

---

# **audoma - API Automatic Documentation Maker**

***Release 0.1***

**Iteo**

**Sep 20, 2022**



## CONTENTS:

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Python version . . . . .	3
1.2	Dependencies . . . . .	3
1.3	Install Audoma . . . . .	4
1.4	Audoma initial configuration . . . . .	4
<b>2</b>	<b>Usage</b>	<b>5</b>
2.1	ViewSet defined serializers . . . . .	5
2.2	Permissions . . . . .	12
2.3	Custom choices . . . . .	14
2.4	Filters . . . . .	15
2.5	Validators . . . . .	17
2.6	Decorators . . . . .	18
2.7	Examples . . . . .	30
2.8	Extra Fields . . . . .	31
2.9	Serializer Field links . . . . .	33
2.10	Schema Extensions . . . . .	33
<b>3</b>	<b>Contributing to audoma</b>	<b>35</b>
3.1	Issues . . . . .	35
3.2	Pull requests . . . . .	35
3.3	Running unit tests . . . . .	36
3.4	Example application . . . . .	36
<b>4</b>	<b>License</b>	<b>37</b>
<b>5</b>	<b>Changelog</b>	<b>39</b>
<b>6</b>	<b>Overview</b>	<b>41</b>
6.1	audoma.choices . . . . .	41
6.2	audoma.decorators . . . . .	41
6.3	audoma.examples . . . . .	42
6.4	audoma.hooks . . . . .	42
6.5	audoma.mixins . . . . .	42
6.6	audoma.openapi . . . . .	43
6.7	audoma.django.db.fields . . . . .	44
6.8	audoma.django.forms.fields . . . . .	45
6.9	audoma.drf.decorators . . . . .	45
6.10	audoma.drf.fields . . . . .	46
6.11	audoma.drf.filters . . . . .	47
6.12	audoma.drf.generics . . . . .	47

6.13	audoma.drf.mixins . . . . .	47
6.14	audoma.drf.serializers . . . . .	49
6.15	audoma.drf.validators . . . . .	49
6.16	audoma.drf.viewsets . . . . .	50
<b>7</b>	<b>Indices and tables</b>	<b>51</b>
	<b>Python Module Index</b>	<b>53</b>
	<b>Index</b>	<b>55</b>

Audoma is a Python library that was designed to make Django Rest Framework documentation easier and more automatic. It's build on the top of *drf-spectacular* and extends some of its functionalities.



## INSTALLATION

### 1.1 Python version

**Audoma supports and was tested for the following versions of python:**

- python 3.7
- python 3.8
- python 3.9

### 1.2 Dependencies

**These distributions will be installed automatically when installing audoma (if you don't have them installed already):**

- Django
- django-rest-framework
- drf-spectacular
- `exrex` - a regular expression generator
- `django-phonenumbers-field` - a phone number field package for django
- `django-macaddress` - a mac address field package for django
- `phonenumbers` - python package for parsing phone numbers - we use it for generating examples of phone numbers
- `django-filter` - django package that allows to filter down a queryset based on a model's fields
- `django-money` - django package that allows to store money and currency values in the database
- `lorem` - generator for random text that looks like Latin.

## 1.3 Install Audoma

Using *pip*:

```
$ pip install audoma
```

## 1.4 Audoma initial configuration

After successful installation, set *AudomaAutoSchema* as a default schema class for your Django Rest Framework project:

```
REST_FRAMEWORK = {  
    # YOUR SETTINGS  
    'DEFAULT_SCHEMA_CLASS': 'audoma.drf.openapi.AudomaAutoSchema',  
}
```

Audoma allows you to add path prefixes that should be included in schema exclusively. All you need to do is declare a variable *SCHEMA\_PATTERN\_PREFIX* in your project's *settings.py* file and add preprocessing function *preprocess\_include\_path\_format* as preprocessing hook in *SPECTACULAR\_SETTINGS* dictionary as in the example below.

```
SCHEMA_PATTERN_PREFIX = 'api'  
  
SPECTACULAR_SETTINGS = {  
    ...  
    'PREPROCESSING_HOOKS': ['audoma.hooks.preprocess_include_path_format'],  
    # OTHER SETTINGS  
}
```



## 2.1 Viewset defined serializers

### 2.1.1 Serializer class custom configs

By default Django Rest Framework provides a method to get a serializer - *get\_serializer*. This checks if viewset instance has set *serializer\_class* property and returns its instance. Audoma Extends this behavior by, extending the number of possible serializer class declarations.

First of all, you are allowed to define *collect* and *response* serializer classes for viewset. Collect serializer will be used to collect and process request data. Response serializers will be used to process data for the response.

Variable name pattern: *common\_{type}\_serializer\_class* (type can be result or collect)  
Example:

```
1  from audoma.drf import viewsets
2  from audoma.drf import mixins
3  from example_app.serializers import (
4      MyCollectSerializer,
5      MyResultSerializer
6  )
7
8  class MyViewSet(
9      mixins.ActionModelMixin,
10     mixins.ListModelMixin,
11     viewsets.GenericViewSet
12 ):
13     common_collect_serializer_class = MyCollectSerializer
14     common_result_serializer_class = MyResultSerializer
```

Additionally audoma allows a definition of a custom serializer for each action in the viewset. This is possible for generic drfs' actions and also for custom actions, created with *@action* decorator.

Variable name pattern: *{action\_name}\_serializer\_class*

Example:

```
1  from rest_framework.decorators import action
2  from rest_framework.response import Response
3
4  from audoma.drf import viewsets
5  from audoma.drf import mixins
6  from example_app.serializers import (
7      MyCreateSerializer,
8      MyCustomActionSerializer
9  )
10
11  class MyViewSet(
12      mixins.ActionModelMixin,
13      mixins.CreateModelMixin,
14      viewsets.GenericViewSet
15  )::
16      create_serializer_class = MyCreateSerializer
17      custom_action_serializer_class = MyCustomActionSerializer
18
19      @action(detail=True, methods=["post"])
20      def custom_action(self, request):
21          serializer = self.get_serializer(data=request.data)
22          serializer.is_valid(raise_exception=True)
23          serializer.save()
24          return Response(serializer.instance, status_code=200)
```

It is also possible for action to serve more than one HTTP method.

In audoma, it is allowed to assign different serializers for each of the actions HTTP methods.

Variable name pattern: *{http\_method}\_{action\_name}\_serializer\_class*

Example:

```
1  from audoma.drf import viewsets
2  from audoma.drf import mixins
3  from example_app.serializers import (
4      MyCreateSerializer,
5      MyCustomActionSerializer
6  )
7
8  class MyViewSet(
9      mixins.ActionModelMixin,
10     mixins.ListModelMixin,
11     viewsets.GenericViewSet
12 ):
13     get_list_serializer_class = MyListSerializer
14     post_list_serializer_class = MyBulkCreateSerializer
```

Back to *collect* and *result* serializers.

Each action may have defined different *collect* and *result* serializer classes.

Variable name pattern: *{action\_name}\_{type}\_serializer\_class* (type can be result or collect)

Example:

```

1  from rest_framework.decorators import action
2  from rest_framework.response import Response
3
4  from audoma.drf import viewsets
5  from example_app.serializers import (
6      MyCreateSerializer,
7      MyCustomActionSerializer
8  )
9
10 class MyViewSet(
11     mixins.ActionModelMixin,
12     viewsets.GenericViewSet
13 ):
14     custom_action_collect_serializer = MyModelCreateSerializer
15     custom_action_result_serializer = MyModelSerializer
16
17     @action(detail=True, methods=["post"])
18     def custom_action(self, request):
19         serializer = self.get_serializer(data=request.data, serializer_type="collect")
20         serializer.is_valid(raise_exception=True)
21         serializer.save()
22         response_serializer = self.get_result_serializer(instance=serializer.instance)
23         return Response(response_serializer.data, status_code=201)

```

The most atomic way of defining serializer classes in audoma is to define serializer per method, action and type.

This means that each action's HTTP method will have *result* and *collect* serializer classes.

Variable name pattern: *{http\_method}\_{action\_name}\_{type}\_serializer\_class* (type can be result or collect)

Example:

```

1  from rest_framework.decorators import action
2  from rest_framework.response import Response
3
4  from audoma.drf import viewsets
5  from audoma.drf import mixins
6  from example_app.serializers import (
7      MyListSerializer,
8      MySerializer,
9      MyCreateSerializer

```

(continues on next page)

(continued from previous page)

```

10 )
11
12 class MyViewSet(
13     mixins.ActionModelMixin,
14     mixins.ListModelMixin,
15     viewsets.GenericViewSet
16 ):
17     get_new_action_result_serializer_class = MyListSerializer
18     post_new_action_result_serializer_class = MySerializer
19     post_new_action_collect_serializer_class = MyCreateSerializer
20
21     @action(detail=True, methods=["post", "get"])
22     def new_action(self, request, *args, **kwargs):
23         if request.method == "POST":
24             serializer = self.get_serializer(data=request.data, serializer_type="collect
25             ↪")
26             serializer.is_valid(raise_exception=True)
27             serializer.save()
28             instance = serializer.instance
29         else:
30             instance = self.get_object()
31             response_serializer = self.get_result_serializer(instance=instance)
32             return Response(response_serializer.data, status_code=201)

```

As you surely presume, all of those serializer classes variables may be defined on one viewset at once. Then those will be traversed in the defined order. The first one matching will be used.

