Django REST Framework JSON API Documentation

Release 4.1.0

Django REST Framework JSON API contributors

Mar 08, 2021

CONTENTS

1 Getting Started **3**
- 1.1 Requirements . . . 4
- 1.2 Installation . . . 4
- 1.3 Running the example app . . . 4
- 1.4 Running Tests . . . 5

2 Usage **7**
- 2.1 Configuration . . . 7
- 2.2 Pagination . . . 8
- 2.3 Filter Backends . . . 9
- 2.4 Exception handling . . . 11
- 2.5 Performance Testing . . . 12
- 2.6 Serializers . . . 12
- 2.7 Setting the resource_name . . . 12
- 2.8 Inflecting object and relation keys . . . 13
- 2.9 Related fields . . . 16
- 2.10 RelationshipView . . . 20
- 2.11 Working with polymorphic resources . . . 20
- 2.12 Meta . . . 21
- 2.13 Links . . . 22
- 2.14 Included . . . 22
- 2.15 Generating an OpenAPI Specification (OAS) 3.0 schema document . . . 24

3 API Reference **27**
- 3.1 rest_framework_json_api package . . . 27

4 Contributing **41**
- 4.1 Setup development environment . . . 41
- 4.2 For maintainers . . . 42

5 Indices and tables **43**

Python Module Index **45**

Index **47**

Contents:

CHAPTER

ONE

GETTING STARTED

Note: this package is named Django REST Framework JSON API to follow the naming convention of other Django REST Framework packages. Since that's quite a bit to say or type this package will be referred to as DJA elsewhere in these docs.

By default, Django REST Framework produces a response like:

```
{
    "count": 20,
    "next": "http://example.com/api/1.0/identities/?page=3",
    "previous": "http://example.com/api/1.0/identities/?page=1",
    "results": [{
        "id": 3,
        "username": "john",
        "full_name": "John Coltrane"
    }]
}
```

However, for the same `identity` model in JSON API format the response should look like the following:

```
{
    "links": {
        "first": "http://example.com/api/1.0/identities",
        "last": "http://example.com/api/1.0/identities?page=5",
        "next": "http://example.com/api/1.0/identities?page=3",
        "prev": "http://example.com/api/1.0/identities",
    },
    "data": [{
        "type": "identities",
        "id": "3",
        "attributes": {
            "username": "john",
            "full-name": "John Coltrane"
        }
    }],
    "meta": {
        "pagination": {
          "page": "2",
          "pages": "5",
          "count": "20"
        }
    }
}
```

1.1 Requirements

1. Python (3.6, 3.7, 3.8, 3.9)
2. Django (2.2, 3.0, 3.1)
3. Django REST Framework (3.12)

We **highly** recommend and only officially support the latest patch release of each Python, Django and REST Framework series.

Generally Python and Django series are supported till the official end of life. For Django REST Framework the last two series are supported.

1.2 Installation

From PyPI

```
pip install djangorestframework-jsonapi
# for optional package integrations
pip install djangorestframework-jsonapi['django-filter']
pip install djangorestframework-jsonapi['django-polymorphic']
pip install djangorestframework-jsonapi['openapi']
```

From Source

```
git clone https://github.com/django-json-api/django-rest-framework-json-api.git
cd django-rest-framework-json-api && pip install -e .
```

1.3 Running the example app

```
git clone https://github.com/django-json-api/django-rest-framework-json-api.git
cd django-rest-framework-json-api
python3 -m venv env
source env/bin/activate
pip install -Ur requirements.txt
django-admin migrate --settings=example.settings
django-admin loaddata drf_example --settings=example.settings
django-admin runserver --settings=example.settings
```

Browse to

- http://localhost:8000 for the list of available collections (in a non-JSONAPI format!),
- http://localhost:8000/swagger-ui/ for a Swagger user interface to the dynamic schema view, or
- http://localhost:8000/openapi for the schema view's OpenAPI specification document.

1.4 Running Tests

```
pip install tox
tox
```

CHAPTER

TWO

USAGE

The DJA package implements a custom renderer, parser, exception handler, query filter backends, and pagination. To get started enable the pieces in `settings.py` that you want to use.

Many features of the JSON:API format standard have been implemented using Mixin classes in `serializers.py`. The easiest way to make use of those features is to import ModelSerializer variants from `rest_framework_json_api` instead of the usual `rest_framework`

2.1 Configuration

We suggest that you copy the settings block below and modify it if necessary.

```
REST_FRAMEWORK = {
    'PAGE_SIZE': 10,
    'EXCEPTION_HANDLER': 'rest_framework_json_api.exceptions.exception_handler',
    'DEFAULT_PAGINATION_CLASS':
        'rest_framework_json_api.pagination.JsonApiPageNumberPagination',
    'DEFAULT_PARSER_CLASSES': (
        'rest_framework_json_api.parsers.JSONParser',
        'rest_framework.parsers.FormParser',
        'rest_framework.parsers.MultiPartParser'
    ),
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework_json_api.renderers.JSONRenderer',
        # If you're performance testing, you will want to use the browseable API
        # without forms, as the forms can generate their own queries.
        # If performance testing, enable:
        # 'example.utils.BrowsableAPIRendererWithoutForms',
        # Otherwise, to play around with the browseable API, enable:
        'rest_framework_json_api.renderers.BrowsableAPIRenderer'
    ),
    'DEFAULT_METADATA_CLASS': 'rest_framework_json_api.metadata.JSONAPIMetadata',
    'DEFAULT_SCHEMA_CLASS': 'rest_framework_json_api.schemas.openapi.AutoSchema',
    'DEFAULT_FILTER_BACKENDS': (
        'rest_framework_json_api.filters.QueryParameterValidationFilter',
        'rest_framework_json_api.filters.OrderingFilter',
        'rest_framework_json_api.django_filters.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter',
    ),
    'SEARCH_PARAM': 'filter[search]',
    'TEST_REQUEST_RENDERER_CLASSES': (
        'rest_framework_json_api.renderers.JSONRenderer',
    ),
```

(continues on next page)

(continued from previous page)

```
    'TEST_REQUEST_DEFAULT_FORMAT': 'vnd.api+json'
}
```

2.2 Pagination

DJA pagination is based on DRF pagination.

When pagination is enabled, the renderer will return a `meta` object with record count and a `links` object with the next, previous, first, and last links.

Optional query parameters can also be provided to customize the page size or offset limit.

2.2.1 Configuring the Pagination Style

Pagination style can be set on a particular viewset with the `pagination_class` attribute or by default for all viewsets by setting `REST_FRAMEWORK['DEFAULT_PAGINATION_CLASS']` and by setting `REST_FRAMEWORK['PAGE_SIZE']`.

You can configure fixed values for the page size or limit – or allow the client to choose the size or limit via query parameters.

Two pagination classes are available:

- `JsonApiPageNumberPagination` breaks a response up into pages that start at a given page number with a given size (number of items per page). It can be configured with the following attributes:
  - `page_query_param` (default `page[number]`)
  - `page_size_query_param` (default `page[size]`) Set this to `None` if you don't want to allow the client to specify the size.
  - `page_size` (default `REST_FRAMEWORK['PAGE_SIZE']`) default number of items per page unless overridden by `page_size_query_param`.
  - `max_page_size` (default `100`) enforces an upper bound on the `page_size_query_param`. Set it to `None` if you don't want to enforce an upper bound.
- `JsonApiLimitOffsetPagination` breaks a response up into pages that start from an item's offset in the viewset for a given number of items (the limit). It can be configured with the following attributes:
  - `offset_query_param` (default `page[offset]`).
  - `limit_query_param` (default `page[limit]`).
  - `default_limit` (default `REST_FRAMEWORK['PAGE_SIZE']`) is the default number of items per page unless overridden by `limit_query_param`.
  - `max_limit` (default `100`) enforces an upper bound on the limit. Set it to `None` if you don't want to enforce an upper bound.

Examples

These examples show how to configure the parameters to use non-standard names and different limits:

```
from rest_framework_json_api.pagination import JsonApiPageNumberPagination,␣
↪JsonApiLimitOffsetPagination

class MyPagePagination(JsonApiPageNumberPagination):
    page_query_param = 'page_number'
    page_size_query_param = 'page_length'
    page_size = 3
    max_page_size = 1000

class MyLimitPagination(JsonApiLimitOffsetPagination):
    offset_query_param = 'offset'
    limit_query_param = 'limit'
    default_limit = 3
    max_limit = None
```

2.3 Filter Backends

Following are descriptions of JSON:API-specific filter backends and documentation on suggested usage for a standard DRF keyword-search filter backend that makes it consistent with JSON:API.

2.3.1 QueryParameterValidationFilter

`QueryParameterValidationFilter` validates query parameters to be one of the defined JSON:API query parameters (sort, include, filter, fields, page) and returns a `400 Bad Request` if a non-matching query parameter is used. This can help the client identify misspelled query parameters, for example.

If you want to change the list of valid query parameters, override the `.query_regex` attribute:

```
# compiled regex that matches the allowed http://jsonapi.org/format/#query-parameters
# `sort` and `include` stand alone; `filter`, `fields`, and `page` have []'s
query_regex = re.compile(r'^(sort|include)$|^(filter|fields|page)(\[[\w\.\-]+\])?$')
```

For example:

```
import re
from rest_framework_json_api.filters import QueryParameterValidationFilter

class MyQPValidator(QueryParameterValidationFilter):
    query_regex = re.compile(r'^(sort|include|page|page_size)$|^(filter|fields|page)(\
↪[[\w\.\-]+\])?$')
```

If you don't care if non-JSON:API query parameters are allowed (and potentially silently ignored), simply don't use this filter backend.

2.3.2 OrderingFilter

`OrderingFilter` implements the JSON:API `sort` and uses DRF's ordering filter.

Per the JSON:API specification, "If the server does not support sorting as specified in the query parameter `sort`, it **MUST** return `400 Bad Request`." For example, for `?sort=abc,foo,def` where `foo` is a valid field name and the other two are not valid:

```
{
    "errors": [
        {
            "detail": "invalid sort parameters: abc,def",
            "source": {
                "pointer": "/data"
            },
            "status": "400"
        }
    ]
}
```

If you want to silently ignore bad sort fields, just use `rest_framework.filters.OrderingFilter` and set `ordering_param` to `sort`.

2.3.3 DjangoFilterBackend

`DjangoFilterBackend` implements a Django ORM-style JSON:API `filter` using the django-filter package.

