extend_schema_field on django-filter is not modified [Take Weiland]
- Do not forcefully overwrite enum setting on custom django-filter schema [Take Weiland]
- django-filter: Enable type extraction fallback for MultipleChoiceFilter as well [Take Weiland]
- Add examples camelization note to settings.py [Zac Miller]
- fix codecov badge url issue on github #713

Breaking changes / important additions:

- A whole bunch of smaller bug fixes.
- OpenAPI Callbacks should now be production ready
- Introduction of `rest_framework_gis` support. This might impact APIs that are using GIS so this is a y-stream release.

### 1.9.11 0.22.1 (2022-04-25)

- Update customization.rst [Lane Zhang]
- Remove invalid example in drf-yasg migration documentation. [Nick Pope]
- Avoid using default role in documentation. [Nick Pope]
- Small documentation fixes. [Nick Pope]
- improve parameter many handling and warnings #703
- bugfix unconsidered warnings/errors for return code #706 #702
- Include a list of applications urls as a parameter for SERVE_URLCONF #709 [anoirak]
- bugfix/improve analyze_named_regex_pattern(path) #697 [Jon Iturmendi]
- django-filter: added type extraction fallback for ChoiceFields #690
- fix test, more precise naming, also wrap validation #693
- bugfix PolymorphicProxySerializer many handling and add manual mode #692
- Use Django management CommandError to eliminate the traceback on error [Brandon W Maister]
- add `swagger_fake_view` FAQ entry #321
- Fix #688 - avoid a TypeError when ChoiceFilter choices are a callable [Glenn Matthews]
- map explicit float hints/decoration to double #687 #674

Breaking changes / important additions:

- Small release consisting of minor bug fixes, improved `PolymorphicProxySerializer`, cleaned up documentation, and some improvements to **django-filter**

### 1.9.12 0.22.0 (2022-03-21)

- Added `detype_patterns()` with `@cache`. [Nick Pope]
- add "externalDocs" to operation via extend_schema #681
- warn on invalid components names #685
- wrap examples in list/pagination when serializer is many=True #641 #640 #595
- python's and django's float is really "double precision" #674
- Support negative numbers in pattern regex for coerced decimal fields [Mike Hansen]
- add OpenAPI callback operations #665
- Keep the urlpatterns in the apiview and pass it to the generator [Jorge Cardona]
- django-filter: raise priority of explicitly given filter method type hints #660
- also allow @extend_schema_field on django-filter filter method #660
- accommodate pyright limitations #657
- fix doc extraction for built-in types #654
- use get_doc for description [Josh Ferge]
- add more information to resolved TypedDicts [Josh Ferge]
- fix url escaping bug introduced in #556 (#650)

- pass through version from UI to schema endpoint #650
- factor out schema_url generation #650
- relax AcceptHeaderVersioning contraint for modification #650
- Enable the use of lists in extend_schema_view() [François Travais]

Breaking changes / important additions:

- This is a y-stream release with a lot of bugfixes, some new features and potentially small schema changes (if affected features are used).
- Examples are now wrapped in pagination/lists when endpoint/serializer is `many=True`
- django-filter had some internal restructuring and thus overrides are now always honored.
- added callback functionality (EXPERIMENTAL and subject to change due to pending issue)
- Many thanks to all the contributors!

### 1.9.13 0.21.2 (2022-02-01)

- Add support for djangorestframework-dataclasses [Oxan van Leeuwen]
- add version to schema for AcceptHeaderVersioning #637
- FAQ for @api_view #635
- add extensions for dj_rest_auth's JWTCookieAuthentication #626

Breaking changes / important additions:

- Some minor bugfixes and feature additions. Schemas using AcceptHeaderVersioning contain a small change.

### 1.9.14 0.21.1 (2021-12-20)

- add root level extension setting #619
- ease schema browser handling with "Content-Disposition" #607
- custom settings per SpectacularAPIView instance #365
- Support new X | Y union syntax in Python 3.10 (PEP 604) [Marti Raudsepp]
- upstream release updates, compat test fix for jwt, consistency fix
- add blueprint for django-auth-adfs [1110sillabo]
- use is_list_serializer instead of isinstance() [Roman Sichnyi]
- Fix schema generation for RecursiveField(many=True) [Roman Sichnyi]
- enable clearing auth methods with empty list #99
- Fix typos in the code example [Marcin Kurczewski]

Breaking changes / important additions:

- Some minor bugfixes and small feature additions. No large schema changes are expected