Let's have a look at an example viewset:

```

1  from rest_framework.decorators import action
2  from rest_framework.response import Response
3
4  from audoma.drf import viewsets
5  from example_app.serializers import (
6      MySerachCollectSerializer,
7      MySearchResultSerializer,
8      MyCountCreateSerializer,
9      MyCountUpdateSerializer,
10     MyCountResultSerializer,
11     MyDefaultSerializer
12 )
13 from example_app.models import (
14     MyModel,
15     CountModel
16 )
17

```

(continues on next page)

(continued from previous page)

```

18
19 class MyViewSet(
20     mixins.ActionModelMixin,
21     mixins.CreateModelMixin,
22     mixins.RetrieveModelMixin,
23     mixins.DestroyModelMixin,
24     mixins.ListModelMixin,
25     viewsets.GenericViewSet,
26 ):
27
28     queryset = MyModel.objects.all()
29
30     post_search_collect_serializer_class = MySerachCollectSerializer
31     post_search_result_serializer_class = MySearchResultSerializer
32
33     post_count_collect_serializer_class = MyCountCreateSerializer
34     put_count_collect_serializer_class = MyCountUpdateSerializer
35     count_result_serializer_class = MyCountResultSerializer
36
37     serializer_class = MyDefaultSerializer
38
39     def get_object(self, pk=None):
40         return self.querset.get(pk=pk)
41
42     @action(detail=False, methods=["post"])
43     def search(self, request):
44         serializer = self.get_serializer(data=request.data, serializer_type="collect")
45         serializer.is_valid(raise_exception=True)
46         serializer.save()
47         result_serializer = self.get_result_serializer(instance=serializer.instance)
48         return Response(result_serializer.data, status=201)
49
50     @action(detail=True, methods=["post", "get", "put"])
51     def count(self, request, *args, **kwargs):
52         code = 200
53         if request.method != "GET":
54             serializer = self.get_serializer(data=request.data, serializer_type="collect
55             ↪")
56             serializer.is_valid(raise_exception=True)
57             serializer.save()
58             instance = serializer.instance
59             code = 201 if request.method == "POST"
60         else:
61             instance = CountModel.objects.get_count(slug=kwargs.pop("slug"))
62
63         result_serializer = self.get_result_serializer(instance=instance)
64         return Response(result_serializer.data, status=code)

```

Let's examine the above example.

Action search has two serializers defined, both are defined for the POST method.

One of those will be used to collect data, the other to return the result.

In this case we may also simplify the serializer classes variable names, because search only serves the POST method, so we may also name those variables like this:

```
...
search_collect_serializer_class = MySerachCollectSerializer
search_result_serializer_class = MySearchResultSerializer
...
```

This will work the same way as serializer classes defined in the example.

For the *count* action we have defined three serializers.

First two serializers handle collecting data for “*POST* and *PUT* HTTP methods.

The third serializer is common for all served by *count* HTTP methods, it is a result serializer.

No matter which method we will use, this is the serializer that will be used to return the result.

In this case, if there won’t be further changes in *count* action

we may define *count\_result\_serializer\_class* as *count\_serializer\_class*.

This will work the same way because of the name traversing order defined in audoma.

But this solution may be problematic during introducing any changes.

```
...
post_count_collect_serializer_class = MyCountCreateSerializer
put_count_collect_serializer_class = MyCountUpdateSerializer
count_serializer_class = MyCountResultSerializer
...
```

The one last thing that is left in this viewset is *serializer\_class*.

This variable will be used by all other actions supported by this viewset.

In the viewset definition there are few mixin classes passed, so those will provide some basic functionalities to our viewset.

If this is going to be necessary it is possible to create a separate serializer for those actions also.

Example:

```
1 from rest_framework.decorators import action
2 from rest_framework.response import Response
3
4 from audoma.drf import viewsets
5 from example_app.serializers import (
6     MySerachCollectSerializer,
7     MySearchResultSerializer,
8     MyCountCreateSerializer,
```

(continues on next page)

(continued from previous page)

```

9     MyCountUpdateSerializer,
10    MyCountResultSerializer,
11    MyDefaultSerializer,
12    MyListSerializer,
13    MyCreateSerializer
14 )
15 from example_app.models import (
16     MyModel,
17     CountModel
18 )
19
20
21 class MyViewSet(
22     mixins.ActionModelMixin,
23     mixins.CreateModelMixin,
24     mixins.RetrieveModelMixin,
25     mixins.DestroyModelMixin,
26     mixins.ListModelMixin,
27     viewsets.GenericViewSet,
28 ):
29
30     queryset = MyModel.objects.all()
31
32     post_search_collect_serializer_class = MySerachCollectSerializer
33     post_search_result_serializer_class = MySearchResultSerializer
34
35     post_count_collect_serializer_class = MyCountCreateSerializer
36     put_count_collect_serializer_class = MyCountUpdateSerializer
37     count_result_serializer_class = MyCountResultSerializer
38
39     list_serializer_class = MyListSerializer
40     create_serializer_class = MyCreateSerializer
41     serializer_class = MyDefaultSerializer
42
43     def get_object(self, pk=None):
44         return self.querset.get(pk=pk)
45
46     @action(detail=False, methods=["post"])
47     def search(self, request):
48         serializer = self.get_serializer(data=request.data, serializer_type="collect")
49         serializer.is_valid(raise_exception=True)
50         serializer.save()
51         result_serializer = self.get_result_serializer(instance=serializer.instance)
52         return Response(result_serializer.data, status=201)
53
54     @action(detail=True, methods=["post", "get", "put"])
55     def count(self, request, *args, **kwargs):
56         code = 200
57         if request.method != "GET":
58             serializer = self.get_serializer(data=request.data, serializer_type="collect"
↪ ")
59             serializer.is_valid(raise_exception=True)

```

(continues on next page)

(continued from previous page)

```

60         serializer.save()
61         instance = serializer.instance
62         code = 201 if request.method == "POST"
63     else:
64         instance = CountModel.objects.get_count(slug=kwargs.pop("slug"))
65
66     result_serializer = self.get_result_serializer(instance=instance)
67     return Response(result_serializer.data, status=code)

```

## 2.1.2 Serializer classes name traverse order

After examining the above examples, it is obvious that there is some defined order while traversing defined variables. The variable which will be used as the serializer class is being picked in this order:

- *{http\_method}\_{action\_name}\_{type}\_serializer\_class* (type can be result or collect)
- *{action\_name}\_{type}\_serializer\_class* (type can be result or collect)
- *{http\_method}\_{action\_name}\_serializer\_class*
- *{action\_name}\_serializer\_class*
- *common\_{type}\_serializer\_class* (type can be result or collect)
- *serializer\_class*

For all serializers defined this way, there is also support for proper documentation in api schema.

## 2.2 Permissions

By default, in the *drf-spectacular* viewset permissions were not documented at all. In audoma, permissions are being documented for each viewset separately.

You don't have to define anything extra, this is being handled just out of the box. The only thing it is required is to define permissions on your viewset.

Example:

```

1  from rest_framework.decorators import action
2  from rest_framework.response import Response
3
4  from audoma.drf import viewsets
5  from example_app.serializers import (
6      MySerachCollectSerializer,
7      MySearchResultSerializer,
8      MyCountCreateSerializer,
9      MyCountUpdateSerializer,
10     MyCountResultSerializer,
11     MyDefaultSerializer,

```

(continues on next page)



(continued from previous page)

```

12     MyListSerializer,
13     MyCreateSerializer
14 )
15 from example_app.permissions import (
16     AlternatePermission1,
17     AlternatePermission2,
18     DetailPermission,
19     ViewAndDetailPermission,
20     ViewPermission,
21 )
22 from example_app.models import (
23     MyModel,
24     CountModel
25 )
26
27
28 class MyViewSet(
29     mixins.ActionModelMixin,
30     mixins.CreateModelMixin,
31     mixins.RetrieveModelMixin,
32     mixins.DestroyModelMixin,
33     mixins.ListModelMixin,
34     viewsets.GenericViewSet,
35 ):
36     permission_classes = [
37         IsAuthenticated,
38         ViewAndDetailPermission,
39         DetailPermission,
40         ViewPermission,
41         AlternatePermission1 | AlternatePermission2,
42     ]
43
44     queryset = MyModel.objects.all()
45
46     post_search_collect_serializer_class = MySerachCollectSerializer
47     post_search_result_serializer_class = MySearchResultSerializer
48
49     post_count_collect_serializer_class = MyCountCreateSerializer
50     put_count_collect_serializer_class = MyCountUpdateSerializer
51     count_result_serializer_class = MyCountResultSerializer
52
53     list_serializer_class = MyListSerializer
54     create_serializer_class = MyCreateSerializer
55     serializer_class = MyDefaultSerializer
56
57     def get_object(self, pk=None):
58         return self.querset.get(pk=pk)
59
60     @action(detail=False, methods=["post"])
61     def search(self, request):
62         serializer = self.get_serializer(data=request.data, serializer_type="collect")
63         serializer.is_valid(raise_exception=True)

```

(continues on next page)

(continued from previous page)

```

64         serializer.save()
65         result_serializer = self.get_result_serializer(instance=serializer.instance)
66         return Response(result_serializer.data, status=201)
67
68     @action(detail=True, methods=["post", "get", "put"])
69     def count(self, request, *args, **kwargs):
70         code = 200
71         if request.method != "GET":
72             serializer = self.get_serializer(data=request.data, serializer_type="collect
73         ↪")
74             serializer.is_valid(raise_exception=True)
75             serializer.save()
76             instance = serializer.instance
77             code = 201 if request.method == "POST"
78         else:
79             instance = CountModel.objects.get_count(slug=kwargs.pop("slug"))
80
81         result_serializer = self.get_result_serializer(instance=instance)
82         return Response(result_serializer.data, status=code)

```

Currently there is no way to customize this behavior in audoma, also it is not possible to disable permissions documentation.