This filter is not part of the JSON:API standard per-se, other than the requirement to use the `filter` keyword: It is an optional implementation of a style of filtering in which each filter is an ORM expression as implemented by `DjangoFilterBackend` and seems to be in alignment with an interpretation of the JSON:API *recommendations*, including relationship chaining.

Filters can be:

- A resource field equality test: `?filter[qty]=123`
- Apply other field lookup operators: `?filter[name.icontains]=bar` or `?filter[name.isnull]=true`
- Membership in a list of values: `?filter[name.in]=abc,123,zzz (name in ['abc','123','zzz'])`
- Filters can be combined for intersection (AND): `?filter[qty]=123&filter[name.in]=abc,123,zzz&filter[...]` or `?filter[authors.id]=1&filter[authors.id]=2`
- A related resource path can be used: `?filter[inventory.item.partNum]=123456` (where `inventory.item` is the relationship path)

The filter returns a `400 Bad Request` error for invalid filter query parameters as in this example for `GET http://127.0.0.1:8000/nopage-entries?filter[bad]=1`:

```
{
    "errors": [
        {
            "detail": "invalid filter[bad]",
            "source": {
                "pointer": "/data"
            },
            "status": "400"
```

(continues on next page)

(continued from previous page)

```
        }
    ]
}
```

As this feature depends on `django-filter` you need to run

```
pip install djangorestframework-jsonapi['django-filter']
```

2.3.4 SearchFilter

To comply with JSON:API query parameter naming standards, DRF's SearchFilter should be configured to use a `filter[_something_]` query parameter. This can be done by default by adding the SearchFilter to `REST_FRAMEWORK['DEFAULT_FILTER_BACKENDS']` and setting `REST_FRAMEWORK['SEARCH_PARAM']` or adding the `.search_param` attribute to a custom class derived from `SearchFilter`. If you do this and also use *`DjangoFilterBackend`*, make sure you set the same values for both classes.

2.3.5 Configuring Filter Backends

You can configure the filter backends either by setting the `REST_FRAMEWORK['DEFAULT_FILTER_BACKENDS']` as shown in the *example settings* or individually add them as `.filter_backends` View attributes:

```
from rest_framework_json_api import filters
from rest_framework_json_api import django_filters
from rest_framework import SearchFilter
from models import MyModel

class MyViewset(ModelViewSet):
   queryset = MyModel.objects.all()
   serializer_class = MyModelSerializer
   filter_backends = (filters.QueryParameterValidationFilter, filters.OrderingFilter,
                      django_filters.DjangoFilterBackend, SearchFilter)
   filterset_fields = {
       'id': ('exact', 'lt', 'gt', 'gte', 'lte', 'in'),
       'descriptuon': ('icontains', 'iexact', 'contains'),
       'tagline': ('icontains', 'iexact', 'contains'),
   }
   search_fields = ('id', 'description', 'tagline',)
```

2.4 Exception handling

For the `exception_handler` class, if the optional `JSON_API_UNIFORM_EXCEPTIONS` is set to True, all exceptions will respond with the JSON API error format.

When `JSON_API_UNIFORM_EXCEPTIONS` is False (the default), non-JSON API views will respond with the normal DRF error format.

2.5 Performance Testing

If you are trying to see if your viewsets are configured properly to optimize performance, it is preferable to use `example.utils.BrowsableAPIRendererWithoutForms` instead of the default `BrowsableAPIRenderer` to remove queries introduced by the forms themselves.

2.6 Serializers

It is recommended to import the base serializer classes from this package rather than from vanilla DRF. For example,

```
from rest_framework_json_api import serializers

class MyModelSerializer(serializers.ModelSerializer):
    # ...
```

2.7 Setting the resource_name

You may manually set the `resource_name` property on views, serializers, or models to specify the `type` key in the json output. In the case of setting the `resource_name` property for models you must include the property inside a `JSONAPIMeta` class on the model. It is automatically set for you as the plural of the view or model name except on resources that do not subclass `rest_framework.viewsets.ModelViewSet`:

Example - `resource_name` on View:

```
class Me(generics.GenericAPIView):
    """
    Current user's identity endpoint.

    GET /me
    """
    resource_name = 'users'
    serializer_class = identity_serializers.IdentitySerializer
    allowed_methods = ['GET']
    permission_classes = (permissions.IsAuthenticated, )
```

If you set the `resource_name` property on the object to `False` the data will be returned without modification.

Example - `resource_name` on Model:

```
class Me(models.Model):
    """
    A simple model
    """
    name = models.CharField(max_length=100)

    class JSONAPIMeta:
        resource_name = "users"
```

If you set the `resource_name` on a combination of model, serializer, or view in the same hierarchy, the name will be resolved as following: view > serializer > model. (Ex: A view `resource_name` will always override a `resource_name` specified on a serializer or model). Setting the `resource_name` on the view should be used sparingly as serializers and models are shared between multiple endpoints. Setting the `resource_name` on views may result in a different `type` being set depending on which endpoint the resource is fetched from.

2.8 Inflecting object and relation keys

This package includes the ability (off by default) to automatically convert json api field names of requests and responses from the python/rest_framework's preferred underscore to a format of your choice. To hook this up include the following setting in your project settings:

```
JSON_API_FORMAT_FIELD_NAMES = 'dasherize'
```

Possible values:

- dasherize
- camelize (first letter is lowercase)
- capitalize (camelize but with first letter uppercase)
- underscore

Note: due to the way the inflector works `address_1` can camelize to `address1` on output but it cannot convert `address1` back to `address_1` on POST or PATCH. Keep this in mind when naming fields with numbers in them.

Example - Without format conversion:

```
{
    "data": [{
        "type": "identities",
        "id": "3",
        "attributes": {
            "username": "john",
            "first_name": "John",
            "last_name": "Coltrane",
            "full_name": "John Coltrane"
        },
    }],
    "meta": {
        "pagination": {
          "count": 20
        }
    }
}
```

Example - With format conversion set to `dasherize`:

```
{
    "data": [{
        "type": "identities",
        "id": "3",
        "attributes": {
            "username": "john",
            "first-name": "John",
            "last-name": "Coltrane",
            "full-name": "John Coltrane"
        },
    }],
    "meta": {
        "pagination": {
          "count": 20
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

2.8.1 Types

A similar option to `JSON_API_FORMAT_FIELD_NAMES` can be set for the types:

```
JSON_API_FORMAT_TYPES = 'dasherize'
```

Example without format conversion:

```
{
        "data": [{
        "type": "blog_identity",
        "id": "3",
        "attributes": {
                ...
        },
        "relationships": {
            "home_town": {
                "data": [{
                    "type": "home_town",
                    "id": 3
                }]
            }
        }
    }]
}
```

When set to dasherize:

```
{
        "data": [{
        "type": "blog-identity",
        "id": "3",
        "attributes": {
                ...
        },
        "relationships": {
            "home_town": {
                "data": [{
                    "type": "home-town",
                    "id": 3
                }]
            }
        }
    }]
}
```

It is also possible to pluralize the types like so:

```
JSON_API_PLURALIZE_TYPES = True
```

Example without pluralization:

```
{
        "data": [{
        "type": "identity",
        "id": "3",
        "attributes": {
                ...
        },
        "relationships": {
            "home_towns": {
                "data": [{
                    "type": "home_town",
                    "id": "3"
                }]
            }
        }
    }]
}
```

When set to pluralize:

```
{
        "data": [{
        "type": "identities",
        "id": "3",
        "attributes": {
                ...
        },
        "relationships": {
            "home_towns": {
                "data": [{
                    "type": "home_towns",
                    "id": "3"
                }]
            }
        }
    }]
}
```

2.8.2 Related URL segments

Serializer properties in relationship and related resource URLs may be infected using the `JSON_API_FORMAT_RELATED_LINKS` setting.

```
JSON_API_FORMAT_RELATED_LINKS = 'dasherize'
```

For example, with a serializer property `created_by` and with `'dasherize'` formatting:

```
{
  "data": {
      "type": "comments",
      "id": "1",
      "attributes": {
          "text": "Comments are fun!"
      },
      "links": {
```

(continues on next page)

(continued from previous page)

```
            "self": "/comments/1"
        },
        "relationships": {
          "created_by": {
            "links": {
              "self": "/comments/1/relationships/created-by",
              "related": "/comments/1/created-by"
            }
          }
        }
    },
    "links": {
        "self": "/comments/1"
    }
}
```

The relationship name is formatted by the `JSON_API_FORMAT_FIELD_NAMES` setting, but the URL segments are formatted by the `JSON_API_FORMAT_RELATED_LINKS` setting.

2.9 Related fields

2.9.1 ResourceRelatedField

Because of the additional structure needed to represent relationships in JSON API, this package provides the `ResourceRelatedField` for serializers, which works similarly to `PrimaryKeyRelatedField`. By default, `rest_framework_json_api.serializers.ModelSerializer` will use this for related fields automatically. It can be instantiated explicitly as in the following example:

```
from rest_framework_json_api import serializers
from rest_framework_json_api.relations import ResourceRelatedField

from myapp.models import Order, LineItem, Customer


class OrderSerializer(serializers.ModelSerializer):
    class Meta:
        model = Order

    line_items = ResourceRelatedField(
        queryset=LineItem.objects,
        many=True  # necessary for M2M fields & reverse FK fields
    )

    customer = ResourceRelatedField(
        queryset=Customer.objects  # queryset argument is required
    )                              # except when read_only=True
```

In the JSON:API spec, relationship objects contain links to related objects. To make this work on a serializer we need to tell the `ResourceRelatedField` about the corresponding view. Use the `HyperlinkedModelSerializer` and instantiate the `ResourceRelatedField` with the relevant keyword arguments:

```
from rest_framework_json_api import serializers
from rest_framework_json_api.relations import ResourceRelatedField
```

(continues on next page)

(continued from previous page)

```
from myapp.models import Order, LineItem, Customer


class OrderSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Order

    line_items = ResourceRelatedField(
        queryset=LineItem.objects,
        many=True,
        related_link_view_name='order-lineitems-list',
        related_link_url_kwarg='order_pk',
        self_link_view_name='order_relationships'
    )

    customer = ResourceRelatedField(
        queryset=Customer.objects,
        related_link_view_name='order-customer-detail',
        related_link_url_kwarg='order_pk',
        self_link_view_name='order-relationships'
    )
```

- `related_link_view_name` is the name of the route for the related view.
- `related_link_url_kwarg` is the keyword argument that will be passed to the view that identifies the 'parent' object, so that the results can be filtered to show only those objects related to the 'parent'.
- `self_link_view_name` is the name of the route for the `RelationshipView` (see below).

In this example, `reverse('order-lineitems-list', kwargs={'order_pk': 3}` should resolve to something like `/orders/3/lineitems`, and that route should instantiate a view or viewset for `LineItem` objects that accepts a keyword argument `order_pk`. The drf-nested-routers package is useful for defining such nested routes in your urlconf.

The corresponding viewset for the `line-items-list` route in the above example might look like the following. Note that in the typical use case this would be the same viewset used for the `/lineitems` endpoints; when accessed through the nested route `/orders/<order_pk>/lineitems` the queryset is filtered using the `order_pk` keyword argument to include only the lineitems related to the specified order.