### 1.9.15 0.21.0 (2021-11-10)

- add renderer & parser whitelist setting #598
- catch attr exception for invalid SerializerMethodField #592
- add regression test for catch-all status codes #573
- bugfix OpenApiResponse without description argument #591
- introduce direction literal / import consolidation #582
- mitigate CORS issues for external requests in Swagger UI #588
- Swagger UI authorized schema retrieval #342 #458
- remove cyclic import warning as fixes haves mitigated the issue. #581
- bugfix: anchor parameter patterns with ^$
- bugfix isolation of derivatives for @extend_schema_serializer/@extend_schema_field #585
- add support for djangorestframework-recursive #586
- Add blueprint for drf-extra-fields Base64FileField [johnthagen]
- Add note about extensions registering themselves [johnthagen]
- Document alternative to drf-yasg swagger_schema_field [johnthagen]
- allow to bypass list detection for filter discovery #407
- add blueprint (closes #448), fix test misnomer
- non-blank string enforcement for parameters #282
- add setting ENFORCE_NON_BLANK_FIELDS to enable blank checks #186

Breaking changes / important additions:

- Fixed two more decorator isolation issues.
- Added Swagger UI plugin to handle reloading the schema on authentication changes (`'SERVE_PUBLIC': False`).
- Added `minLength` where a blank value is not allowed. Apart the the dedicated setting, it is implicitly enabled by `COMPONENT_SPLIT_REQUEST`.
- Several other small fixes and additional settings for corner cases. This is mainly a y-steam release due to the potential impact on the Swagger UI and `minLength` changes.

### 1.9.16 0.20.2 (2021-10-15)

- add setting for manual path prefix: SCHEMA_PATH_PREFIX_INSERT #567
- improve type hint for @extend_schema_field #569
- bugfix COMPONENT_SPLIT_REQUEST for empty req/resp serializers #572
- Make it cleared that ENUM_NAME_OVERRIDES is a key within SPECTACULAR_SETTINGS [johnthagen]
- Improve formatting in customization docs [johnthagen]
- bugfix @extend_schema_view on @api_view #554
- bugfix isolation for @extend_schema/@extend_schema_view reorg #554

- Fix inheritance bugs with @extend_schema_view(). [Nick Pope]
- Allow methods in @extend_schema to be case insensitive. [Nick Pope]
- Added a documentation blueprint for RapiDoc. [Nick Pope]
- Tidy templates for documentation views. [Nick Pope]
- Use latest version for CDN packages. [Nick Pope]

Breaking changes / important additions:

- Mainly a bugfix release that solves several longstanding issues with `@extend_schema_view`/`@extend_schema` annotation isolation. There should be no more side effects from arbitrarily mixing and matching the decorators.
- Improved handling of completely empty serializers with COMPONENT_SPLIT_REQUEST.

### 1.9.17 0.20.1 (2021-10-03)

- move swagger CDN to jsdelivr (unpkg has been flaky)
- bugfix wrong DIST setting in Redoc #546
- Allow paginated_name customization [Georgy Komarov]

Breaking changes / important additions:

- Hotfix release due to regression in the Redoc template

### 1.9.18 0.20.0 (2021-10-01)

- Add support for specification extensions. [Nick Pope]
- add example injection for (discovered) parameters #414
- Fix crash with read-only polymorphic sub-serializer. [Nick Pope]
- Add arbitrarily deep ListSerializer nesting #539
- tighten serializer assumptions #539
- fix whitespace stripping on methods
- Rename `AutoSchema._map_field_validators()` → `.insert_field_validators()`. [Nick Pope]
- Rename `AutoSchema._map_min_max()` → `.insert_min_max()`. [Nick Pope]
- Fix detection of int64 from min/max values. [Nick Pope]
- Fix zero handling in _map_min_max(). [Nick Pope]
- Add support for introspection of nested validators. [Nick Pope]
- Fix invalid schemas caused by validator introspection. [Nick Pope]
- Overhaul validator logic. [Nick Pope]
- support multiple headers in OpenApiAuthenticationExtension #537
- docs: Missing end quote for INSTALLED_APPS [Prayash Mohapatra]
- update doc #530
- introducing the spectacular sidecar
- fallback improvements to typing system with typing_extensions

Breaking changes / important additions:

- Added vendor specification extensions
- Completetly overhauled validator logic and bugfixes
- Offline UI assets with optional *drf-spectacular-sidecar* package
- several internal logic improvements and stricter assumptions