## 2.3 Custom choices

Audoma provides a new way of defining choices and new choices class which allows calling choice by its name.

Example definition and usage:

```

1  from audoma.django.db import models
2  from audoma.choices import make_choices
3
4
5  class CarModel(models.Model):
6
7
8      CAR_BODY_TYPES = make_choices(
9          "BODY_TYPES",
10         (
11             (1, "SEDAN", "Sedan"),
12             (2, "COUPE", "Coupe"),
13             (3, "HATCHBACK", "Hatchback"),
14             (4, "PICKUP", "Pickup Truck"),
15         ),
16     )

```

(continues on next page)

(continued from previous page)

```

17     name = models.CharField(max_length=255)
18     body_type = models.IntegerField(choices=CAR_BODY_TYPES.get_choices())
19
20     engine_size = models.FloatField()
21
22     def is_sedan(self):
23         return self.body_type is BODY_TYPE_CHOICES.SEDAN
24 
```

Additionally it's worth mentioning that those choices will be shown in docs in the fields description. Those will also appear in the schema as *x-choices*.

## 2.4 Filters

### 2.4.1 Default Filters

In *drf*, it's possible to define *filterset\_fields* and *filterset\_class*.

By default, *drf-spectacular* supports *django-filters*. Which are being documented.

Audoma has been tested with the default DRFs filter backend and *django\_filters.rest\_framework.DjangoFilterBackend*.

For more accurate documentation, we recommend using *django\_filters.rest\_framework.DjangoFilterBackend* as the default one.

Filters and search fields are being documented out of the box.

Example:

```

1  from rest_framework.filters import SearchFilter
2  from audoma.drf import mixins
3  from audoma.drf import viewsets
4  from django_filters import rest_framework as df_filters
5
6  from example_app.models import CarModel
7  from example_app.serializers import CarModelSerializer
8
9  class CarViewSet(
10     mixins.ActionModelMixin,
11     mixins.RetrieveModelMixin,
12     mixins.ListModelMixin,
13     viewsets.GenericViewSet,
14 ):
15     queryset = CarModel.objects.all()
16     serializer_class = CarModelSerializer
17
18     filter_backends = [SearchFilter, df_filters.DjangoFilterBackend]
19 
```

(continues on next page)

(continued from previous page)

```
20     filterset_fields = ["body_type"]
21     search_fields = ["=manufacturer", "name"]
```

It is also possible to define the *filterset* class which will also be documented without any additional steps.

```
1  from rest_framework.filters import SearchFilter
2  from audoma.drf import mixins
3  from audoma.drf import viewsets
4  from django_filters import rest_framework as df_filters
5
6  from example_app.models import CarModel
7  from example_app.serializers import CarModelSerializer
8
9
10 class CarFilter(df_filters.FilterSet):
11     body_type = df_filters.TypedChoiceFilter(
12         Car.CAR_BODY_TYPES.get_choices(), "body_type",
13         lookup_expr="exact", field_name="body_type"
14     )
15
16     class Meta:
17         model = CarModel
18         fields = [
19             "body_type",
20         ]
21
22
23 class CarViewSet(
24     mixins.ActionModelMixin,
25     mixins.RetrieveModelMixin,
26     mixins.ListModelMixin,
27     viewsets.GenericViewSet,
28 ):
29     queryset = CarModel.objects.all()
30     serializer_class = CarModelSerializer
31
32     filter_backends = [SearchFilter, df_filters.DjangoFilterBackend]
33
34     filterset_class = CarFilter
35     search_fields = ["=manufacturer", "name"]
```

Audoma extends documenting filters with two main features.

Additional enum documentation in field description:

In *drf-spectacular*, enums are being shown only as values possible to pass to the filter.

With audoma, you also get a display value of enum field.

This is being shown as: \* api value - display value

The next feature is schema extension which is not visible in OpenApi frontend.

This schema extension is *x-choices*. Which provides mapping for filter values.

Passing x-choices in schema allows frontend developers to use mapping to show display/value fields without looking into a field description.

## 2.5 Validators

### 2.5.1 ExclusiveFieldsValidator

This is an additional validator, which allows defining mutually exclusive fields in the serializer.

It validates if any of the fields have been given and if not all exclusive fields have been given.

This validator takes params:

- fields - list or a tuple of field names
- message - string message, which will replace the default validator message
- required - boolean which determines if any of the fields must be given
- message\_required - a message which will be displayed if one of the fields is required, and none has been passed

Usage is simple:

```
1  from audoma.drf import serializers
2  from audoma.drf.validators import ExclusiveFieldsValidator
3
4
5  class MutuallyExclusiveExampleSerializer(serializers.Serializer):
6      class Meta:
7          validators = [
8              ExclusiveFieldsValidator(
9                  fields=[
10                     "example_field",
11                     "second_example_field",
12                 ]
13             ),
14         ]
15
16     example_field = serializers.CharField(required=False)
17     second_example_field = serializers.CharField(required=False)
```

## 2.6 Decorators

### 2.6.1 @extend\_schema\_field

Spectacular Docs

This decorator is by default *drf-spectacular* feature.

Audoma only changes its behavior, in *drf-spectacular* using this decorator causes overriding all informations about the field. Audoma does not override information, it only updates available information with those passed to the decorator.

This may be very useful while defining examples.

We don't want to erase all other field information just because we want to define an example for this field.

Also passing all field information additionally just because we want to define an example seems unnecessary and redundant.

Example:

```
1  from audoma.drf.fields import FloatField
2
3  from drf_spectacular.utils import extend_schema_field
4
5  @extend_schema_field(
6      field={
7          "example": 10.00
8      }
9  )
10 class CustomExampleFloatField(FloatField):
11     pass
```

Above we simply add a default example for all fields which will be of class *CustomExampleFloatField*.

### 2.6.2 @audoma\_action

DRFs action docs <<https://www.django-rest-framework.org/api-guide/viewsets/#marking-extra-actions-for-routing>>

This is one of the most complex features offered by audoma, an extension of an action decorator.

Decorator by default is Django Rest Framework functionality.

It also allows registering custom action for viewset.

In the case of *audoma\_action*, it changes a bit how the action function should work, using *audoma\_action* action function should not return a *Response* object, it should return

tuple of instance and status code, *audoma\_action* will take care of creating response out of it.

## Usage

Let's take an example viewset:

```

1  from audoma.drf import mixins
2  from audoma.drf import viewsets
3
4  from app.serializers import (
5      CarListSerializer,
6      CarWriteSerializer,
7      CarDetailsSerializer,
8      CarCreateRateSerializer,
9      CarRateSerializer
10 )
11 from app.models import (
12     Car,
13     CarRate
14 )
15
16
17 class CarViewSet(
18     mixins.ActionModelMixin,
19     mixins.CreateModelMixin,
20     mixins.RetrieveModelMixin,
21     mixins.ListModelMixin,
22     viewsets.GenericViewSet,
23 ):
24
25     permission_classes = [
26         IsAuthenticated,
27         ViewAndDetailPermission,
28         DetailPermission,
29         ViewPermission,
30         AlternatePermission1 | AlternatePermission2,
31     ]
32
33     create_collect_serializer_class = CarWriteSerializer
34     create_result_serializer_class = CarDetailsSerializer
35     retrieve_serializer_class = CarDetailsSerializer
36     list_serializer_class = CarListSerializer
37
38     queryset = {}
39     @audoma_action(
40         detail=True,
41         methods=["get", "post"]
42         collectors=CarCreateRateSerializer,
43         results=CarRateSerializer,
44         errors=[CustomCarRateException]
45     )

```

(continues on next page)

(continued from previous page)

```
46 def rate(self, request, pk=None, *args, **kwargs):
47     if request.method == "POST":
48         collect_serializer = kwargs.pop("collect_serializer")
49         instance = collect_serializer.save()
50     else:
51         instance = CarRate.objects.get_random_car_rate(car_pk=pk)
52     return instance, 200
```

Let's examine the above example.

We've created the viewset with some initial actions served, and serializers assigned to those actions.

Next we've defined a new custom action called *rate*.

This action serves *get* and *post* methods, in case of this action ' we use a single result and collect serializers.

As you may see, *audoma\_action* method does not return the default response, it returns instance and status\_code, the *audoma\_action* decorator takes care of creating the response from this.