```
from rest_framework import viewsets

from myapp.models import LineItem
from myapp.serializers import LineItemSerializer


class LineItemViewSet(viewsets.ModelViewSet):
    queryset = LineItem.objects
    serializer_class = LineItemSerializer

    def get_queryset(self):
        queryset = super(LineItemViewSet, self).get_queryset()

        # if this viewset is accessed via the 'order-lineitems-list' route,
        # it wll have been passed the `order_pk` kwarg and the queryset
        # needs to be filtered accordingly; if it was accessed via the
        # unnested '/lineitems' route, the queryset should include all LineItems
```

(continues on next page)

(continued from previous page)

```
        if 'order_pk' in self.kwargs:
            order_pk = self.kwargs['order_pk']
            queryset = queryset.filter(order__pk=order_pk)

        return queryset
```

2.9.2 HyperlinkedRelatedField

`relations.HyperlinkedRelatedField` has same functionality as `ResourceRelatedField` but does not render `data`. Use this in case you only need links of relationships and want to lower payload and increase performance.

2.9.3 SerializerMethodResourceRelatedField

`relations.SerializerMethodResourceRelatedField` combines behaviour of DRF `SerializerMethodField` and `ResourceRelatedField`, so it accepts `method_name` together with `model` and links-related arguments. `data` is rendered in `ResourceRelatedField` manner.

```
from rest_framework_json_api import serializers
from rest_framework_json_api.relations import SerializerMethodResourceRelatedField

from myapp.models import Order, LineItem


class OrderSerializer(serializers.ModelSerializer):
    class Meta:
        model = Order

    line_items = SerializerMethodResourceRelatedField(
        model=LineItem,
        many=True,
        method_name='get_big_line_items'
    )

    small_line_items = SerializerMethodResourceRelatedField(
        model=LineItem,
        many=True,
        # default to method_name='get_small_line_items'
    )

    def get_big_line_items(self, instance):
        return LineItem.objects.filter(order=instance).filter(amount__gt=1000)

    def get_small_line_items(self, instance):
        return LineItem.objects.filter(order=instance).filter(amount__lte=1000)
```

or using `related_link_*` with `HyperlinkedModelSerializer`

```
class OrderSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Order

    line_items = SerializerMethodResourceRelatedField(
```

(continues on next page)

(continued from previous page)

```
        model=LineItem,
        many=True,
        method_name='get_big_line_items',
        related_link_view_name='order-lineitems-list',
        related_link_url_kwarg='order_pk',
    )

    def get_big_line_items(self, instance):
        return LineItem.objects.filter(order=instance).filter(amount__gt=1000)
```

2.9.4 Related urls

There is a nice way to handle "related" urls like `/orders/3/lineitems/` or `/orders/3/customer/`. All you need is just add to `urls.py`:

```
url(r'^orders/(?P<pk>[^/.]+)/$',
        OrderViewSet.as_view({'get': 'retrieve'}),
        name='order-detail'),
url(r'^orders/(?P<pk>[^/.]+)/(?P<related_field>\w+)/$',
        OrderViewSet.as_view({'get': 'retrieve_related'}),
        name='order-related'),
```

Make sure that RelatedField declaration has `related_link_url_kwarg='pk'` or simply skipped (will be set by default):

```
    line_items = ResourceRelatedField(
        queryset=LineItem.objects,
        many=True,
        related_link_view_name='order-related',
        related_link_url_kwarg='pk',
        self_link_view_name='order-relationships'
    )

    customer = ResourceRelatedField(
        queryset=Customer.objects,
        related_link_view_name='order-related',
        self_link_view_name='order-relationships'
    )
```

And, the most important part - declare serializer for each related entity:

```
class OrderSerializer(serializers.HyperlinkedModelSerializer):
    ...
    related_serializers = {
        'customer': 'example.serializers.CustomerSerializer',
        'line_items': 'example.serializers.LineItemSerializer'
    }
```

Or, if you already have `included_serializers` declared and your `related_serializers` look the same, just skip it:

```
class OrderSerializer(serializers.HyperlinkedModelSerializer):
    ...
    included_serializers = {
```

(continues on next page)

(continued from previous page)

```
        'customer': 'example.serializers.CustomerSerializer',
        'line_items': 'example.serializers.LineItemSerializer'
    }
```

2.10 RelationshipView

`rest_framework_json_api.views.RelationshipView` is used to build relationship views (see the JSON:API spec). The `self` link on a relationship object should point to the corresponding relationship view.

The relationship view is fairly simple because it only serializes Resource Identifier Objects rather than full resource objects. In most cases the following is sufficient:

```
from rest_framework_json_api.views import RelationshipView

from myapp.models import Order


class OrderRelationshipView(RelationshipView):
    queryset = Order.objects
```

The urlconf would need to contain a route like the following:

```
url(
    regex=r'^orders/(?P<pk>[^/.]+)/relationships/(?P<related_field>[^/.]+)$',
    view=OrderRelationshipView.as_view(),
    name='order-relationships'
)
```

The `related_field` kwarg specifies which relationship to use, so if we are interested in the relationship represented by the related model field `Order.line_items` on the Order with pk 3, the url would be `/orders/3/relationships/line_items`. On `HyperlinkedModelSerializer`, the `ResourceRelatedField` will construct the url based on the provided `self_link_view_name` keyword argument, which should match the `name=` provided in the urlconf, and will use the name of the field for the `related_field` kwarg. Also we can override `related_field` in the url. Let's say we want the url to be: `/order/3/relationships/order_items` - all we need to do is just add `field_name_mapping` dict to the class:

```
field_name_mapping = {
        'order_items': 'line_items'
    }
```

2.11 Working with polymorphic resources

Polymorphic resources allow you to use specialized subclasses without requiring special endpoints to expose the specialized versions. For example, if you had a `Project` that could be either an `ArtProject` or a `ResearchProject`, you can have both kinds at the same URL.

DJA tests its polymorphic support against django-polymorphic. The polymorphic feature should also work with other popular libraries like django-polymodels or django-typed-models.

As this feature depends on `django-polymorphic` you need to run

```
pip install djangorestframework-jsonapi['django-polymorphic']
```

2.11.1 Writing polymorphic resources

A polymorphic endpoint can be set up if associated with a polymorphic serializer. A polymorphic serializer takes care of (de)serializing the correct instances types and can be defined like this:

```
class ProjectSerializer(serializers.PolymorphicModelSerializer):
    polymorphic_serializers = [ArtProjectSerializer, ResearchProjectSerializer]

    class Meta:
        model = models.Project
```

It must inherit from `serializers.PolymorphicModelSerializer` and define the `polymorphic_serializers` list. This attribute defines the accepted resource types.

Polymorphic relations can also be handled with `relations.PolymorphicResourceRelatedField` like this:

```
class CompanySerializer(serializers.ModelSerializer):
    current_project = relations.PolymorphicResourceRelatedField(
        ProjectSerializer, queryset=models.Project.objects.all())
    future_projects = relations.PolymorphicResourceRelatedField(
        ProjectSerializer, queryset=models.Project.objects.all(), many=True)

    class Meta:
        model = models.Company
```

They must be explicitly declared with the `polymorphic_serializer` (first positional argument) correctly defined. It must be a subclass of `serializers.PolymorphicModelSerializer`.

2.12 Meta

You may add metadata to the rendered json in two different ways: `meta_fields` and `get_root_meta`.

On any `rest_framework_json_api.serializers.ModelSerializer` you may add a `meta_fields` property to the `Meta` class. This behaves in the same manner as the default `fields` property and will cause `SerializerMethodFields` or model values to be added to the `meta` object within the same `data` as the serializer.

To add metadata to the top level `meta` object add:

```
def get_root_meta(self, resource, many):
    if many:
      # Dealing with a list request
      return {
          'size': len(resource)
      }
    else:
      # Dealing with a detail request
      return {
        'foo': 'bar'
      }
```

to the serializer. It must return a dict and will be merged with the existing top level `meta`.

To access metadata in incoming requests, the `JSONParser` will add the metadata under a top level `_meta` key in the parsed data dictionary. For instance, to access meta data from a `serializer` object, you may use `serializer.initial_data.get("_meta")`. To customize the `_meta` key, see *here*.

2.13 Links

Adding `url` to `fields` on a serializer will add a `self` link to the `links` key.

Related links will be created automatically when using the Relationship View.

2.14 Included

JSON API can include additional resources in a single network request. The specification refers to this feature as Compound Documents. Compound Documents can reduce the number of network requests which can lead to a better performing web application. To accomplish this, the specification permits a top level `included` key. The list of content within this key are the extra resources that are related to the primary resource.

To make a Compound Document, you need to modify your `ModelSerializer`. `included_serializers` is required to inform DJA of what and how you would like to include. `included_resources` tells DJA what you want to include by default.

For example, suppose you are making an app to go on quests, and you would like to fetch your chosen knight along with the quest. You could accomplish that with:

```
class KnightSerializer(serializers.ModelSerializer):
    class Meta:
        model = Knight
        fields = ('id', 'name', 'strength', 'dexterity', 'charisma')


class QuestSerializer(serializers.ModelSerializer):
    included_serializers = {
        'knight': KnightSerializer,
    }

    class Meta:
        model = Quest
        fields = ('id', 'title', 'reward', 'knight')

    class JSONAPIMeta:
        included_resources = ['knight']
```

2.14.1 Performance improvements

Be aware that using included resources without any form of prefetching **WILL HURT PERFORMANCE** as it will introduce m*(n+1) queries.

A viewset helper was therefore designed to automatically preload data when possible. Such is automatically available when subclassing `ModelViewSet`.

It also allows to define custom `select_related` and `prefetch_related` for each requested `include` when needed in special cases:

`rest_framework_json_api.views.ModelViewSet`:

```
from rest_framework_json_api import views

# When MyViewSet is called with ?include=author it will dynamically prefetch author␣
↪and author.bio
class MyViewSet(views.ModelViewSet):
    queryset = Book.objects.all()
    select_for_includes = {
        'author': ['author__bio'],
    }
    prefetch_for_includes = {
        '__all__': [],
        'all_authors': [Prefetch('all_authors', queryset=Author.objects.select_
↪related('bio'))],
        'category.section': ['category']
    }
```

An additional convenience DJA class exists for read-only views, just as it does in DRF.