### 1.9.19 0.19.0 (2021-09-21)

- fix/cleanup suffixed path variable coercion #516
- remove superseded Request mock from oauth_toolkit
- be gracious on Enums that are not recognized by DRF #500
- remove non-required empty descriptions
- added test case for lookup_field #524
- Fix grammatical typo [johnthagen]
- remove mapping for re.Pattern (no 3.6 and mypy issues) #526
- Add missing types defined in specification. [Nick Pope]
- Add type mappings for IP4, IP6, TIME & DURATION. [Nick Pope]
- add support for custom converters and coverter override #502
- cache static loading function calls
- prevent settings loading in types, lazy load in plumbing instead
- lazy settings loading in drainage
- Improve guide for migration from drf-yasg. [Nick Pope]
- handle default value for SerializerMethodField #422
- consolidate bearer scheme generation & bugfix #515
- prevent uncaught exception on modified django-filter #519
- add decoupled model docstrings #522
- Fix warnings raised during testing. [Nick Pope]
- add name override to @extend_schema_serializer #517
- Fix deprecation warning about default_app_config from Django 3.2+ [Janne Rönkkö]
- Remove obsolete value from IMPORT_STRINGS. [Nick Pope]
- Add extension for TokenVerifySerializer. [Nick Pope]
- Use SESSION_COOKIE_NAME in SessionScheme. [Nick Pope]
- add regex path parameter extraction for explicit cases #510
- honor lookup_url_kwarg name customization #509
- add contrib compat tests for drf-nested-routers
- improve path coersion model resolution
- add test_fields API response test #501

- Handle 'lookup_field' containing relationships for path parameters [Luke Plant]
- add BinaryField case to tests #506
- fix: BinaryField's schema type should be string #505 (#506) [jtamm-red]
- bugfix incomplete regex stripping for literal dots #507
- Fix tests [Jameel Al-Aziz]
- Fix type hint support for functools cached_property wrapped funcs [Jameel Al-Aziz]
- Extend enum type hint support to more Enum subclasses [Jameel Al-Aziz]

Breaking changes / important additions:

- Severely improved path parameter detection for Django-style parameters, RE parameters, and custom converters
- Significantly more defensive settings loading for safer project imports (less prone to import loops)
- Improved type hint support for `Enum` and other native types
- Explicit support for *drf-nested-routers*
- A lot more small improvements

### 1.9.20 0.18.2 (2021-09-04)

- fix default value handling for custom ModelField #422
- fill html title with title from settings #491
- add Enum support in type hints #492
- Move system check registration to AppConfig [Jameel Al-Aziz]

Breaking changes / important additions:

- Primarily ironing out another issue with the Django check and some minor improvements

### 1.9.21 0.18.1 (2021-08-31)

- Improved docs regarding how ENUM_NAME_OVERRIDES works [Luke Plant]
- bugfix raw schema handling for @extend_schema_field on SerializerMethodField method 481
- load common SwaggerUI dep SwaggerUIStandalonePreset #483
- allow versioning of SpectacularAPIView via query #483
- update swagger UI
- move checks to "–deploy" section, bugfix public=True #487

Breaking changes / important additions:

- This is a hotfix release as the newly introduced Django check was executing the wrong code path.
- Check also moved into the `--deploy` section to prevent double execution. This can be disabled with `ENABLE_DJANGO_DEPLOY_CHECK`
- Facitities added to utilize SwaggerUI Topbar for versioning.

### 1.9.22 0.18.0 (2021-08-25)

- prevent exception and warn when ReadOnlyField is used with non-ModelSerializer #432
- allow raw JS in Swagger settings #457
- add support for check framework #477
- improve common FAQ @action question #399
- update @extend_schema doc #476
- adapt to changes in iMerica/dj-rest-auth 2.1.10 (ResendEmailVerification)
- add raw schema to @extend_schema(request={MIME: RAW}) #476
- bugfix test case for 3.6 #474
- bugfix header underscore handling for simplejwt #474
- properly parse TokenMatchesOASRequirements (oauth toolkit) #469
- add whitelist setting to manage auth method exposure #326 #471
- Update set_password instead of list [Greg Campion]
- Update documentation to illustrate how to override a specific method [Greg Campion]

Breaking changes / important additions:

- This is a y-stream release because we added Django checks which might emit warnings and subsequently break CI. This can be easily suppressed with Django's `SILENCED_SYSTEM_CHECKS`.
- Several small fixes and features that should not have a big impact.