Let's modify our example, let there be a custom exception raised.

```
1 from audoma.drf import mixins
2 from audoma.drf import viewsets
3 from rest_framework.exceptions import APIException
4
5 from app.serializers import (
6     CarListSerializer,
7     CarWriteSerializer,
8     CarDetailsSerializer,
9     CarCreateRateSerializer,
10    CarRateSerializer
11 )
12 from app.models import (
13     Car,
14     CarRate
15 )
16
17
18 class CustomCarRateException(APIException):
19     default_detail = "Error during retrieving car rate!"
20     status_code = 500
21
22
23 class CarViewSet(
24     mixins.ActionModelMixin,
```

(continues on next page)



(continued from previous page)

```

25     mixins.CreateModelMixin,
26     mixins.RetrieveModelMixin,
27     mixins.ListModelMixin,
28     viewsets.GenericViewSet,
29 ):
30
31     permission_classes = [
32         IsAuthenticated,
33         ViewAndDetailPermission,
34         DetailPermission,
35         ViewPermission,
36         AlternatePermission1 | AlternatePermission2,
37     ]
38
39     create_collect_serializer_class = CarWriteSerializer
40     create_result_serializer_class = CarDetailsSerializer
41     retrieve_serializer_class = CarDetailsSerializer
42     list_serializer_class = CarListSerializer
43
44     queryset = {}
45
46     @audoma_action(
47         detail=True,
48         methods=["get", "post"]
49         collectors=CarCreateRateSerializer,
50         results=CarRateSerializer,
51         errors=[CustomCarRateException]
52     )
53     def rate(self, request, pk=None, *args, **kwargs):
54         if request.method == "POST":
55             collect_serializer = kwargs.pop("collect_serializer")
56             instance = collect_serializer.save()
57         else:
58             instance = CarRate.objects.get_random_car_rate(car_pk=pk)
59             if not instance:
60                 raise CustomCarRateException
61         return instance, 200

```

After this change it is possible to raise any exception of type *CustomCarRateException* in rate action. Also this exception will be documented in this action schema.

Let's presume that we now want to return status code *201* and rate instance on *post*, but on *get* we want to return the car instance with random rate and status code *200*.

```

1  from audoma.drf import mixins
2  from audoma.drf import viewsets
3  from rest_framework.exceptions import APIException
4

```

(continues on next page)

(continued from previous page)

```

5  from app.serializers import (
6      CarListSerializer,
7      CarWriteSerializer,
8      CarDetailsSerializer,
9      CarCreateRateSerializer,
10     CarRateSerializer
11 )
12 from app.models import (
13     Car,
14     CarRate
15 )
16
17
18 class CustomCarException(APIException):
19     default_detail = "Car can't be found"
20     status_code = 500
21
22
23 class CarViewSet(
24     mixins.ActionModelMixin,
25     mixins.CreateModelMixin,
26     mixins.RetrieveModelMixin,
27     mixins.ListModelMixin,
28     viewsets.GenericViewSet,
29 ):
30
31     permission_classes = [
32         IsAuthenticated,
33         ViewAndDetailPermission,
34         DetailPermission,
35         ViewPermission,
36         AlternatePermission1 | AlternatePermission2,
37     ]
38
39     create_collect_serializer_class = CarWriteSerializer
40     create_result_serializer_class = CarDetailsSerializer
41     retrieve_serializer_class = CarDetailsSerializer
42     list_serializer_class = CarListSerializer
43
44     queryset = {}
45
46     @audoma_action(
47         detail=False,
48         methods=["get", "post"]
49         collectors=CarCreateRateSerializer,
50         results={"post":{201: CarRateSerializer}, "get":{200: CarDetailsSerializer}},
51         errors=[CustomCarException]
52     )
53     def rate(self, request, *args, **kwargs):
54         if request.method == "POST":
55             collect_serializer = kwargs.pop("collect_serializer")
56             instance = collect_serializer.save()

```

(continues on next page)

(continued from previous page)

```

57         return instance. 201
58     else:
59         instance = car.objects.get(pk=pk)
60         if not instance:
61             raise CustomCarException
62         return instance, 200

```

Now we use different a serializer for each method, depending on returned status code.  
Each of this serializer is using different model, *audoma\_action* makes such situations super easy.

Let's take a different example, we have an action that should return a string message, depending on current car state.

```

1  from audoma.drf import mixins
2  from audoma.drf import viewsets
3  from rest_framework.exceptions import APIException
4
5  from app.serializers import (
6      CarListSerializer,
7      CarWriteSerializer,
8      CarDetailsSerializer,
9      CarCreateRateSerializer,
10     CarRateSerializer
11 )
12 from app.models import (
13     Car,
14     CarRate
15 )
16
17 class CustomCarException(APIException):
18     default_detail = "Car can't be found"
19     status_code = 500
20
21
22 class CarViewSet(
23     mixins.ActionModelMixin,
24     mixins.CreateModelMixin,
25     mixins.RetrieveModelMixin,
26     mixins.ListModelMixin,
27     viewsets.GenericViewSet,
28 ):
29
30     permission_classes = [
31         IsAuthenticated,
32         ViewAndDetailPermission,
33         DetailPermission,
34         ViewPermission,
35

```

(continues on next page)

(continued from previous page)

```

36     AlternatePermission1 | AlternatePermission2,
37 ]
38
39 create_collect_serializer_class = CarWriteSerializer
40 create_result_serializer_class = CarDetailsSerializer
41 retrieve_serializer_class = CarDetailsSerializer
42 list_serializer_class = CarListSerializer
43
44 queryset = {}
45
46 @audoma_action(
47     detail=False,
48     methods=["get", "post"]
49     collectors=CarCreateRateSerializer,
50     results={"post":{201: CarRateSerializer}, "get":{200: CarDetailsSerializer}},
51     errors=[CustomCarException]
52 )
53 def rate(self, request, *args, **kwargs):
54     if request.method == "POST":
55         collect_serializer = kwargs.pop("collect_serializer")
56         instance = collect_serializer.save()
57         return instance, 201
58     else:
59         instance = car.objects.get(pk=pk)
60         if not instance:
61             raise CustomCarException
62         return instance, 200
63
64
65 @audoma_action(
66     detail=False,
67     methods=["get"],
68     results="Car is available"
69 )
70 def active(self, request, pk=None):
71     instance = self.get_object(pk=pk)
72     if instance.active:
73         return None, 200
74     return "Car is unavailable", 200

```

This action may return *None* or *string*, but as you may see in the results we have also string defined.

The string default in the results is a string that will be the message returned by default.

The default message will be returned if the instance is *None*.

If returned string instance won't be *None*, then the returned instance will be included in the response.

While returning string message as an instance, audoma simply wraps this message into *json*.

Wrapped message would look like this:

```
{
    "message": "Car is available"
}
```

We can combine those results, so in one action

we may return string instance and model instance.

Let's modify our rate function, so it'll return the default message if the rating is disabled.

```
1  from audoma.drf import mixins
2  from audoma.drf import viewsets
3  from rest_framework.exceptions import APIException
4  from django.conf import settings
5
6  from app.serializers import (
7      CarListSerializer,
8      CarWriteSerializer,
9      CarDetailsSerializer,
10     CarCreateRateSerializer,
11     CarRateSerializer
12 )
13 from app.models import (
14     Car,
15     CarRate
16 )
17
18
19 class CustomCarException(APIException):
20     default_detail = "Car can't be found"
21     status_code = 500
22
23
24 class CarViewSet(
25     mixins.ActionModelMixin,
26     mixins.CreateModelMixin,
27     mixins.RetrieveModelMixin,
28     mixins.ListModelMixin,
29     viewsets.GenericViewSet,
30 ):
31
32     permission_classes = [
33         IsAuthenticated,
34         ViewAndDetailPermission,
35         DetailPermission,
36         ViewPermission,
37         AlternatePermission1 | AlternatePermission2,
38     ]
39
40     create_collect_serializer_class = CarWriteSerializer
41     create_result_serializer_class = CarDetailsSerializer
```

(continues on next page)

(continued from previous page)

```

42 retrieve_serializer_class = CarDetailsSerializer
43 list_serializer_class = CarListSerializer
44
45 queryset = {}
46
47 @audoma_action(
48     detail=False,
49     methods=["get", "post"]
50     collectors=CarCreateRateSerializer,
51     results={
52         "post":{201: CarRateSerializer},
53         "get":{200: CarDetailsSerializer, 204:"Rate service currently unavailable"}
54     },
55     errors=[CustomCarException]
56 )
57 def rate(self, request, *args, **kwargs):
58     if settings.RATE_AVAILABLE:
59         return None, 204
60
61     if request.method == "POST":
62         collect_serializer = kwargs.pop("collect_serializer")
63         instance = collect_serializer.save()
64         return instance, 201
65     else:
66         instance = car.objects.get(pk=pk)
67         if not instance:
68             raise CustomCarException
69         return instance, 200
70
71
72 @audoma_action(
73     detail=False,
74     methods=["get"],
75     results="Car is available"
76 )
77 def active(self, request, pk=None):
78     instance = self.get_object(pk=pk)
79     if instance.active:
80         return None, 200
81     return "Car is unavailable", 200

```

## Params

Decorator *audoma\_action* takes all params which may be passed to the *action* decorator. It also takes additional params, which we will describe below:

## collectors

This param allows defining serializer class which will collect and process request data. To define this, action must serve POST/PATCH or PUT method, otherwise defining those will cause an exception. Collectors may be passed as:

- Serializer class which must inherit from *serializers.BaseSerializer*

```
@audoma_action(  
    detail=False,  
    methods=["post"],  
    results=ExampleOneFieldSerializer,  
    collectors=ExampleOneFieldSerializer,  
)
```

- A dictionary with HTTP methods as keys and serializer classes as values. This allows defining different collector for each HTTP method.

```
@audoma_action(  
    detail=True,  
    methods=["post"],  
    collectors={"post": ExampleModelCreateSerializer},  
    results=ExampleModelSerializer,  
)
```

If you are using PATCH or PUT method for your action, you may ask how to pass an instance to your collect serializer. You simply have to override *get\_object* method on your viewset, and make it return the object you want to pass to collect serializer as an instance for given action and method.