```
from rest_framework_json_api import views

class MyReadOnlyViewSet(views.ReadOnlyModelViewSet):
    # ...
```

The special keyword `__all__` can be used to specify a prefetch which should be done regardless of the include, similar to making the prefetch yourself on the QuerySet.

Using the helper to prefetch, rather than attempting to minimise queries via `select_related` might give you better performance depending on the characteristics of your data and database.

For example:

If you have a single model, e.g. Book, which has four relations e.g. Author, Publisher, CopyrightHolder, Category.

To display 25 books and related models, you would need to either do:

a) 1 query via selected_related, e.g. SELECT * FROM books LEFT JOIN author LEFT JOIN publisher LEFT JOIN CopyrightHolder LEFT JOIN Category

b) 4 small queries via prefetch_related.

If you have 1M books, 50k authors, 10k categories, 10k copyrightholders in the `select_related` scenario, you've just created a in-memory table with 1e18 rows which will likely exhaust any available memory and slow your database to crawl.

The `prefetch_related` case will issue 4 queries, but they will be small and fast queries.

2.15 Generating an OpenAPI Specification (OAS) 3.0 schema document

DRF >= 3.12 has a new OAS schema functionality to generate an OAS 3.0 schema as a YAML or JSON file.

DJA extends DRF's schema support to generate an OAS schema in the JSON:API format.

2.15.1 AutoSchema Settings

In order to produce an OAS schema that properly represents the JSON:API structure you have to either add a `schema` attribute to each view class or set the `REST_FRAMEWORK['DEFAULT_SCHEMA_CLASS']` to DJA's version of AutoSchema.

View-based

```
from rest_framework_json_api.schemas.openapi import AutoSchema

class MyViewset(ModelViewSet):
    schema = AutoSchema
    ...
```

Default schema class

```
REST_FRAMEWORK = {
    # ...
    'DEFAULT_SCHEMA_CLASS': 'rest_framework_json_api.schemas.openapi.AutoSchema',
}
```

2.15.2 Adding additional OAS schema content

You can extend the OAS schema document by subclassing `SchemaGenerator` and extending `get_schema`.

Here's an example that adds OAS `info` and `servers` objects.

```
from rest_framework_json_api.schemas.openapi import SchemaGenerator as␣
↪JSONAPISchemaGenerator


class MySchemaGenerator(JSONAPISchemaGenerator):
    """
    Describe my OAS schema info in detail (overriding what DRF put in) and list the␣
↪servers where it can be found.
    """
    def get_schema(self, request, public):
        schema = super().get_schema(request, public)
        schema['info'] = {
            'version': '1.0',
            'title': 'my demo API',
            'description': 'A demonstration of [OAS 3.0](https://www.openapis.org)',
            'contact': {
                'name': 'my name'
```

(continues on next page)

(continued from previous page)

```
            },
            'license': {
                'name': 'BSD 2 clause',
                'url': 'https://github.com/django-json-api/django-rest-framework-json-
↪api/blob/master/LICENSE',
            }
        }
        schema['servers'] = [
            {'url': 'https://localhost/v1', 'description': 'local docker'},
            {'url': 'http://localhost:8000/v1', 'description': 'local dev'},
            {'url': 'https://api.example.com/v1', 'description': 'demo server'},
            {'url': '{serverURL}', 'description': 'provide your server URL',
             'variables': {'serverURL': {'default': 'http://localhost:8000/v1'}}}
        ]
        return schema
```

2.15.3 Generate a Static Schema on Command Line

See DRF documentation for generateschema To generate an OAS schema document, use something like:

```
$ django-admin generateschema --settings=example.settings \
                              --generator_class myapp.views.MySchemaGenerator >
↪myschema.yaml
```

You can then use any number of OAS tools such as swagger-ui-watcher to render the schema:

```
$ swagger-ui-watcher myschema.yaml
```

Note: Swagger-ui-watcher will complain that "DELETE operations cannot have a requestBody" but it will still work. This error in the OAS specification will be fixed when OAS 3.1.0 is published.

(swagger-ui will work silently.)

2.15.4 Generate a Dynamic Schema in a View

See DRF documentation for a Dynamic Schema.

```
from rest_framework.schemas import get_schema_view

urlpatterns = [
    ...
    path('openapi', get_schema_view(
        title="Example API",
        description="API for all things ...",
        version="1.0.0",
        generator_class=MySchemaGenerator,
    ), name='openapi-schema'),
    path('swagger-ui/', TemplateView.as_view(
        template_name='swagger-ui.html',
        extra_context={'schema_url': 'openapi-schema'}
    ), name='swagger-ui'),
    ...
]
```

CHAPTER

THREE

API REFERENCE

This API reference is autogenerated from the Python docstrings – which need to be improved!

3.1 rest_framework_json_api package

3.1.1 Subpackages

rest_framework_json_api.django_filters package

Submodules

rest_framework_json_api.django_filters.backends module

class rest_framework_json_api.django_filters.backends.**DjangoFilterBackend**

Bases: django_filters.rest_framework.backends.DjangoFilterBackend

A Django-style ORM filter implementation, using *django-filter*.

This is not part of the jsonapi standard per-se, other than the requirement to use the *filter* keyword: This is an optional implementation of style of filtering in which each filter is an ORM expression as implemented by DjangoFilterBackend and seems to be in alignment with an interpretation of http://jsonapi.org/recommendations/#filtering, including relationship chaining. It also returns a 400 error for invalid filters.

Filters can be:

- A resource field equality test:

  `?filter[qty]=123`

- Apply other https://docs.djangoproject.com/en/stable/ref/models/querysets/#field-lookups operators:

  `?filter[name.icontains]=bar` or `?filter[name.isnull]=true...`

- Membership in a list of values:

  `?filter[name.in]=abc,123,zzz` (name in ['abc','123','zzz'])

- Filters can be combined for intersection (AND):

  `?filter[qty]=123&filter[name.in]=abc,123,zzz&filter[...]`

- A related resource path can be used:

  `?filter[inventory.item.partNum]=123456` (where *inventory.item* is the relationship path)

If you are also using rest_framework.filters.SearchFilter you'll want to customize the name of the query parameter for searching to make sure it doesn't conflict with a field name defined in the filterset. The recommended value is: *search_param="filter[search]"* but just make sure it's *filter[<something>]* to comply with the jsonapi spec requirement to use the filter keyword. The default is "search" unless overriden but it's used here just to make sure we don't complain about it being an invalid filter.

search_param = 'filter[search]'

filter_regex = re.compile('^filter(?P<ldelim>\\[?)(?P<assoc>[\\w\\.\\-]*)(?P<rdelim>

get_filterset(*request*, *queryset*, *view*)
Sometimes there's no *filterset_class* defined yet the client still requests a filter. Make sure they see an error too. This means we have to *get_filterset_kwargs()* even if there's no *filterset_class*.

get_filterset_kwargs(*request*, *queryset*, *view*)
Turns filter[<field>]=<value> into <field>=<value> which is what DjangoFilterBackend expects

Raises **ValidationError** – for bad filter syntax

get_schema_operation_parameters(*view*)
Convert backend filter *name* to JSON:API-style *filter[name]*. For filters that are relationship paths, rewrite ORM-style __ to our preferred .. For example: *blog__name__contains* becomes *filter[blog.name.contains]*.

This is basically the reverse of *get_filterset_kwargs* above.

rest_framework_json_api.schemas package

Submodules

rest_framework_json_api.schemas.openapi module

class rest_framework_json_api.schemas.openapi.**AutoSchema**(*tags=None*, *operation_id_base=None*, *component_name=None*)
Bases: rest_framework.schemas.openapi.AutoSchema

Extend DRF's openapi.AutoSchema for JSONAPI serialization.

content_types = ['application/vnd.api+json']

get_operation(*path*, *method*)
JSONAPI adds some standard fields to the API response that are not in upstream DRF: - some that only apply to GET/HEAD methods. - collections - special handling for POST, PATCH, DELETE

get_operation_id(*path*, *method*)
The upstream DRF version creates non-unique operationIDs, because the same view is used for the main path as well as such as related and relationships. This concatenates the (mapped) method name and path as the spec allows most any

get_request_body(*path*, *method*)
A request body is required by jsonapi for POST, PATCH, and DELETE methods.

map_serializer(*serializer*)
Custom map_serializer that serializes the schema using the jsonapi spec. Non-attributes like related and identity fields, are move to 'relationships' and 'links'.

class rest_framework_json_api.schemas.openapi.**SchemaGenerator**(*title=None*, *url=None*, *description=None*, *patterns=None*, *urlconf=None*, *version=None*)

Bases: rest_framework.schemas.openapi.SchemaGenerator

Extend DRF's SchemaGenerator to implement jsonapi-flavored generateschema command.

get_schema(*request=None*, *public=False*)

Generate a JSONAPI OpenAPI schema. Overrides upstream DRF's get_schema.

jsonapi_components = {'parameters': {'fields': {'description': '[sparse fieldsets](

3.1.2 Submodules

rest_framework_json_api.exceptions module

rest_framework_json_api.exceptions.**rendered_with_json_api**(*view*)

rest_framework_json_api.exceptions.**exception_handler**(*exc*, *context*)

exception rest_framework_json_api.exceptions.**Conflict**(*detail=None*, *code=None*)

Bases: rest_framework.exceptions.APIException

status_code = 409

default_detail = 'Conflict.'

rest_framework_json_api.filters module

class rest_framework_json_api.filters.**OrderingFilter**

Bases: rest_framework.filters.OrderingFilter

A backend filter that implements http://jsonapi.org/format/#fetching-sorting and raises a 400 error if any sort field is invalid.

If you prefer *not* to report 400 errors for invalid sort fields, just use rest_framework.filters.OrderingFilter with ordering_param = "sort"

Also applies DJA format_value() to convert (e.g. camelcase) to underscore. (See JSON_API_FORMAT_FIELD_NAMES in docs/usage.md)

ordering_param = 'sort'

override rest_framework.filters.OrderingFilter.ordering_param with JSON:API-compliant query parameter name.

remove_invalid_fields(*queryset*, *fields*, *view*, *request*)

Extend rest_framework.filters.OrderingFilter.remove_invalid_fields() to validate that all provided sort fields exist (as contrasted with the super's behavior which is to silently remove invalid fields).

Raises **ValidationError** – if a sort field is invalid.

class rest_framework_json_api.filters.**QueryParameterValidationFilter**

Bases: rest_framework.filters.BaseFilterBackend

A backend filter that performs strict validation of query parameters for JSON:API spec conformance and raises a 400 error if non-conforming usage is found.

If you want to add some additional non-standard query parameters, override *query_regex* adding the new parameters. Make sure to comply with the rules at http://jsonapi.org/format/#query-parameters.

query_regex = re.compile('^(sort|include)$|^(?P<type>filter|fields|page)(\\[[\\w\\.\\-
compiled regex that matches the allowed http://jsonapi.org/format/#query-parameters: *sort* and *include* stand alone; *filter*, *fields*, and *page* have []'s

validate_query_params(*request*)
Validate that query params are in the list of valid query keywords in *query_regex*

Raises **ValidationError** – if not.

filter_queryset(*request*, *queryset*, *view*)
Overrides `BaseFilterBackend.filter_queryset()` by first validating the query params with *validate_query_params()*

rest_framework_json_api.metadata module

class rest_framework_json_api.metadata.**JSONAPIMetadata**
Bases: `rest_framework.metadata.SimpleMetadata`

This is the JSON:API metadata implementation. It returns an ad-hoc set of information about the view. There are not any formalized standards for *OPTIONS* responses for us to base this on.

type_lookup = <rest_framework.utils.field_mapping.ClassLookupDict object>

relation_type_lookup = <rest_framework.utils.field_mapping.ClassLookupDict object>

determine_metadata(*request*, *view*)

get_serializer_info(*serializer*)
Given an instance of a serializer, return a dictionary of metadata about its fields.

get_field_info(*field*)
Given an instance of a serializer field, return a dictionary of metadata about it.

rest_framework_json_api.pagination module

Pagination fields

class rest_framework_json_api.pagination.**JsonApiPageNumberPagination**
Bases: `rest_framework.pagination.PageNumberPagination`

A json-api compatible pagination format.

page_query_param = 'page[number]'

page_size_query_param = 'page[size]'

max_page_size = 100

build_link(*index*)

get_paginated_response(*data*)

class rest_framework_json_api.pagination.**JsonApiLimitOffsetPagination**
Bases: `rest_framework.pagination.LimitOffsetPagination`

A limit/offset based style. For example:

```
http://api.example.org/accounts/?page[limit]=100
http://api.example.org/accounts/?page[offset]=400&page[limit]=100
```

limit_query_param = 'page[limit]'

offset_query_param = 'page[offset]'

max_limit = 100

get_last_link()

get_first_link()

get_paginated_response(*data*)

rest_framework_json_api.parsers module

Parsers

class rest_framework_json_api.parsers.**JSONParser**

Bases: rest_framework.parsers.JSONParser

Similar to *JSONRenderer*, the *JSONParser* you may override the following methods if you need highly custom parsing control.

A JSON API client will send a payload that looks like this:

```
{
    "data": {
        "type": "identities",
        "id": 1,
        "attributes": {
            "first_name": "John",
            "last_name": "Coltrane"
        }
    }
}
```

We extract the attributes so that DRF serializers can work as normal.

media_type = 'application/vnd.api+json'

renderer_class

alias of *rest_framework_json_api.renderers.JSONRenderer*

static parse_attributes(*data*)

static parse_relationships(*data*)

static parse_metadata(*result*)

Returns a dictionary which will be merged into parsed data of the request. By default, it reads the *meta* content in the request body and returns it in a dictionary with a *_meta* top level key.

parse(*stream*, *media_type=None*, *parser_context=None*)

Parses the incoming bytestream as JSON and returns the resulting data

rest_framework_json_api.relations module

class rest_framework_json_api.relations.**SkipDataMixin**(**args*, ***kwargs*)
Bases: object

This workaround skips "data" rendering for relationships in order to save some sql queries and improve performance

get_attribute(*instance*)

to_representation(**args*)

class rest_framework_json_api.relations.**ManyRelatedFieldWithNoData**(**args*, ***kwargs*)
Bases: *rest_framework_json_api.relations.SkipDataMixin*, rest_framework.relations.ManyRelatedField

class rest_framework_json_api.relations.**HyperlinkedMixin**(*self_link_view_name=None*, *related_link_view_name=None*, ***kwargs*)
Bases: object

self_link_view_name = None

related_link_view_name = None

related_link_lookup_field = 'pk'

get_url(*name*, *view_name*, *kwargs*, *request*)
Given a name, view name and kwargs, return the URL that hyperlinks to the object.

May raise a *NoReverseMatch* if the *view_name* and *lookup_field* attributes are not configured to correctly match the URL conf.

get_links(*obj=None*, *lookup_field='pk'*)

class rest_framework_json_api.relations.**HyperlinkedRelatedField**(**args*, ***kwargs*)
Bases: *rest_framework_json_api.relations.HyperlinkedMixin*, *rest_framework_json_api.relations.SkipDataMixin*, rest_framework.relations.RelatedField

classmethod many_init(**args*, ***kwargs*)
This method handles creating a parent *ManyRelatedField* instance when the *many=True* keyword argument is passed.

Typically you won't need to override this method.

Note that we're over-cautious in passing most arguments to both parent and child classes in order to try to cover the general case. If you're overriding this method you'll probably want something much simpler, eg:

```
@classmethod
def many_init(cls, *args, **kwargs):
    kwargs['child'] = cls()
    return CustomManyRelatedField(*args, **kwargs)
```

class rest_framework_json_api.relations.**ResourceRelatedField**(**args*, ***kwargs*)
Bases: *rest_framework_json_api.relations.HyperlinkedMixin*, rest_framework.relations.PrimaryKeyRelatedField

self_link_view_name = None

related_link_view_name = None

related_link_lookup_field = 'pk'

default_error_messages = {'does_not_exist': 'Invalid pk "{pk_value}" - object does not

use_pk_only_optimization()

conflict(*key*, ***kwargs*)
A helper method that simply raises a validation error.

to_internal_value(*data*)

to_representation(*value*)

get_resource_type_from_included_serializer()
Check to see it this resource has a different resource_name when included and return that name, or None

get_parent_serializer()

is_serializer(*candidate*)

get_choices(*cutoff=None*)

class rest_framework_json_api.relations.**PolymorphicResourceRelatedField**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.relations.ResourceRelatedField*

Inform DRF that the relation must be considered polymorphic. Takes a *polymorphic_serializer* as the first positional argument to retrieve then validate the accepted types set.

default_error_messages = {'does_not_exist': 'Invalid pk "{pk_value}" - object does not

use_pk_only_optimization()

to_internal_value(*data*)

class rest_framework_json_api.relations.**SerializerMethodFieldBase**(**args*, ***kwargs*)

Bases: rest_framework.fields.Field

bind(*field_name*, *parent*)

get_attribute(*instance*)

class rest_framework_json_api.relations.**ManySerializerMethodResourceRelatedField**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.relations.SerializerMethodFieldBase*, *rest_framework_json_api.relations.ResourceRelatedField*

to_representation(*value*)

class rest_framework_json_api.relations.**SerializerMethodResourceRelatedField**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.relations.SerializerMethodFieldBase*, *rest_framework_json_api.relations.ResourceRelatedField*

Allows us to use serializer method RelatedFields with return querysets

many_kwargs = ['read_only', 'write_only', 'required', 'default', 'initial', 'source',

many_cls
alias of *rest_framework_json_api.relations.ManySerializerMethodResourceRelatedField*

classmethod many_init(**args*, ***kwargs*)

class rest_framework_json_api.relations.**ManySerializerMethodHyperlinkedRelatedField**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.relations.SkipDataMixin*, *rest_framework_json_api.relations.ManySerializerMethodResourceRelatedField*

class rest_framework_json_api.relations.**SerializerMethodHyperlinkedRelatedField**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.relations.SkipDataMixin*, *rest_framework_json_api.relations.SerializerMethodResourceRelatedField*

many_cls

alias of *rest_framework_json_api.relations.ManySerializerMethodHyperlinkedRelatedField*

rest_framework_json_api.renderers module

Renderers

class rest_framework_json_api.renderers.**JSONRenderer**

Bases: rest_framework.renderers.JSONRenderer

The *JSONRenderer* exposes a number of methods that you may override if you need highly custom rendering control.

Render a JSON response per the JSON API spec:

```
{
  "data": [
    {
      "type": "companies",
      "id": "1",
      "attributes": {
        "name": "Mozilla",
        "slug": "mozilla",
        "date-created": "2014-03-13 16:33:37"
      }
    }
  ]
}
```

media_type = 'application/vnd.api+json'

format = 'vnd.api+json'

classmethod extract_attributes(*fields*, *resource*)

Builds the *attributes* object of the JSON API resource object.

classmethod extract_relationships(*fields*, *resource*, *resource_instance*)

Builds the relationships top level object based on related serializers.

classmethod extract_relation_instance(*field*, *resource_instance*)

Determines what instance represents given relation and extracts it.

Relation instance is determined exactly same way as it determined in parent serializer

classmethod extract_included(*fields*, *resource*, *resource_instance*, *included_resources*, *included_cache*)

Adds related data to the top level included key when the request includes ?include=example,example_field2

classmethod extract_meta(*serializer*, *resource*)
Gathers the data from serializer fields specified in meta_fields and adds it to the meta object.

classmethod extract_root_meta(*serializer*, *resource*)
Calls a *get_root_meta* function on a serializer, if it exists.

classmethod build_json_resource_obj(*fields*, *resource*, *resource_instance*, *resource_name*, *serializer*, *force_type_resolution=False*)
Builds the resource object (type, id, attributes) and extracts relationships.

render_relationship_view(*data*, *accepted_media_type=None*, *renderer_context=None*)

render_errors(*data*, *accepted_media_type=None*, *renderer_context=None*)

render(*data*, *accepted_media_type=None*, *renderer_context=None*)

class rest_framework_json_api.renderers.**BrowsableAPIRenderer**
Bases: rest_framework.renderers.BrowsableAPIRenderer

template = 'rest_framework_json_api/api.html'

includes_template = 'rest_framework_json_api/includes.html'

get_context(*data*, *accepted_media_type*, *renderer_context*)

get_includes_form(*view*)

rest_framework_json_api.serializers module

class rest_framework_json_api.serializers.**ResourceIdentifierObjectSerializer**(**args*, ***kwargs*)
Bases: rest_framework.serializers.BaseSerializer

default_error_messages = {'does_not_exist': 'Invalid pk "{pk_value}" - object does not

model_class = None

to_representation(*instance*)

to_internal_value(*data*)

class rest_framework_json_api.serializers.**SparseFieldsetsMixin**(**args*, ***kwargs*)
Bases: object

A serializer mixin that adds support for sparse fieldsets through *fields* query parameter.

Specification: https://jsonapi.org/format/#fetching-sparse-fieldsets

class rest_framework_json_api.serializers.**IncludedResourcesValidationMixin**(**args*, ***kwargs*)
Bases: object

A serializer mixin that adds validation of *include* query parameter to support compound documents.

Specification: https://jsonapi.org/format/#document-compound-documents)

class rest_framework_json_api.serializers.**SerializerMetaclass**(*name*, *bases*, *attrs*)
Bases: rest_framework.serializers.SerializerMetaclass

class rest_framework_json_api.serializers.**Serializer**(**args*, ***kwargs*)
Bases: *rest_framework_json_api.serializers.IncludedResourcesValidationMixin*, *rest_framework_json_api.serializers.SparseFieldsetsMixin*, rest_framework.serializers.Serializer

A *Serializer* is a model-less serializer class with additional support for json:api spec features.

As in json:api specification a type is always required you need to make sure that you define *resource_name* in your *Meta* class when deriving from this class.

Included Mixins:

- A mixin class to enable sparse fieldsets is included
- A mixin class to enable validation of included resources is included

class rest_framework_json_api.serializers.**HyperlinkedModelSerializer**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.serializers.IncludedResourcesValidationMixin*, *rest_framework_json_api.serializers.SparseFieldsetsMixin*, rest_framework.serializers.HyperlinkedModelSerializer

A type of *ModelSerializer* that uses hyperlinked relationships instead of primary key relationships. Specifically:

- A 'url' field is included instead of the 'id' field.
- Relationships to other instances are hyperlinks, instead of primary keys.

Included Mixins:

- A mixin class to enable sparse fieldsets is included
- A mixin class to enable validation of included resources is included

class rest_framework_json_api.serializers.**ModelSerializer**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.serializers.IncludedResourcesValidationMixin*, *rest_framework_json_api.serializers.SparseFieldsetsMixin*, rest_framework.serializers.ModelSerializer

A *ModelSerializer* is just a regular *Serializer*, except that:

- A set of default fields are automatically populated.
- A set of default validators are automatically populated.
- Default *.create()* and *.update()* implementations are provided.

The process of automatically determining a set of serializer fields based on the model fields is reasonably complex, but you almost certainly don't need to dig into the implementation.

If the *ModelSerializer* class *doesn't* generate the set of fields that you need you should either declare the extra/differing fields explicitly on the serializer class, or simply use a *Serializer* class.

Included Mixins:

- A mixin class to enable sparse fieldsets is included
- A mixin class to enable validation of included resources is included

serializer_related_field

alias of *rest_framework_json_api.relations.ResourceRelatedField*

get_field_names(*declared_fields*, *info*)

We override the parent to omit explicity defined meta fields (such as SerializerMethodFields) from the list of declared fields

class rest_framework_json_api.serializers.**PolymorphicSerializerMetaclass**(*name*, *bases*, *attrs*)

Bases: *rest_framework_json_api.serializers.SerializerMetaclass*

This metaclass ensures that the *polymorphic_serializers* is correctly defined on a *PolymorphicSerializer* class and make a cache of model/serializer/type mappings.

class rest_framework_json_api.serializers.**PolymorphicModelSerializer**(**args*, ***kwargs*)

Bases: *rest_framework_json_api.serializers.ModelSerializer*

A serializer for polymorphic models. Useful for "lazy" parent models. Leaves should be represented with a regular serializer.

get_fields()
Return an exhaustive list of the polymorphic serializer fields.

classmethod get_polymorphic_serializer_for_instance(*instance*)
Return the polymorphic serializer associated with the given instance/model. Raise *NotImplementedError* if no serializer is found for the given model. This usually means that a serializer is missing in the class's *polymorphic_serializers* attribute.

classmethod get_polymorphic_model_for_serializer(*serializer*)
Return the polymorphic model associated with the given serializer. Raise *NotImplementedError* if no model is found for the given serializer. This usually means that a serializer is missing in the class's *polymorphic_serializers* attribute.

classmethod get_polymorphic_serializer_for_type(*obj_type*)
Return the polymorphic serializer associated with the given type. Raise *NotImplementedError* if no serializer is found for the given type. This usually means that a serializer is missing in the class's *polymorphic_serializers* attribute.

classmethod get_polymorphic_model_for_type(*obj_type*)
Return the polymorphic model associated with the given type. Raise *NotImplementedError* if no model is found for the given type. This usually means that a serializer is missing in the class's *polymorphic_serializers* attribute.

classmethod get_polymorphic_types()
Return the list of accepted types.

to_representation(*instance*)
Retrieve the appropriate polymorphic serializer and use this to handle representation.

to_internal_value(*data*)
Ensure that the given type is one of the expected polymorphic types, then retrieve the appropriate polymorphic serializer and use this to handle internal value.

rest_framework_json_api.settings module

This module provides the *json_api_settings* object that is used to access JSON API REST framework settings, checking for user settings first, then falling back to the defaults.

class rest_framework_json_api.settings.**JSONAPISettings**(*user_settings=<LazySettings "example.settings">*, *defaults={'FORMAT_FIELD_NAMES': False, 'FORMAT_RELATED_LINKS': False, 'FORMAT_TYPES': False, 'PLURALIZE_TYPES': False, 'UNIFORM_EXCEPTIONS': False}*)

Bases: object

A settings object that allows json api settings to be access as properties.

rest_framework_json_api.settings.**reload_json_api_settings**(**args*, ***kwargs*)

rest_framework_json_api.utils module

rest_framework_json_api.utils.**get_resource_name**(*context*, *expand_polymorphic_types=False*)
Return the name of a resource.

rest_framework_json_api.utils.**get_serializer_fields**(*serializer*)

rest_framework_json_api.utils.**format_field_names**(*obj*, *format_type=None*)
Takes a dict and returns it with formatted keys as set in *format_type* or *JSON_API_FORMAT_FIELD_NAMES*

Format_type Either 'dasherize', 'camelize', 'capitalize' or 'underscore'

rest_framework_json_api.utils.**format_value**(*value*, *format_type=None*)

rest_framework_json_api.utils.**format_resource_type**(*value*, *format_type=None*, *pluralize=None*)

rest_framework_json_api.utils.**format_link_segment**(*value*, *format_type=None*)
Takes a string value and returns it with formatted keys as set in *format_type* or *JSON_API_FORMAT_RELATED_LINKS*.

Format_type Either 'dasherize', 'camelize', 'capitalize' or 'underscore'

rest_framework_json_api.utils.**get_related_resource_type**(*relation*)

rest_framework_json_api.utils.**get_resource_type_from_model**(*model*)

rest_framework_json_api.utils.**get_resource_type_from_queryset**(*qs*)

rest_framework_json_api.utils.**get_resource_type_from_instance**(*instance*)

rest_framework_json_api.utils.**get_resource_type_from_manager**(*manager*)

rest_framework_json_api.utils.**get_resource_type_from_serializer**(*serializer*)

rest_framework_json_api.utils.**get_included_resources**(*request*, *serializer=None*)
Build a list of included resources.

rest_framework_json_api.utils.**get_default_included_resources_from_serializer**(*serializer*)

rest_framework_json_api.utils.**get_included_serializers**(*serializer*)

rest_framework_json_api.utils.**get_relation_instance**(*resource_instance*, *source*, *serializer*)

class rest_framework_json_api.utils.**Hyperlink**(*url*, *name*)
Bases: str

A string like object that additionally has an associated name. We use this for hyperlinked URLs that may render as a named link in some contexts, or render as a plain URL in others.

Comes from Django REST framework 3.2 https://github.com/tomchristie/django-rest-framework

is_hyperlink = True

rest_framework_json_api.utils.**format_drf_errors**(*response*, *context*, *exc*)

rest_framework_json_api.utils.**format_error_object**(*message*, *pointer*, *response*)

rest_framework_json_api.utils.**format_errors**(*data*)

rest_framework_json_api.views module

class rest_framework_json_api.views.**PreloadIncludesMixin**
Bases: object

This mixin provides a helper attributes to select or prefetch related models based on the include specified in the URL.

__all__ can be used to specify a prefetch which should be done regardless of the include

```
# When MyViewSet is called with ?include=author it will prefetch author and␣
↪authorbio
class MyViewSet(viewsets.ModelViewSet):
    queryset = Book.objects.all()
    prefetch_for_includes = {
        '__all__': [],
        'category.section': ['category']
    }
    select_for_includes = {
        '__all__': [],
        'author': ['author', 'author__authorbio'],
    }
```

get_select_related(*include*)

get_prefetch_related(*include*)

get_queryset(**args*, ***kwargs*)

class rest_framework_json_api.views.**AutoPrefetchMixin**
Bases: object

get_queryset(**args*, ***kwargs*)
This mixin adds automatic prefetching for OneToOne and ManyToMany fields.

class rest_framework_json_api.views.**RelatedMixin**
Bases: object

This mixin handles all related entities, whose Serializers are declared in "related_serializers"

retrieve_related(*request*, **args*, ***kwargs*)

get_related_serializer(*instance*, ***kwargs*)

get_related_serializer_class()

get_related_field_name()

get_related_instance()

class rest_framework_json_api.views.**ModelViewSet**(***kwargs*)
Bases: *rest_framework_json_api.views.AutoPrefetchMixin*, *rest_framework_json_api.views.PreloadIncludesMixin*, *rest_framework_json_api.views.RelatedMixin*, rest_framework.viewsets.ModelViewSet

http_method_names = ['get', 'post', 'patch', 'delete', 'head', 'options']

class rest_framework_json_api.views.**ReadOnlyModelViewSet**(***kwargs*)
Bases: *rest_framework_json_api.views.AutoPrefetchMixin*, *rest_framework_json_api.views.RelatedMixin*, rest_framework.viewsets.ReadOnlyModelViewSet

http_method_names = ['get', 'post', 'patch', 'delete', 'head', 'options']

class rest_framework_json_api.views.**RelationshipView**(***kwargs*)

Bases: rest_framework.generics.GenericAPIView

serializer_class

alias of *rest_framework_json_api.serializers.ResourceIdentifierObjectSerializer*

self_link_view_name = None

related_link_view_name = None

field_name_mapping = {}

http_method_names = ['get', 'post', 'patch', 'delete', 'head', 'options']

get_serializer_class()

get_url(*name*, *view_name*, *kwargs*, *request*)

Given a name, view name and kwargs, return the URL that hyperlinks to the object.

May raise a *NoReverseMatch* if the *view_name* and *lookup_field* attributes are not configured to correctly match the URL conf.

get_links()

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

remove_relationships(*instance_manager*, *field*)

patch(*request*, **args*, ***kwargs*)

post(*request*, **args*, ***kwargs*)

delete(*request*, **args*, ***kwargs*)

get_related_instance()

get_related_field_name()

get_resource_name()

set_resource_name(*value*)

property resource_name

CHAPTER

FOUR

CONTRIBUTING

Django REST Framework JSON API (aka DJA) should be easy to contribute to. If anything is unclear about how to contribute, please submit an issue on GitHub so that we can fix it!

Before writing any code, have a conversation on a GitHub issue to see if the proposed change makes sense for the project.

4.1 Setup development environment

4.1.1 Clone

To start developing on Django REST Framework JSON API you need to first clone the repository:

```
git clone https://github.com/django-json-api/django-rest-framework-json-api.git
```

4.1.2 Testing

To run tests clone the repository, and then:

```
# Setup the virtual environment
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt

# Format code
black .

# Run linting
flake8

# Run tests
pytest
```

4.1.3 Running against multiple environments

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
```

4.1.4 Setup pre-commit

pre-commit hooks is an additional option to check linting and formatting of code independent of an editor before you commit your changes with git.