### 1.9.23 0.17.3 (2021-07-26)

- port custom "Bearer" bugfix/workaround to simplejwt #467
- add setting for listing/paginating/filtering on non-2XX #402 #277
- fix Typo [Eunsub LEE]
- nit typofix [adamsteele-city]
- Add a few return type annotations [Nikhil Benesch]
- add django-filter queryset annotation and `extend_schema_field` support
- account for functools.partial wrapped type hints #451
- Update swagger_ui.js [Jordan Facibene]
- Update customization.rst to fix example typo [Atsuo Shiraki]
- update swagger-ui version
- add oauth2 config for swagger ui #438

Breaking changes / important additions:

- Just a few bugfixes and some small features with minimal impact on existing schema

### 1.9.24 0.17.2 (2021-06-15)

- prevent endless loop in extensions when augmenting schema #426
- bugfix secondary import cycle (generics.APIView) #430
- fix: avoid circular import of/via rest_framework's APIView [Daniel Hahler]

Breaking changes / important additions:

- Hotfix release that addresses a carelessly added import in 0.17.1. In certain use-cases, this may have led to an import cycle inside DRF.

### 1.9.25 0.17.1 (2021-06-12)

- bugfix 201 response for (List)CreateAPIVIew #428
- support paginated ListSerializer with field child #413
- fix django-filter.BooleanFilter subclass issue #317
- serializer field deprecation #415
- improve extension documentation #426
- improve type hints and fix mypy issues on tests.
- add missing usage case to type hints #418
- Typo(?) README fix [Jan Jurec]

Breaking changes / important additions:

- This release is mainly for fixing incomplete type hints which mypy will potentially complain about.
- A few small fixes that should either have no or a very small impact in schemas.

### 1.9.26 0.17.0 (2021-06-01)

- improve type hint detection for Iterable and NamedTuple #404
- bugfix ReadOnlyField when used as ListSerlializer child #404
- improve component discard logic #395
- allow disabling operation sorting for sorting in PREPROCESSIN_HOOKS #410
- add regression test for #407
- fix error on read-only serializer [Matthieu Treussart]
- invert component exclusion logic (OpenApiSerializerExtension) #351 #391
- add many=True support to PolymorphicProxySerializer #382
- improve documentation, remove py2 wheel tag, mark as mypy-enabled
- bugfix YAML serialization errors that are ok with JSON #388
- bugfix missing auth extension for JWTTokenUserAuthentication #387
- Rename MethodSerializerField -> SerializerMethodField in README [Christoph Krybus]

Breaking changes / important additions:

- Quite a few small improvements. The biggest change is the inversion of the component discard logic. This should have no negative impact, but to be on the safe side we'll opt for a y-stream release.
- The package is now marked as being typed, which should get picked up natively by mypy

### 1.9.27 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.

### 1.9.28 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.

### 1.9.29 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.

### 1.9.30 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`

### 1.9.31 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

### 1.9.32 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]

### 1.9.33 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*.

### 1.9.34 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

### 1.9.35 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.

### 1.9.36 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.

### 1.9.37 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

### 1.9.38 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

### 1.9.39 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.

### 1.9.40 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.

### 1.9.41 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

### 1.9.42 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

### 1.9.43 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]

### 1.9.44 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

### 1.9.45 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

### 1.9.46 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]

### 1.9.47 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

### 1.9.48 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.

### 1.9.49 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`

### 1.9.50 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

### 1.9.51 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.

### 1.9.52 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.

### 1.9.53 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`

### 1.9.54 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.

### 1.9.55 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.

### 1.9.56 0.8.3 (2020-03-02)

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

### 1.9.57 0.8.2 (2020-03-02)

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

### 1.9.58 0.8.1 (2020-03-01)

- Initial published version.

## 1.10 License

## 1.11 Package overview

### 1.11.1 drf_spectacular.utils

class drf_spectacular.utils.**OpenApiCallback**(*name: str*, *path: str*, *decorator: Union[Callable[[drf_spectacular.utils.F], drf_spectacular.utils.F], Dict[str, Callable[[drf_spectacular.utils.F], drf_spectacular.utils.F]], Dict[str, Any]]*)

Bases: *drf_spectacular.utils.OpenApiSchemaBase*

Helper class to bundle a callback definition. This specifies a view on the callee's side, effectively stating the expectations on the receiving end. Please note that this particular *@extend_schema* instance operates from the perspective of the callback origin, which means that `request` specifies the outgoing request.