---

### Note:

Passing collectors is optional, so you don't have to pass them.

If collectors won't be passed, and request method will be in *[PUT, POST, PATCH]* then by default, *audoma\_action* will fallback to default *get\_serializer\_class* method for audoma.

---

---

### Note:

If you are using collectors it is important to remember, that your action method should accept additional kwarg *collect\_serializer* which will be a validated collector instance.

## results

This param allows defining custom results for each method and each response status code.

Results param may be passed as:

- Serializer class or which must inherit from *serializers.BaseSerializer* or string variable In this case, the serializer class passed will be used to produce every response coming from this action.

```
@audoma_action(  
    detail=True,  
    methods=["put", "patch"],  
    collectors=ExampleModelCreateSerializer,  
    results=ExampleModelSerializer,  
)
```

- A dictionary with HTTP methods as keys and serializer classes or string variables as values. In This case, there will be a different response serializer for each HTTP method.

```
@audoma_action(  
    detail=False,  
    methods=["get", "post"],  
    collectors={"post": MyCreateSerializer},  
    results={"post": MySerializer, "get": MyListSerializer}  
)
```

- A dictionary with HTTP methods as keys and dictionaries as values. Those dictionaries have status codes as keys and serializer classes or string variables as values.

```
@audoma_action(  
    detail=False,  
    methods=["post"],  
    collectors={"post": MyCreateSerializer},  
    results={"post": {201: MySerializer, 204: MyNoContentSerializer}}  
)
```

---

### Note:

Results param is not mandatory, if you won't pass the results param into `audoma_action`, then there will be a fallback to default *ViewSet defined serializers*.

---



## errors

This param is a list of classes and instances of exceptions, which are allowed to rise in this action. Such behavior prevents rising, not defined exceptions, and allows to document exceptions properly in OpenApi schema.

The main difference between passing exception class and exception instance, is that if you pass exception instance, audoma will not only check if exception type matches, it'll also validate its content. We presume that if you pass, the exception class, you want to accept all exceptions of this class.

In case the risen exception is not defined in `audoma_action errors`, there will be another exception risen: `AudomaActionException`, in case the `settings.DEBUG = False`, this exception will be handled silently by logging it, but the code will pass. In the case of `settings.DEBUG = True`, then the exception won't be silent.

By default audoma accepts some exceptions, which are defined globally. Those exceptions are:

- `NotFound`
- `NotAuthenticated`
- `AuthenticationFailed`
- `ParseError`
- `PermissionDenied`

If you want to extend this list of globally accepted exceptions, you can do it by defining `COMMON_API_ERRORS` in your settings, for example:

```
COMMON_API_ERRORS = [  
    myexceptions.SomeException  
]
```

---

### Note:

Errors param is optional, but if they won't be passed, action will only allow rising globally defined exceptions.

---

### **ignore\_view\_collectors**

Boolean variable which tells if audoma\_action should fallback to default way of retrieving collector from view, if the collector has not been passed and action use method which allows collecting serializer usage.

### **many**

This param decides if the returned instance should be treated as *many* by a serializer. Currently it can only be set to the concrete action, it is impossible to return a instance and multiple instances from one action method using *audoma\_action*.

## **2.7 Examples**

### **2.7.1 Define an example for the field**

Above we described *@extend\_schema\_field* decorator which allows defining example for the field. For all fields defined in audoma, there are examples generated automatically, but you may also pass your example as a field parameter.

Example:

```
from audom.drf import serializers

class SalesContactSerializer(serializers.Serializer):
    phone_number = serializers.PhoneNumberField(example="+48 123 456 789")
    name = serializers.CharField(example="John", max_length=255)
```

After passing the example, it'll be the value shown in example requests in docs.

### **2.7.2 Define custom fields with auto-generated examples**

If you want to define your field with auto example generation, it is possible, that your field class should inherit from the base *ExampleMixin* class, set proper example class.

```
from rest_framework import fields
from audoma.mixins import ExampleMixin
from audoma.examples import NumericExample,
```

(continues on next page)

(continued from previous page)

```
class SaleAmountField(ExampleMixin, fields.Field):
    audoma_example_class = NumericExample
```

## 2.7.3 Define custom example classes

It is possible to define your custom example classes, by default audio has defined two specific example classes inside the *audoma.examples* module:

- *NumericExample*
- *RegexExample*

And one base class:

- *Example*

To define your example class, you should inherit from the *Example* class and override the *generate\_value* method

```
from audoma.examples import Example

class SaleExample(Example):
    def generate_value(self):
        return f"{self.amount} $"
```

## 2.8 Extra Fields

### 2.8.1 Money Field

django\_money docs

Our money field is an extension of the *MoneyField* known from *django\_money*. This field is defined as one field in the model, but it creates two fields in the database. There is nothing complex in this field usage, simply define it in your model:

```
from audoma.django.db import models

class SalesmanStats(models.Model):
    salesman = models.ForeignKey("sale.Salesman", on_delete=models.CASCADE)
    earned = models.MoneyField(max_digits=14, decimal_places=2, default_currency="PLN")
```

Field defined on the model required passing to it two variables.

Currency and amount, in our case we have set the default currency, so passing currency is not obligatory.

Those values may be passed in a few ways:

```
stats = SalesmanStats.objects.get(id=20)
# Simply pass the Money object
stats.earned = Money("99900.23", "PLN")
# You may also pass those variables to objects.create separately
sales = Salesman.objects.get(id=1)
stats = SalesmanStats.objects.create(
    salesman=sales, earned_amount=120,
    earned_currency="PLN"
)
# In our case we defined the default currency, so it also may be
stats = SalesmanStats.objects.create(
    salesman=sales, earned_amount=120
)
# To get the amount we type
print(stats.earned) # this will print 120
print(stats.earned.currency) # will print PLN
```

## 2.8.2 PhoneNumberField

[django-phonenumbers-field docs](#)

Audoma provides a *PhoneNumberField* which is an extension of the *django-phonenumbers-field*.

You can use it in your models straight away, just as the original *PhoneNumberField*, and what we added here is an automatically generated example in documentation, based on country code.

Example:

```
from audoma.django.db import models

class SalesmanStats(models.Model):
    salesman = models.ForeignKey("sale.Salesman", on_delete=models.CASCADE)
    earned = models.MoneyField(max_digits=14, decimal_places=2, default_currency="PLN")
    phone_number = models.PhoneNumberField(region="GB")
```

The above code will result in the following example in the documentation:

```
{
  "salesman": 1,
  "earned": 500,
```

(continues on next page)

(continued from previous page)

```
"phone_number": "+44 20 7894 5678",  
}
```

## 2.9 Serializer Field links

Audoma allows defining links for serializer fields, which values are related to other endpoints. This is useful if you want to limit value choices to other filtered endpoint lists.

Such link won't be visible in redoc/swagger frontend.  
It'll be included in OpenApi schema as *x-choices*.

Link definition:

- `viewname` - the name of a view from which variables should be retrieved
- `value_field` - field name from which value should be retrieved
- `display_field` - field name from which display value should be retrieved

## 2.10 Schema Extensions

### 2.10.1 x-choices

This extension is being added to all fields schema which have limited choice to some range.  
All fields which have defined choices as enum will have this included in their schema.  
If the filter field is also limited to choices this also will be included.

X-choices may have two different forms.  
The first one when it's just a representation of choices enum.  
Then it'll be a mapping:

```
{  
  "x-choices": {  
    "choices": {  
      "value1": "displayValue1",  
      "value2": "displayValue2",  
      "value3": "displayValue3",  
      "value4": "displayValue4",  
    }  
  }  
}
```

This is simply a mapping of values to display values.

This may be useful during displaying choices in for example drop-down.

The second form of x-choices is:

```
{
  "x-choices": {
    "operationRef": "#/paths/manufacturer_viewset~1",
    "value": "$response.body#results/*/id",
    "display": "$response.body#results/*/name"
  }
}
```

This x-choices is a reference to a different endpoint.

This may be used to read limited choices from the related endpoint.