To setup pre-commit hooks first create a testing environment as explained above before running below commands:

```
pip install pre-commit
pre-commit install
```

4.2 For maintainers

To upload a release (using version 1.2.3 as the example) first setup testing environment as above before running below commands:

```
python setup.py sdist bdist_wheel
twine upload dist/*
git tag -a v1.2.3 -m 'Release 1.2.3'
git push --tags
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

r

rest_framework_json_api, 27
rest_framework_json_api.django_filters, 27
rest_framework_json_api.django_filters.backends, 27
rest_framework_json_api.exceptions, 29
rest_framework_json_api.filters, 29
rest_framework_json_api.metadata, 30
rest_framework_json_api.pagination, 30
rest_framework_json_api.parsers, 31
rest_framework_json_api.relations, 32
rest_framework_json_api.renderers, 34
rest_framework_json_api.schemas, 28
rest_framework_json_api.schemas.openapi, 28
rest_framework_json_api.serializers, 35
rest_framework_json_api.settings, 37
rest_framework_json_api.utils, 38
rest_framework_json_api.views, 39

INDEX

A

AutoPrefetchMixin (*class in rest_framework_json_api.views*), 39
AutoSchema (*class in rest_framework_json_api.schemas.openapi*), 28

B

bind() (*rest_framework_json_api.relations.SerializerMethodFieldBase method*), 33
BrowsableAPIRenderer (*class in rest_framework_json_api.renderers*), 35
build_json_resource_obj() (*rest_framework_json_api.renderers.JSONRenderer class method*), 35
build_link() (*rest_framework_json_api.pagination.JsonApiPageNumberPagination method*), 30

C

Conflict, 29
conflict() (*rest_framework_json_api.relations.ResourceRelatedField method*), 33
content_types (*rest_framework_json_api.schemas.openapi.AutoSchema attribute*), 28

D

default_detail (*rest_framework_json_api.exceptions.Conflict attribute*), 29
default_error_messages (*rest_framework_json_api.relations.PolymorphicResourceRelatedField attribute*), 33
default_error_messages (*rest_framework_json_api.relations.ResourceRelatedField attribute*), 33
default_error_messages (*rest_framework_json_api.serializers.ResourceIdentifierObjectSerializer attribute*), 35
delete() (*rest_framework_json_api.views.RelationshipView method*), 40
determine_metadata() (*rest_framework_json_api.metadata.JSONAPIMetadata method*), 30
DjangoFilterBackend (*class in rest_framework_json_api.django_filters.backends*), 27

E

exception_handler() (*in module rest_framework_json_api.exceptions*), 29
extract_attributes() (*rest_framework_json_api.renderers.JSONRenderer class method*), 34
extract_included() (*rest_framework_json_api.renderers.JSONRenderer class method*), 34
extract_meta() (*rest_framework_json_api.renderers.JSONRenderer class method*), 34
extract_relation_instance() (*rest_framework_json_api.renderers.JSONRenderer class method*), 34
extract_relationships() (*rest_framework_json_api.renderers.JSONRenderer class method*), 34
extract_root_meta() (*rest_framework_json_api.renderers.JSONRenderer class method*), 35

F

field_name_mapping (*rest_framework_json_api.views.RelationshipView attribute*), 40
filter_queryset() (*rest_framework_json_api.filters.QueryParameterValidationFilter method*), 30
filter_regex (*rest_framework_json_api.django_filters.backends.DjangoFilterBackend attribute*), 28
format (*rest_framework_json_api.renderers.JSONRenderer attribute*), 34
format_drf_errors() (*in module rest_framework_json_api.utils*), 38
format_error_object() (*in module rest_framework_json_api.utils*), 38
format_errors() (*in module rest_framework_json_api.utils*), 38

format_field_names() (*in module rest_framework_json_api.utils*), 38
format_link_segment() (*in module rest_framework_json_api.utils*), 38
format_resource_type() (*in module rest_framework_json_api.utils*), 38
format_value() (*in module rest_framework_json_api.utils*), 38

G

get() (*rest_framework_json_api.views.RelationshipView method*), 40
get_attribute() (*rest_framework_json_api.relations.SerializerMethodFieldBase method*), 33
get_attribute() (*rest_framework_json_api.relations.SkipDataMixin method*), 32
get_choices() (*rest_framework_json_api.relations.ResourceRelatedField method*), 33
get_context() (*rest_framework_json_api.renderers.BrowsableAPIRenderer method*), 35
get_default_included_resources_from_serializer() (*in module rest_framework_json_api.utils*), 38
get_field_info() (*rest_framework_json_api.metadata.JSONAPIMetadata method*), 30
get_field_names() (*rest_framework_json_api.serializers.ModelSerializer method*), 36
get_fields() (*rest_framework_json_api.serializers.PolymorphicModelSerializer method*), 37
get_filterset() (*rest_framework_json_api.django_filters.backends.DjangoFilterBackend method*), 28
get_filterset_kwargs() (*rest_framework_json_api.django_filters.backends.DjangoFilterBackend method*), 28
get_first_link() (*rest_framework_json_api.pagination.JsonApiLimitOffsetPagination method*), 31
get_included_resources() (*in module rest_framework_json_api.utils*), 38
get_included_serializers() (*in module rest_framework_json_api.utils*), 38
get_includes_form() (*rest_framework_json_api.renderers.BrowsableAPIRenderer method*), 35
get_last_link() (*rest_framework_json_api.pagination.JsonApiLimitOffsetPagination method*), 31
get_links() (*rest_framework_json_api.relations.HyperlinkedMixin method*), 32
get_links() (*rest_framework_json_api.views.RelationshipView method*), 40
get_operation() (*rest_framework_json_api.schemas.openapi.AutoSchema method*), 28
get_operation_id() (*rest_framework_json_api.schemas.openapi.AutoSchema method*), 28
get_paginated_response() (*rest_framework_json_api.pagination.JsonApiLimitOffsetPagination method*), 31
get_paginated_response() (*rest_framework_json_api.pagination.JsonApiPageNumberPagination method*), 30
get_parent_serializer() (*rest_framework_json_api.relations.ResourceRelatedField method*), 33
get_polymorphic_model_for_serializer() (*rest_framework_json_api.serializers.PolymorphicModelSerializer class method*), 37
get_polymorphic_model_for_type() (*rest_framework_json_api.serializers.PolymorphicModelSerializer class method*), 37
get_polymorphic_serializer_for_instance() (*rest_framework_json_api.serializers.PolymorphicModelSerializer class method*), 37
get_polymorphic_serializer_for_type() (*rest_framework_json_api.serializers.PolymorphicModelSerializer class method*), 37
get_polymorphic_types() (*rest_framework_json_api.serializers.PolymorphicModelSerializer class method*), 37
get_prefetch_related() (*rest_framework_json_api.views.PreloadIncludesMixin method*), 39
get_queryset() (*rest_framework_json_api.views.AutoPrefetchMixin method*), 39
get_queryset() (*rest_framework_json_api.views.PreloadIncludesMixin method*), 39
get_related_field_name() (*rest_framework_json_api.views.RelatedMixin method*), 39
get_related_field_name() (*rest_framework_json_api.views.RelationshipView method*), 40
get_related_instance() (*rest_framework_json_api.views.RelatedMixin method*), 39
get_related_instance() (*rest_framework_json_api.views.RelationshipView method*), 40
get_related_resource_type() (*in module rest_framework_json_api.utils*), 38
get_related_serializer() (*rest_framework_json_api.views.RelatedMixin method*), 39
get_related_serializer_class() (*rest_framework_json_api.views.RelatedMixin method*), 39
get_relation_instance() (*in module rest_framework_json_api.utils*), 38
get_request_body()

(*rest_framework_json_api.schemas.openapi.AutoSchema method*), 28
get_resource_name() (*in module rest_framework_json_api.utils*), 38
get_resource_name() (*rest_framework_json_api.views.RelationshipView method*), 40
get_resource_type_from_included_serializer() (*rest_framework_json_api.relations.ResourceRelatedField method*), 33
get_resource_type_from_instance() (*in module rest_framework_json_api.utils*), 38
get_resource_type_from_manager() (*in module rest_framework_json_api.utils*), 38
get_resource_type_from_model() (*in module rest_framework_json_api.utils*), 38
get_resource_type_from_queryset() (*in module rest_framework_json_api.utils*), 38
get_resource_type_from_serializer() (*in module rest_framework_json_api.utils*), 38
get_schema() (*rest_framework_json_api.schemas.openapi.SchemaGenerator method*), 29
get_schema_operation_parameters() (*rest_framework_json_api.django_filters.backends.DjangoFilterBackend method*), 28
get_select_related() (*rest_framework_json_api.views.PreloadIncludesMixin method*), 39
get_serializer_class() (*rest_framework_json_api.views.RelationshipView method*), 40
get_serializer_fields() (*in module rest_framework_json_api.utils*), 38
get_serializer_info() (*rest_framework_json_api.metadata.JSONAPIMetadata method*), 30
get_url() (*rest_framework_json_api.relations.HyperlinkedMixin method*), 32
get_url() (*rest_framework_json_api.views.RelationshipView method*), 40

H

http_method_names (*rest_framework_json_api.views.ModelViewSet attribute*), 39
http_method_names (*rest_framework_json_api.views.ReadOnlyModelViewSet attribute*), 39
http_method_names (*rest_framework_json_api.views.RelationshipView attribute*), 40
Hyperlink (*class in rest_framework_json_api.utils*), 38
HyperlinkedMixin (*class in rest_framework_json_api.relations*), 32
HyperlinkedModelSerializer (*class in rest_framework_json_api.serializers*), 36
HyperlinkedRelatedField (*class in rest_framework_json_api.relations*), 32

I

IncludedResourcesValidationMixin (*class in rest_framework_json_api.serializers*), 35