For convenience sake, we assume the callback sends `application/json` and return a `200`. If that is not sufficient, you can use `request` and `responses` overloads just as you normally would.

**Parameters**

- **name** – Name under which the this callback is listed in the schema.
- **path** – Path on which the callback operation is performed. To reference request body contents, please refer to OpenAPI specification's key expressions for valid choices.
- **decorator** – *@extend_schema* decorator that specifies the receiving endpoint. In this special context the allowed parameters are `requests`, `responses`, `summary`, `description`, `deprecated`.

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*, *parameter_only: Optional[Tuple[str, typing_extensions.Literal[query, path, header, cookie]]] = None*, *media_type: Optional[str] = None*, *status_codes: Optional[Sequence[Union[str, int]]] = None*)

Bases: *drf_spectacular.utils.OpenApiSchemaBase*

Helper class to document a API parameter / request body / response body with a concrete example value.

It is recommended to provide a singular example value, since pagination and list responses are handled by drf-spectacular.

The example will be attached to the operation object where appropriate, i.e. where the given `media_type`, `status_code` and modifiers match. Example that do not match any scenario are ignored.

- media_type will default to 'application/json' unless implicitly specified through *OpenApiResponse*
- status_codes will default to [200, 201] unless implicitly specified through *OpenApiResponse*

class drf_spectacular.utils.**OpenApiParameter**(*name: str, type: Union[rest_framework.serializers.Serializer, Type[rest_framework.serializers.Serializer], Type[Union[str, float, bool, bytes, int, dict, uuid.UUID, decimal.Decimal, datetime.datetime, datetime.date, datetime.time, datetime.timedelta, ipaddress.IPv4Address, ipaddress.IPv6Address]], drf_spectacular.types.OpenApiTypes, dict] = <class 'str'>, location: typing_extensions.Literal[query, path, header, cookie] = 'query', required: bool = False, description: str = '', enum: Optional[Sequence[Any]] = None, pattern: Optional[str] = None, deprecated: bool = False, style: Optional[str] = None, explode: Optional[bool] = None, default: Optional[Any] = None, allow_blank: bool = True, many: Optional[bool] = None, examples: Optional[Sequence[drf_spectacular.utils.OpenApiExample]] = None, extensions: Optional[Dict[str, Any]] = None, exclude: bool = False, response: Union[bool, Sequence[Union[str, int]]] = False*)

Bases: *drf_spectacular.utils.OpenApiSchemaBase*

Helper class to document request query/path/header/cookie parameters. Can also be used to document response headers.

Please note that not all arguments apply to all `location`/`type`/direction variations, e.g. path parameters are `required=True` by definition.

For valid `style` choices please consult the OpenAPI specification.

**COOKIE: typing_extensions.Final = 'cookie'**

**HEADER: typing_extensions.Final = 'header'**

**PATH: typing_extensions.Final = 'path'**

**QUERY: typing_extensions.Final = 'query'**

class drf_spectacular.utils.**OpenApiRequest**(*request: Optional[Any] = None*, *encoding: Optional[Dict[str, Dict[str, Any]]] = None*, *examples: Optional[Sequence[*drf_spectacular.utils.OpenApiExample*]] = None*)

Bases: *drf_spectacular.utils.OpenApiSchemaBase*

Helper class to combine a request object (`Serializer`, `OpenApiType`, raw schema, etc.) together with an encoding object and/or examples. Examples can alternatively be provided via *@extend_schema*.

This class is especially helpful for customizing value encoding for `application/x-www-form-urlencoded` and `multipart/*`. The encoding parameter takes a dictionary with field names as keys and encoding objects as values. Refer to the specification on how to build a valid encoding object.

class drf_spectacular.utils.**OpenApiResponse**(*response: Optional[Any] = None*, *description: str = ''*, *examples: Optional[Sequence[*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*.

This class is especially helpful for explicitly describing status codes on a "Response Object" level.

**class** drf_spectacular.utils.**OpenApiSchemaBase**
Bases: object

**class** drf_spectacular.utils.**PolymorphicProxySerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.Serializer

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 it will raise an AssertionError if used in that way. It **cannot** be used in views as `serializer_class` or as field in an actual serializer. It is solely meant for annotation purposes.

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. Another use-case is explicit control over sub-serializer's `many` attribute. To explicitly control this aspect, you need disable the discriminator with `resource_type_field_name=None` as well as disable automatic list handling with `many=False`.