\* operationRef - is a JSON pointer to the related endpoint which should be accessible in this schema

\* value - shows which field should be taken as a field value

\* display - shows which field should be taken as field display value

## CONTRIBUTING TO AUDOMA

Audoma as an open source project welcomes any form of contribution.  
Every contribution is important for us, even if it's a small one

There are different types of contributions

- Documentation fixes, clarifications or improvements
- Creating pull requests
- Creating issues as feature requests or bug reports
- Questions that highlight inconsistencies or workflow issues

### 3.1 Issues

Please include to each submitted issue: \* Brief description of the issue \* Stack trace if there has been an exception raised \* Code that caused the issue \* audoma/drf-spectacular/Django/DRF versions

### 3.2 Pull requests

We also welcome pull requests with fixes or new features.

We strongly advise you to stick to these rules, during creating pull requests: \* Test your changes before creating pull requests, your code has to pass the whole test suite to get merged. The best way will be by launching docker tests, included with the project \* Use linters included in audoma \* Write your own tests for your changes \* If your changes are not trivial consider creating an issue first with changes proposal to get some early feedback \* If you are introducing a new feature, please add an example for this feature in audoma example application

### 3.3 Running unit tests

To test audoma we are using tox with multiple Django/DRF/python configurations.

To run tests enter audoma root dir and simply use:

```
docker-compose -f docker/docker-compose.yml up tests
```

This will run the whole testing process.

### 3.4 Example application

Audoma has included an example Django/DRF application to demonstrate its possibilities.

Every supported feature should have its representation in an example application

To star an example application, from the root folder,

simply run *docker-compose -f docker/docker-compose.yml up example\_app*.

As mentioned above the example application should illustrate every feature of audoma.

If you are going to introduce a new feature it would be nice of you

to include an example of this feature in the example application, this should

make it easier to keep the project consistent.



**LICENSE****MIT License**

Copyright (c) 2022 iteo

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



**CHANGELOG**



## OVERVIEW

## 6.1 audoma.choices

`audoma.choices.make_choices(name: str, choices_tuple: Tuple[Any, str, str]) → audoma.choices._T`

## 6.2 audoma.decorators

`audoma_action.__init__(collectors: Optional[Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]] = None, results: Optional[Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]] = None, errors: Optional[List[Union[Exception, Type[Exception]]]] = None, many: bool = False, ignore_view_collectors: bool = False, **kwargs) → None`

`audoma_action.__call__(func: Callable) → Callable`  
 ” Call of audoma\_action decorator. This is where the magic happens.

**Args:** func - decorated function

**Returns:** wrapper callable.

**class** `audoma.decorators.AudomaActionException`

**class** `audoma.decorators.AudomaArgs(results: Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]], collectors: Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]], errors: List[Union[Exception, Type[Exception]]], many: bool)`

## 6.3 audoma.examples

```
class audoma.examples.Example(field, example=<class 'audoma.examples.DEFAULT'>)
    Class that represents an example for a field. It allows to add example to the field during initialization.

    generate_value() → Type[audoma.examples.DEFAULT]
    get_value() → Any
    to_representation(value) → Any

class audoma.examples.NumericExample(field, example=<class 'audoma.examples.DEFAULT'>)

    generate_value() → float
        Extracts information from the field and generates a random value based on min_value and max_value.
        Returns: Random value between min_value and max_value

class audoma.examples.RegexExample(field, example=<class 'audoma.examples.DEFAULT'>)

    generate_value() → str
        Extracts information from the field and generates a random value based on field's RegexValidators.
        Returns: Generated regex string if regex is found, otherwise returns None
```

## 6.4 audoma.hooks

```
audoma.hooks.postprocess_common_errors_section(result: dict, request, **kwargs) → dict
    Postprocessing hook which adds COMMON_API_ERRORS description to the API description.

audoma.hooks.preprocess_include_path_format(endpoints: Tuple[str, str, str, Callable], **kwargs) →
    List[Tuple[str, str, str, Callable]]
    Preprocessing hook that filters {format} prefixed paths, in case schema pattern prefix is used and {format} path
    params are wanted.
```

## 6.5 audoma.mixins

```
class audoma.mixins.ExampleMixin(*args, example=<class 'audoma.examples.DEFAULT'>, **kwargs)
    A mixin class that adds an example to the field in documentation by overriding field parameter in _spectacular_annotation.

    Args:
        audoma_example_class [Type[Example]] The class that will be used to create the example. Depends on
            the type of field

        audoma_example_class
            alias of audoma.examples.Example

class audoma.mixins.NumericExampleMixin(*args, example=<class 'audoma.examples.DEFAULT'>,
                                         **kwargs)
    A mixin class that adds an example to the field in documentation for numeric fields

    audoma_example_class
        alias of audoma.examples.NumericExample
```

```
class audoma.mixins.RegexExampleMixin(*args, example=<class 'audoma.examples.DEFAULT'>,
                                     **kwargs)
    A mixin class that adds an example to the field in documentation for regex fields
    audoma_example_class
        alias of audoma.examples.RegexExample
```

## 6.6 audoma.openapi

```
class audoma.openapi.AudomaAutoSchema

    _build_exclusive_fields_schema(schema: dict, exclusive_fields: List[str]) → List[dict]
    _extract_action_function(view) → Callable
    _extract_audoma_action_operations(view: django.views.generic.base.View, serializer_type: str) →
        Union[Dict[str, drf_spectacular.utils.OpenApiResponse], Dict[str,
        rest_framework.serializers.BaseSerializer], Dict[str,
        Type[rest_framework.serializers.BaseSerializer]], Dict[str,
        Dict[int, rest_framework.serializers.BaseSerializer]], Dict[int,
        drf_spectacular.utils.OpenApiResponse]]
        Extracts the audoma action operations from the view
    _get_enum_choices_for_field(field)
    _get_link_choices_for_field(field, serializer)
    _get_permissions_description() → str
    _get_request_for_media_type(serializer)
    _get_response_for_code(serializer, status_code, media_types=None)
    _get_serializer(serializer_type: str = 'collect') → Union[rest_framework.serializers.BaseSerializer,
        Type[rest_framework.serializers.BaseSerializer]]
    _handle_permission(permission_class: Union[rest_framework.permissions.OperandHolder,
        rest_framework.permissions.SingleOperandHolder,
        rest_framework.permissions.BasePermission], operations: list, current_operation:
        Type = <class 'rest_framework.permissions.AND'>) → dict
    _map_serializer(serializer: Union[drf_spectacular.types.OpenApiTypes,
        rest_framework.serializers.BaseSerializer,
        Type[rest_framework.serializers.BaseSerializer]], direction: str, bypass_extensions: bool
        = False) → dict
    _map_serializer_field(field: rest_framework.fields.Field, direction: str, bypass_extensions=False) →
        dict
        Allows to use @extend_schema_field with field dict so that it gets updated instead of being overridden
    _parse_action_errors(action_errors) → Dict[int, drf_spectacular.utils.OpenApiResponse]
    _parse_action_result_serializer(serializer: Union[Type[rest_framework.serializers.BaseSerializer],
        str], many: bool = False) →
        Union[rest_framework.serializers.BaseSerializer, str]
```

```
_parse_action_result_serializers(action_serializers: Union[Dict[str, Dict[int, Union[str,
    Type[rest_framework.serializers.BaseSerializer]]], Dict[str,
    Union[str, Type[rest_framework.serializers.BaseSerializer]]], str,
    Type[rest_framework.serializers.BaseSerializer]], many: bool =
    False) → Union[Dict[str, drf_spectacular.utils.OpenApiResponse],
    Dict[str, rest_framework.serializers.BaseSerializer], Dict[str,
    Dict[int, rest_framework.serializers.BaseSerializer]]]

choice_link_schema_generator = <audoma.links.ChoicesOptionsLinkSchemaGenerator
object>

get_description() → str
    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_response_serializers() → Union[rest_framework.serializers.BaseSerializer,
    Type[rest_framework.serializers.BaseSerializer]]
    overrides this for custom behaviour
```

## 6.7 audoma.django.db.fields

Audoma Django model Fields This module contains all the fields from Django models with additional functionality. By inheriting from Audoma Mixins, an example is generated for each field (i.e. FloatField will have example generated based on field's min and max values). We can define custom example by simply passing *example* as an argument to the field.

```
class audoma.django.db.fields.AutoField(*args, **kwargs)
class audoma.django.db.fields.BigAutoField(*args, **kwargs)
class audoma.django.db.fields.BigIntegerField(*args, **kwargs)
class audoma.django.db.fields.BinaryField(*args, **kwargs)
class audoma.django.db.fields.BooleanField(*args, **kwargs)
class audoma.django.db.fields.CharField(*args, **kwargs)
class audoma.django.db.fields.CommaSeparatedIntegerField(*args, **kwargs)
class audoma.django.db.fields.CurrencyField(*args, **kwargs)
class audoma.django.db.fields.DateField(*args, **kwargs)
class audoma.django.db.fields.DateTimeField(*args, **kwargs)
class audoma.django.db.fields.DecimalField(*args, **kwargs)
class audoma.django.db.fields.DurationField(*args, **kwargs)
class audoma.django.db.fields.EmailField(*args, **kwargs)
class audoma.django.db.fields.Field(*args, **kwargs)
class audoma.django.db.fields.FilePathField(*args, **kwargs)
class audoma.django.db.fields.FloatField(*args, **kwargs)
```



```
class audoma.django.db.fields.GenericIPAddressField(*args, **kwargs)
class audoma.django.db.fields.IPAddressField(*args, **kwargs)
class audoma.django.db.fields.IntegerField(*args, **kwargs)
class audoma.django.db.fields.JSONField(*args, **kwargs)
class audoma.django.db.fields.MACAddressField(*args, **kwargs)
class audoma.django.db.fields.MoneyField(*args, **kwargs)

    add_currency_field(cls, name)
        Adds CurrencyField instance to a model class and creates example in documentation.

    formfield(**kwargs)
        Return a django.forms.Field instance for this field.

class audoma.django.db.fields.NullBooleanField(*args, **kwargs)
class audoma.django.db.fields.PhoneNumberField(*args, region=None, **kwargs)
class audoma.django.db.fields.PositiveBigIntegerField(*args, **kwargs)
class audoma.django.db.fields.PositiveIntegerField(*args, **kwargs)
class audoma.django.db.fields.PositiveSmallIntegerField(*args, **kwargs)
class audoma.django.db.fields.SlugField(*args, **kwargs)
class audoma.django.db.fields.SmallAutoField(*args, **kwargs)
class audoma.django.db.fields.SmallIntegerField(*args, **kwargs)
class audoma.django.db.fields.TextField(*args, **kwargs)
class audoma.django.db.fields.TimeField(*args, **kwargs)
class audoma.django.db.fields.URLField(*args, **kwargs)
class audoma.django.db.fields.UUIDField(*args, **kwargs)
```