includes_template (*rest_framework_json_api.renderers.BrowsableAPIRenderer attribute*), 35
is_hyperlink (*rest_framework_json_api.utils.Hyperlink attribute*), 38
is_serializer() (*rest_framework_json_api.relations.ResourceRelated method*), 33

J

jsonapi_components (*rest_framework_json_api.schemas.openapi.SchemaGenerator attribute*), 29
JsonApiLimitOffsetPagination (*class in rest_framework_json_api.pagination*), 30
JSONAPIMetadata (*class in rest_framework_json_api.metadata*), 30
JsonApiPageNumberPagination (*class in rest_framework_json_api.pagination*), 30
JSONAPISettings (*class in rest_framework_json_api.settings*), 37
JSONParser (*class in rest_framework_json_api.parsers*), 31
JSONRenderer (*class in rest_framework_json_api.renderers*), 34

L

limit_query_param (*rest_framework_json_api.pagination.JsonApiLimitOffsetPaginat attribute*), 30

M

many_cls (*rest_framework_json_api.relations.SerializerMethodHyperlink attribute*), 34
many_cls (*rest_framework_json_api.relations.SerializerMethodResource attribute*), 33
many_init() (*rest_framework_json_api.relations.HyperlinkedRelatedFi class method*), 32
many_init() (*rest_framework_json_api.relations.SerializerMethodReso class method*), 33
many_kwargs (*rest_framework_json_api.relations.SerializerMethodReso attribute*), 33
ManyRelatedFieldWithNoData (*class in rest_framework_json_api.relations*), 32

ManySerializerMethodHyperlinkedRelatedField (*class in rest_framework_json_api.relations*), 33
ManySerializerMethodResourceRelatedField (*class in rest_framework_json_api.relations*), 33
map_serializer() (*rest_framework_json_api.schemas.openapi.AutoSchema method*), 28
max_limit (*rest_framework_json_api.pagination.JsonApiLimitOffsetPagination attribute*), 31
max_page_size (*rest_framework_json_api.pagination.JsonApiPageNumberPagination attribute*), 30
media_type (*rest_framework_json_api.parsers.JSONParser attribute*), 31
media_type (*rest_framework_json_api.renderers.JSONRenderer attribute*), 34
model_class (*rest_framework_json_api.serializers.ResourceIdentifierObjectSerializer attribute*), 35
ModelSerializer (*class in rest_framework_json_api.serializers*), 36
ModelViewSet (*class in rest_framework_json_api.views*), 39
module
rest_framework_json_api, 27
rest_framework_json_api.django_filters, 27
rest_framework_json_api.django_filters.backends, 27
rest_framework_json_api.exceptions, 29
rest_framework_json_api.filters, 29
rest_framework_json_api.metadata, 30
rest_framework_json_api.pagination, 30
rest_framework_json_api.parsers, 31
rest_framework_json_api.relations, 32
rest_framework_json_api.renderers, 34
rest_framework_json_api.schemas, 28
rest_framework_json_api.schemas.openapi, 28
rest_framework_json_api.serializers, 35
rest_framework_json_api.settings, 37
rest_framework_json_api.utils, 38
rest_framework_json_api.views, 39

O

offset_query_param (*rest_framework_json_api.pagination.JsonApiLimitOffsetPagination attribute*), 31
ordering_param (*rest_framework_json_api.filters.OrderingFilter attribute*), 29
OrderingFilter (*class in rest_framework_json_api.filters*), 29

P

page_query_param (*rest_framework_json_api.pagination.JsonApiPageNumberPagination attribute*), 30
page_size_query_param (*rest_framework_json_api.pagination.JsonApiPageNumberPagination attribute*), 30
parse() (*rest_framework_json_api.parsers.JSONParser method*), 31
parse_attributes() (*rest_framework_json_api.parsers.JSONParser static method*), 31
parse_metadata() (*rest_framework_json_api.parsers.JSONParser static method*), 31
parse_relationships() (*rest_framework_json_api.parsers.JSONParser static method*), 31
patch() (*rest_framework_json_api.views.RelationshipView method*), 40
PolymorphicModelSerializer (*class in rest_framework_json_api.serializers*), 37
PolymorphicResourceRelatedField (*class in rest_framework_json_api.relations*), 33
PolymorphicSerializerMetaclass (*class in rest_framework_json_api.serializers*), 36
post() (*rest_framework_json_api.views.RelationshipView method*), 40
PreloadIncludesMixin (*class in rest_framework_json_api.views*), 39

Q

query_regex (*rest_framework_json_api.filters.QueryParameterValidationFilter attribute*), 30
QueryParameterValidationFilter (*class in rest_framework_json_api.filters*), 29

R

ReadOnlyModelViewSet (*class in rest_framework_json_api.views*), 39
related_link_lookup_field (*rest_framework_json_api.relations.HyperlinkedMixin attribute*), 32
related_link_lookup_field (*rest_framework_json_api.relations.ResourceRelatedField attribute*), 33
related_link_view_name (*rest_framework_json_api.relations.HyperlinkedMixin attribute*), 32
related_link_view_name (*rest_framework_json_api.relations.ResourceRelatedField attribute*), 32

related_link_view_name (*rest_framework_json_api.views.RelationshipView attribute*), 40
RelatedMixin (*class in rest_framework_json_api.views*), 39
relation_type_lookup (*rest_framework_json_api.metadata.JSONAPIMetadata attribute*), 30
RelationshipView (*class in rest_framework_json_api.views*), 39
reload_json_api_settings() (*in module rest_framework_json_api.settings*), 38
remove_invalid_fields() (*rest_framework_json_api.filters.OrderingFilter method*), 29
remove_relationships() (*rest_framework_json_api.views.RelationshipView method*), 40
render() (*rest_framework_json_api.renderers.JSONRenderer method*), 35
render_errors() (*rest_framework_json_api.renderers.JSONRenderer method*), 35
render_relationship_view() (*rest_framework_json_api.renderers.JSONRenderer method*), 35
rendered_with_json_api() (*in module rest_framework_json_api.exceptions*), 29
renderer_class (*rest_framework_json_api.parsers.JSONParser attribute*), 31
resource_name() (*rest_framework_json_api.views.RelationshipView property*), 40
ResourceIdentifierObjectSerializer (*class in rest_framework_json_api.serializers*), 35
ResourceRelatedField (*class in rest_framework_json_api.relations*), 32
rest_framework_json_api
    module, 27
rest_framework_json_api.django_filters
    module, 27
rest_framework_json_api.django_filters.backends
    module, 27
rest_framework_json_api.exceptions
    module, 29
rest_framework_json_api.filters
    module, 29
rest_framework_json_api.metadata
    module, 30
rest_framework_json_api.pagination
    module, 30
rest_framework_json_api.parsers
    module, 31
rest_framework_json_api.relations
    module, 32
rest_framework_json_api.renderers
    module, 34
rest_framework_json_api.schemas
    module, 28
rest_framework_json_api.schemas.openapi
    module, 28
rest_framework_json_api.serializers
    module, 35
rest_framework_json_api.settings
    module, 37
rest_framework_json_api.utils
    module, 38
rest_framework_json_api.views
    module, 39
retrieve_related() (*rest_framework_json_api.views.RelatedMixin method*), 39

S

SchemaGenerator (*class in rest_framework_json_api.schemas.openapi*), 28
search_param (*rest_framework_json_api.django_filters.backends.Djang attribute*), 28
self_link_view_name (*rest_framework_json_api.relations.HyperlinkedMixin attribute*), 32
self_link_view_name (*rest_framework_json_api.relations.ResourceRelatedField attribute*), 32
self_link_view_name (*rest_framework_json_api.views.RelationshipView attribute*), 40
Serializer (*class in rest_framework_json_api.serializers*), 35
serializer_class (*rest_framework_json_api.views.RelationshipView attribute*), 40
serializer_related_field (*rest_framework_json_api.serializers.ModelSerializer attribute*), 36
SerializerMetaclass (*class in rest_framework_json_api.serializers*), 35
SerializerMethodFieldBase (*class in rest_framework_json_api.relations*), 33
SerializerMethodHyperlinkedRelatedField (*class in rest_framework_json_api.relations*), 34
SerializerMethodResourceRelatedField (*class in rest_framework_json_api.relations*), 33
set_resource_name() (*rest_framework_json_api.views.RelationshipView method*), 40
SkipDataMixin (*class in rest_framework_json_api.relations*), 32

SparseFieldsetsMixin (*class in rest_framework_json_api.serializers*), 35
status_code (*rest_framework_json_api.exceptions.Conflict attribute*), 29

T

template (*rest_framework_json_api.renderers.BrowsableAPIRenderer attribute*), 35
to_internal_value() (*rest_framework_json_api.relations.PolymorphicResourceRelatedField method*), 33
to_internal_value() (*rest_framework_json_api.relations.ResourceRelatedField method*), 33
to_internal_value() (*rest_framework_json_api.serializers.PolymorphicModelSerializer method*), 37
to_internal_value() (*rest_framework_json_api.serializers.ResourceIdentifierObjectSerializer method*), 35
to_representation() (*rest_framework_json_api.relations.ManySerializerMethodResourceRelatedField method*), 33
to_representation() (*rest_framework_json_api.relations.ResourceRelatedField method*), 33
to_representation() (*rest_framework_json_api.relations.SkipDataMixin method*), 32
to_representation() (*rest_framework_json_api.serializers.PolymorphicModelSerializer method*), 37
to_representation() (*rest_framework_json_api.serializers.ResourceIdentifierObjectSerializer method*), 35
type_lookup (*rest_framework_json_api.metadata.JSONAPIMetadata attribute*), 30

U

use_pk_only_optimization() (*rest_framework_json_api.relations.PolymorphicResourceRelatedField method*), 33
use_pk_only_optimization() (*rest_framework_json_api.relations.ResourceRelatedField method*), 33

V

validate_query_params() (*rest_framework_json_api.filters.QueryParameterValidationFilter method*), 30