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.

It is also permissible to provide a callable with no parameters for `serializers`, such as a lambda that will return an appropriate list or dict when evaluated.

**property data**

**property serializers**

**to_internal_value**(*data*)
Dict of native values <- Dict of primitive datatypes.

**to_representation**(*instance*)
Object instance -> Dict of primitive datatypes.

drf_spectacular.utils.**extend_schema**(*operation_id: Optional[str] = None, parameters: Optional[Sequence[Union[drf_spectacular.utils.OpenApiParameter, rest_framework.serializers.Serializer, Type[rest_framework.serializers.Serializer]]]] = None, request: Any = <class 'rest_framework.fields.empty'>, responses: Any = <class 'rest_framework.fields.empty'>, auth: Optional[Sequence[str]] = None, description: Optional[str] = None, summary: Optional[str] = None, deprecated: Optional[bool] = None, tags: Optional[Sequence[str]] = None, filters: Optional[bool] = None, exclude: bool = False, operation: Optional[Dict] = None, methods: Optional[Sequence[str]] = None, versions: Optional[Sequence[str]] = None, examples: Optional[Sequence[drf_spectacular.utils.OpenApiExample]] = None, extensions: Optional[Dict[str, Any]] = None, callbacks: Optional[Sequence[drf_spectacular.utils.OpenApiCallback]] = None, external_docs: Optional[Union[Dict[str, str], str]] = None*) → Callable[[drf_spectacular.utils.F], drf_spectacular.utils.F]

Decorator mainly for the "view" method kind. Partially or completely overrides what would be otherwise 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)
  - 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.
  - `dict` with status codes as keys and one of the above as values. Additionally in this case, it is also possible to provide a raw schema dict as value.
  - `dict` with tuples (status_code, media_type) as keys and one of the above as values. Additionally in this case, it is also possible to provide a raw schema dict as value.
- **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. Additionally, in this case, it is also possible to provide a raw schema dict as value.
- **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
- **filters** – ignore list detection and forcefully enable/disable filter discovery
- **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
- **extensions** – specification extensions, e.g. `x-badges`, `x-code-samples`, etc.
- **callbacks** – associate callbacks with this endpoint
- **external_docs** – Link external documentation. Provide a dict with an "url" key and optionally a "description" key. For convenience, if only a string is given it is treated as the URL.

**Returns**

drf_spectacular.utils.**extend_schema_field**(*field: Union[rest_framework.serializers.Serializer, Type[rest_framework.serializers.Serializer], rest_framework.fields.Field, Type[rest_framework.fields.Field], drf_spectacular.types.OpenApiTypes, Dict]*, *component_name: Optional[str] = None*) → Callable[[drf_spectacular.utils.F], drf_spectacular.utils.F]

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[Sequence[str]] = None*, *deprecate_fields: Optional[Sequence[str]] = None*, *examples: Optional[Sequence[drf_spectacular.utils.OpenApiExample]] = None*, *extensions: Optional[Dict[str, Any]] = None*, *component_name: Optional[str] = None*) → Callable[[drf_spectacular.utils.F], drf_spectacular.utils.F]

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.
- **deprecate_fields** – fields to mark as deprecated while processing the serializer.
- **examples** – define example data to serializer.
- **extensions** – specification extensions, e.g. `x-is-dynamic`, etc.
- **component_name** – override default class name extraction.

drf_spectacular.utils.**extend_schema_view**(***kwargs*) → Callable[[drf_spectacular.utils.F], drf_spectacular.utils.F]

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.

This decorator also takes care of safely attaching annotations to derived view methods, preventing leakage into unrelated views.

**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

### 1.11.2 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`)

**NUMBER = 1**

Converted to `{"type": "number"}`.

**FLOAT = 2**

Converted to `{"type": "number", "format": "float"}`. Equivalent to `float`.

**DOUBLE = 3**

Converted to `{"type": "number", "format": "double"}`.

**BOOL = 4**

Converted to `{"type": "boolean"}`. Equivalent to `bool`.

**STR = 5**

Converted to `{"type": "string"}`. Equivalent to `str`.