## 6.8 audoma.django.forms.fields

```
class audoma.django.forms.fields.MoneyField(*, max_value=None, min_value=None, max_digits=None,
                                             decimal_places=None, **kwargs)

    prepare_value(value) → decimal.Decimal
```

## 6.9 audoma.drf.decorators

```
audoma.drf.decorators.document_and_format(serializer_or_field: Any) → Callable
```

## 6.10 audoma.drf.fields

This module is an override for default drf's field module. Most of those fields, are providing additional example functionality, and also has defined schema type.

```
class audoma.drf.fields.BooleanField(*args, **kwargs)
class audoma.drf.fields.CharField(*args, **kwargs)
class audoma.drf.fields.ChoiceField(*args, **kwargs)
class audoma.drf.fields.DateField(*args, **kwargs)
class audoma.drf.fields.DateTimeField(*args, **kwargs)
class audoma.drf.fields.DecimalField(*args, **kwargs)
class audoma.drf.fields.DictField(*args, **kwargs)
class audoma.drf.fields.DurationField(*args, **kwargs)
class audoma.drf.fields.EmailField(*args, **kwargs)
class audoma.drf.fields.FileField(*args, **kwargs)
class audoma.drf.fields.FilePathField(*args, **kwargs)
class audoma.drf.fields.FloatField(*args, **kwargs)
class audoma.drf.fields.HStoreField(*args, **kwargs)
class audoma.drf.fields.IPAddressField(*args, **kwargs)
class audoma.drf.fields.ImageField(*args, **kwargs)
class audoma.drf.fields.IntegerField(*args, **kwargs)
class audoma.drf.fields.JSONField(*args, **kwargs)
class audoma.drf.fields.ListField(*args, **kwargs)
class audoma.drf.fields.MACAddressField(*args, **kwargs)
class audoma.drf.fields.MoneyField(*args, **kwargs)
class audoma.drf.fields.MultipleChoiceField(*args, **kwargs)
class audoma.drf.fields.NullBooleanField(*args, **kwargs)
class audoma.drf.fields.PhoneNumberField(*args, **kwargs)
class audoma.drf.fields.ReadOnlyField(*args, **kwargs)
class audoma.drf.fields.RegexField(*args, **kwargs)
class audoma.drf.fields.SerializerMethodField(*args, **kwargs)
class audoma.drf.fields.SlugField(*args, **kwargs)
class audoma.drf.fields.TimeField(*args, **kwargs)
class audoma.drf.fields.URLField(*args, **kwargs)
class audoma.drf.fields.UUIDField(*args, **kwargs)
```

## 6.11 audoma.drf.filters

**class** audoma.drf.filters.**DocumentedTypedChoiceFilter**(*full\_choices: NamedTuple, parameter\_name: str, \*\*kwargs*)

Extended TypedChoiceFilter to generate documentation automatically

## 6.12 audoma.drf.generics

**class** audoma.drf.generics.**GenericAPIView**(*\*\*kwargs*)

Extended GenericAPIView known from rest\_framework. This class extends *get\_serializer* and *get\_serializer\_class* methods. Also provides *get\_result\_serializer*, which is a shortcut for *get\_serializer* with proper param.

**\_check\_action\_function**()

**\_extract\_audoma\_action\_serializer**(*serializer\_type: str, audoma\_args: audoma.decorators.AudomaArgs, status\_code: Optional[int] = None*)

**get\_audoma\_action\_config**()

**get\_audoma\_action\_serializer\_class**(*serializer\_type: str, status\_code: Optional[int] = None*)

**get\_result\_serializer**(*\*args, \*\*kwargs*) → rest\_framework.serializers.BaseSerializer  
Shortcut for *get\_serializer*. Simply has *serializer\_type* set to *result*

**get\_serializer**(*\*args, \*\*kwargs*) → rest\_framework.serializers.BaseSerializer  
Passes additional param to *get\_serializer\_class*.

**kwargs:**

**serializer\_type** - defines if serializer is collect or result serializer. result serializer will be used to produce response, collect to process incoming data.

**serializer\_class** - it is possible to pass **serializer\_class** to **get\_serializer**, this will ends with returning passed *serializer\_class* object.

**Returns:** Object of obtained serializer class.

**get\_serializer\_class**(*serializer\_type: str = 'collect', many: bool = False, status\_code: Optional[int] = None*) → Type[rest\_framework.serializers.BaseSerializer]

Extends default *get\_serializer\_class* method. This returns proper *serializer\_class* for current request.

**Args:** *serializer\_type* - *serializer\_type* of serializer to be returned, it may be collect or result serializer.

**Returns:** This returns *serializer\_class*

## 6.13 audoma.drf.mixins

This module overwrites basic mixins provided by django rest framework. Mixins defined here should be used instead of default drf's mixins. Those mixins should be used to allow usage of extended *get\_serializer* method>

Example:

```
from audoma.drf import mixins from audoma.drf import viewsets
```

```
class ExampleModelViewSet( mixins.ActionModelMixin, mixins.CreateModelMixin,
viewsets.GenericViewSet,
```

```
) : serializer_class = ExampleModelSerializer queryset = ExampleModel.objects.all()
```

```
class audoma.drf.mixins.ActionModelMixin
```

```
    get_success_headers(data: dict) → dict
```

```
    perform_action(request: rest_framework.request.Request, success_status: int = 200, instance:
                    Optional[Any] = None, partial: bool = False, **kwargs) →
                    rest_framework.response.Response
```

```
    retrieve_instance(request: rest_framework.request.Request, instance: Optional[Any] = None,
                      success_status: int = 200, **kwargs) → rest_framework.response.Response
```

```
class audoma.drf.mixins.BulkCreateModelMixin
```

Either create a single or many model instances in bulk by using the Serializers many=True ability from Django REST >= 2.2.5. .. note:

This mixin uses the same method to create model instances as ``CreateModelMixin`` because both non-bulk and bulk requests will use ``POST`` request method.

```
    create(request: rest_framework.request.Request, *args, **kwargs) → rest_framework.response.Response
```

```
    perform_bulk_create(serializer: rest_framework.serializers.BaseSerializer) → None
```

```
class audoma.drf.mixins.BulkUpdateModelMixin
```

Update model instances in bulk by using the Serializers many=True ability from Django REST >= 2.2.5.

```
    bulk_update(request: rest_framework.request.Request, *args, **kwargs) →
                rest_framework.response.Response
```

```
    get_object() → Any
```

```
    partial_bulk_update(request: rest_framework.request.Request, *args, **kwargs) →
                        rest_framework.response.Response
```

```
    perform_bulk_update(serializer: rest_framework.serializers.BaseSerializer) → None
```

```
    perform_update(serializer: rest_framework.serializers.BaseSerializer) → None
```

```
class audoma.drf.mixins.CreateModelMixin
```

```
    create(request: rest_framework.request.Request, *args, **kwargs) → rest_framework.response.Response
```

```
class audoma.drf.mixins.DestroyModelMixin
```

```
    destroy(request: rest_framework.request.Request, *args, **kwargs) → rest_framework.response.Response
```

```
class audoma.drf.mixins.ListModelMixin
```

```
    get_paginated_response(data: List[Dict]) → rest_framework.response.Response
```

```
    list(request: rest_framework.request.Request, *args, **kwargs) → rest_framework.response.Response
```

```
class audoma.drf.mixins.RetrieveModelMixin
```

```
    retrieve(request: rest_framework.request.Request, *args, **kwargs) → rest_framework.response.Response
```

```
class audoma.drf.mixins.UpdateModelMixin
```

`update(request: rest_framework.request.Request, *args, **kwargs) → rest_framework.response.Response`

## 6.14 audoma.drf.serializers

`class audoma.drf.serializers.BulkListSerializer(*args, **kwargs)`

`class audoma.drf.serializers.DefaultMessageSerializer(*args, **kwargs)`

`class audoma.drf.serializers.ListSerializer(*args, **kwargs)`

`class audoma.drf.serializers.ModelSerializer(*args, **kwargs)`

Extends default ModelSerializer, modifies serializer\_field\_mapping (replaces some fields with audoma fields).  
Adds support for generating audoma example for field.