**BYTE = 6**
Converted to {"type": "string", "format": "byte"}. Use this for base64-encoded data wrapped in a string.

**BINARY = 7**
Converted to {"type": "string", "format": "binary"}. Equivalent to bytes. Use this for raw binary data.

**PASSWORD = 8**
Converted to {"type": "string", "format": "password"}.

**INT = 9**
Converted to {"type": "integer"}. Equivalent to int.

**INT32 = 10**
Converted to {"type": "integer", "format": "int32"}.

**INT64 = 11**
Converted to {"type": "integer", "format": "int64"}.

**UUID = 12**
Converted to {"type": "string", "format": "uuid"}. Equivalent to UUID.

**URI = 13**
Converted to {"type": "string", "format": "uri"}.

**URI_REF = 14**
Converted to {"type": "string", "format": "uri-reference"}.

**URI_TPL = 15**
Converted to {"type": "string", "format": "uri-template"}.

**IRI = 16**
Converted to {"type": "string", "format": "iri"}.

**IRI_REF = 17**
Converted to {"type": "string", "format": "iri-reference"}.

**IP4 = 18**
Converted to {"type": "string", "format": "ipv4"}. Equivalent to IPv4Address.

**IP6 = 19**
Converted to {"type": "string", "format": "ipv6"}. Equivalent to IPv6Address.

**HOSTNAME = 20**
Converted to {"type": "string", "format": "hostname"}.

**IDN_HOSTNAME = 21**
Converted to {"type": "string", "format": "idn-hostname"}.

**DECIMAL = 22**
Converted to {"type": "number", "format": "double"}. The same as *DOUBLE*. Equivalent to Decimal.

**DATETIME = 23**
Converted to {"type": "string", "format": "date-time"}. Equivalent to datetime.

**DATE = 24**
Converted to {"type": "string", "format": "date"}. Equivalent to date.

**TIME = 25**
Converted to {"type": "string", "format": "time"}. Equivalent to time.

**DURATION = 26**
Converted to {"type": "string", "format": "duration"}. Equivalent to timedelta. Expressed according to ISO 8601.

**EMAIL = 27**
Converted to {"type": "string", "format": "email"}.

**IDN_EMAIL = 28**
Converted to {"type": "string", "format": "idn-email"}.

**JSON_PTR = 29**
Converted to {"type": "string", "format": "json-pointer"}.

**JSON_PTR_REL = 30**
Converted to {"type": "string", "format": "relative-json-pointer"}.

**REGEX = 31**
Converted to {"type": "string", "format": "regex"}.

**OBJECT = 32**
Converted to {"type": "object", ...}. Use this for arbitrary free-form objects (usually a dict). The additionalProperties item is added depending on the GENERIC_ADDITIONAL_PROPERTIES setting.

**NONE = 33**
Equivalent to None. This signals that the request or response is empty.

**ANY = 34**
Converted to {} which sets no type and format. Equivalent to typing.Any.

### 1.11.3 drf_spectacular.views

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

**api_version = None**

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

**custom_settings = None**

**generator_class**
alias of drf_spectacular.generators.SchemaGenerator

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

**patterns = None**

**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'

title = ''

url = None

url_name = 'schema'

class drf_spectacular.views.**SpectacularSwaggerOauthRedirectView**(***kwargs*)
Bases: django.views.generic.base.RedirectView

A view that serves the SwaggerUI oauth2-redirect.html file so that SwaggerUI can authenticate itself using Oauth2

This view should be served as `./oauth2-redirect.html` relative to the SwaggerUI itself. If that is not possible, this views absolute url can also be set via the `SPECTACULAR_SETTINGS.SWAGGER_UI_SETTINGS.oauth2RedirectUrl` django settings.

get_redirect_url(**args*, ***kwargs*)
Return the URL redirect to. Keyword arguments from the URL pattern match generating the redirect request are provided as kwargs to this method.

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'

title = ''

**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'>]**

### 1.11.4 drf_spectacular.extensions

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

Extension for specifying authentication schemes.

The common use-case usually consists of setting a `name` string and returning a dict from `get_security_definition`. To model a group of headers that go together, set a list of names and return a corresponding list of definitions from `get_security_definition`.

The view class is available via `auto_schema.view`, while the original authentication class can be accessed via `self.target`. If you want to override an included extension, be sure to set a higher matching priority by setting the class attribute `priority = 1` or higher.

get_security_requirement is expected to return a dict with security object names as keys and a scope list as value (usually just []). More than one key in the dict means that each entry is required (AND). If you need alternate variations (OR), return a list of those dicts instead.

`get_security_definition()` is expected to return a valid OpenAPI security scheme object

**abstract get_security_definition**(*auto_schema: AutoSchema*) → Union[dict, List[dict]]

**get_security_requirement**(*auto_schema: AutoSchema*) → Union[Dict[str, List[Any]], List[Dict[str, List[Any]]]]

**name: Union[str, List[str]]**

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

Extension for specifying a list of filter parameters for a given `FilterBackend`.

The original filter class object can be accessed via `self.target`. The attached view is accessible via `auto_schema.view`.

`get_schema_operation_parameters()` is expected to return either an empty list or a list of valid raw OpenAPI parameter objects. Using `drf_spectacular.plumbing.build_parameter_type` is recommended to generate the appropriate raw dict objects.

**abstract get_schema_operation_parameters**(*auto_schema: AutoSchema*, **args*, ***kwargs*) → List[dict]

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

Extension for replacing an insufficient or specifying an unknown Serializer schema.

The existing implementation of `map_serializer()` will generate the same result as *drf-spectacular* would. Either augment or replace the generated schema. The view instance is available via `auto_schema.view`, while the original serializer can be accessed via `self.target`.

`map_serializer()` is expected to return a valid OpenAPI schema object.

**get_name**(*auto_schema: AutoSchema*, *direction: typing_extensions.Literal[request, response]*) → Optional[str]
return str for overriding default name extraction

**map_serializer**(*auto_schema: AutoSchema*, *direction: typing_extensions.Literal[request, response]*)
override for customized serializer mapping

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

Extension for replacing an insufficient or specifying an unknown SerializerField schema.

To augment the default schema, you can get what *drf-spectacular* would generate with `auto_schema._map_serializer_field(self.target, direction, bypass_extensions=True)`. and edit the returned schema at your discretion. Beware that this may still emit warnings, in which case manual construction is advisable.

`map_serializer_field()` is expected to return a valid OpenAPI schema object.

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

**abstract map_serializer_field**(*auto_schema: AutoSchema*, *direction: typing_extensions.Literal[request, response]*)
override for customized serializer field mapping

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

Extension for replacing discovered views with a more schema-appropriate/annotated version.

`view_replacement()` is expected to return a subclass of `APIView` (which includes `ViewSet` et al.). The discovered original view instance can be accessed with `self.target` and be subclassed if desired.

**abstract view_replacement**() → Type[APIView]

### 1.11.5 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.

### 1.11.6 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_callbacks**() → List[*drf_spectacular.utils.OpenApiCallback*]
override this for custom behaviour

**get_description**()
override this for custom behaviour

**get_examples**()
override this for custom behaviour

**get_extensions**() → Dict[str, Any]

**get_external_docs**()
override this for custom behaviour

**get_filter_backends**()
override this for custom behaviour

**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_paginated_name**(*serializer_name*)

**get_request_serializer**() → Any
override this for custom behaviour

**get_response_serializers**() → Any
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*, *bypass_extensions=False*) → drf_spectacular.plumbing.ResolvedComponent

### 1.11.7 drf_spectacular.contrib.django_filters

**class** drf_spectacular.contrib.django_filters.**DjangoFilterExtension**(*target*)
Bases: *drf_spectacular.extensions.OpenApiFilterExtension*

Extensions that specifically deals with `django-filter` fields. The introspection attempts to estimate the underlying model field types to generate filter types.

However, there are under-specified filter fields for which heuristics need to be performed. This serves as an explicit list of all partially-handled filter fields:

- `AllValuesFilter`: skip choices to prevent DB query

- `AllValuesMultipleFilter`: skip choices to prevent DB query, multi handled though
- `ChoiceFilter`: enum handled, type under-specified
- `DateRangeFilter`: N/A
- `LookupChoiceFilter`: N/A
- `ModelChoiceFilter`: enum handled
- `ModelMultipleChoiceFilter`: enum, multi handled
- `MultipleChoiceFilter`: enum, multi handled
- `RangeFilter`: min/max handled, type under-specified
- `TypedChoiceFilter`: enum handled
- `TypedMultipleChoiceFilter`: enum, multi handled

In case of warnings or incorrect filter types, you can manually override the underlying field type with a manual `extend_schema_field` decoration. Alternatively, if you have a filter method for your filter field, you can attach `extend_schema_field` to that filter method.

```
class SomeFilter(FilterSet):
    some_field = extend_schema_field(OpenApiTypes.NUMBER)(
        RangeFilter(field_name='some_manually_annotated_field_in_qs')
    )
```

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## d

# INDEX

## H



## I



## J



## M



## N

## V