`build_standard_field(field_name, model_field) → Tuple[Union[Type[rest_framework.fields.Field], dict]]`

Adds support for mapping example from model fields to model serializer fields.

`serializer_choice_field`

alias of `audoma.drf.fields.ChoiceField`

`class audoma.drf.serializers.ResultSerializerClassMixin`

Allows to define wrap for serializer result.

`class audoma.drf.serializers.Serializer(*args, **kwargs)`

`audoma.drf.serializers.result_serializer_class(serializer_class:`

`Type[rest_framework.serializers.BaseSerializer]) →`

`Type[rest_framework.serializers.BaseSerializer]`

Helper function which wraps the serializer result if necessary.

**Args:**

- `SerializerClass` - serializer class which result should be wrapped

Returns: `ResultSerializer` class

## 6.15 audoma.drf.validators

`class audoma.drf.validators.ExclusiveFieldsValidator(fields: Union[List[str], Tuple[str]], message: Optional[str] = None, required: bool = True, message_required: Optional[str] = None)`

This is extra validator defined in audoma. This validator allows to define mutually exclusive fields.

**Attributes:** `fields` - list or a tuple of mutually exclusive field names `message` - string validation error message  
`required` - boolean value, determines if fields are required `message_required` - string message if one of fields is required and none has been passed

**Args:** `fields` - list or a tuple of mutually exclusive field names `message` - string validation error message  
`required` - boolean value, determines if fields are required `message_required` - string message if one of fields is required and none has been passed

`message = 'The fields {field_names} are mutually exclusive arguments.'`

`message_required = 'One of the fields {field_names} is required.'`

## 6.16 audoma.drf.viewsets

**class** `audoma.drf.viewsets.AudomaPagination`

A simple page number based style that supports page numbers as query parameters.

**Note:** If this won't be used it'll cause less explicit pagination documentation. This class does not provide any additional functionality.

**Args:** `page_size` (int) - number of items per page - by default this is set to 25 `max_page_size` (int) - maximum number of items per page - by default this is set to 2000

**get\_paginated\_response\_schema**(*schema: List[dict]*) → dict  
Simple method to add pagination information to the schema.

**Args:** `schema` (List[dict]) - list of schema items

**Returns:** Dictionary with pagination information including examples

**max\_page\_size** = 2000

**page\_size** = 25

**class** `audoma.drf.viewsets.GenericViewSet`(\*\*kwargs)

**\_parse\_response\_data**(*response\_data: List[Any]*) → List[str]

**handle\_exception**(*exc: Exception*) → rest\_framework.response.Response  
Handle any exception that occurs, by returning an appropriate response, or re-raising the error.

**pagination\_class**  
alias of `audoma.drf.viewsets.AudomaPagination`

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### a

- `audoma.choices`, 41
- `audoma.django.db.fields`, 44
- `audoma.django.forms.fields`, 45
- `audoma.drf.decorators`, 45
- `audoma.drf.fields`, 46
- `audoma.drf.filters`, 47
- `audoma.drf.generics`, 47
- `audoma.drf.mixins`, 47
- `audoma.drf.serializers`, 49
- `audoma.drf.validators`, 49
- `audoma.drf.viewsets`, 50
- `audoma.hooks`, 42
- `audoma.mixins`, 42
- `audoma.openapi`, 43



## Symbols

<code>__call__()</code>	( <i>audoma.decorators.audoma_action</i> method), 41	<code>doma.openapi.AudomaAutoSchema</code> method), 43
<code>__init__()</code>	( <i>audoma.decorators.audoma_action</i> method), 41	<code>_map_serializer_field()</code> ( <i>audoma.openapi.AudomaAutoSchema</i> method), 43
<code>_build_exclusive_fields_schema()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	<code>_parse_action_errors()</code> ( <i>audoma.openapi.AudomaAutoSchema</i> method), 43
<code>_check_action_function()</code>	( <i>audoma.drf.generics.GenericAPIView</i> method), 47	<code>_parse_action_result_serializer()</code> ( <i>audoma.openapi.AudomaAutoSchema</i> method), 43
<code>_extract_action_function()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	<code>_parse_action_result_serializers()</code> ( <i>audoma.openapi.AudomaAutoSchema</i> method), 43
<code>_extract_audoma_action_operations()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	<code>_parse_response_data()</code> ( <i>audoma.drf.viewsets.GenericViewSet</i> method), 50
<code>_extract_audoma_action_serializer()</code>	( <i>audoma.drf.generics.GenericAPIView</i> method), 47	
<code>_get_enum_choices_for_field()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	
<code>_get_link_choices_for_field()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	
<code>_get_permissions_description()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	
<code>_get_request_for_media_type()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	
<code>_get_response_for_code()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	
<code>_get_serializer()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	
<code>_handle_permission()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	
<code>_map_serializer()</code>	( <i>audoma.openapi.AudomaAutoSchema</i> method), 43	

## A

<code>ActionModelMixin</code>	(class in <i>audoma.drf.mixins</i> ), 48
<code>add_currency_field()</code>	( <i>audoma.django.db.fields.MoneyField</i> method), 45
<code>audoma.choices</code>	module, 41
<code>audoma.django.db.fields</code>	module, 44
<code>audoma.django.forms.fields</code>	module, 45
<code>audoma.drf.decorators</code>	module, 45
<code>audoma.drf.fields</code>	module, 46
<code>audoma.drf.filters</code>	module, 47
<code>audoma.drf.generics</code>	module, 47
<code>audoma.drf.mixins</code>	module, 47
<code>audoma.drf.serializers</code>	module, 49
<code>audoma.drf.validators</code>	

module, 49  
audoma.drf.viewsets  
  module, 50  
audoma.hooks  
  module, 42  
audoma.mixins  
  module, 42  
audoma.openapi  
  module, 43  
audoma\_example\_class (audoma.mixins.ExampleMixin attribute), 42  
audoma\_example\_class (audoma.mixins.NumericExampleMixin attribute), 42  
audoma\_example\_class (audoma.mixins.RegexExampleMixin attribute), 43  
AudomaActionException (class in audoma.decorators), 41  
AudomaArgs (class in audoma.decorators), 41  
AudomaAutoSchema (class in audoma.openapi), 43  
AudomaPagination (class in audoma.drf.viewsets), 50  
AutoField (class in audoma.django.db.fields), 44

## B

BigAutoField (class in audoma.django.db.fields), 44  
BigIntegerField (class in audoma.django.db.fields), 44  
BinaryField (class in audoma.django.db.fields), 44  
BooleanField (class in audoma.django.db.fields), 44  
BooleanField (class in audoma.drf.fields), 46  
build\_standard\_field() (audoma.drf.serializers.ModelSerializer method), 49

bulk\_update() (audoma.drf.mixins.BulkUpdateModelMixin method), 48  
BulkCreateModelMixin (class in audoma.drf.mixins), 48  
BulkListSerializer (class in audoma.drf.serializers), 49  
BulkUpdateModelMixin (class in audoma.drf.mixins), 48

## C

CharField (class in audoma.django.db.fields), 44  
CharField (class in audoma.drf.fields), 46  
choice\_link\_schema\_generator (audoma.openapi.AudomaAutoSchema attribute), 44  
ChoiceField (class in audoma.drf.fields), 46  
CommaSeparatedIntegerField (class in audoma.django.db.fields), 44  
create() (audoma.drf.mixins.BulkCreateModelMixin method), 48

create() (audoma.drf.mixins.CreateModelMixin method), 48  
CreateModelMixin (class in audoma.drf.mixins), 48  
CurrencyField (class in audoma.django.db.fields), 44

## D

DateField (class in audoma.django.db.fields), 44  
DateField (class in audoma.drf.fields), 46  
DateTimeField (class in audoma.django.db.fields), 44  
DateTimeField (class in audoma.drf.fields), 46  
DecimalField (class in audoma.django.db.fields), 44  
DecimalField (class in audoma.drf.fields), 46  
DefaultMessageSerializer (class in audoma.drf.serializers), 49  
destroy() (audoma.drf.mixins.DestroyModelMixin method), 48  
DestroyModelMixin (class in audoma.drf.mixins), 48  
DictField (class in audoma.drf.fields), 46  
document\_and\_format() (in module audoma.drf.decorators), 45  
DocumentedTypedChoiceFilter (class in audoma.drf.filters), 47  
DurationField (class in audoma.django.db.fields), 44  
DurationField (class in audoma.drf.fields), 46

## E

EmailField (class in audoma.django.db.fields), 44  
EmailField (class in audoma.drf.fields), 46  
Example (class in audoma.examples), 42  
ExampleMixin (class in audoma.mixins), 42  
ExclusiveFieldsValidator (class in audoma.drf.validators), 49

## F

FileField (class in audoma.django.db.fields), 44  
FileField (class in audoma.drf.fields), 46  
FilePathField (class in audoma.django.db.fields), 44  
FilePathField (class in audoma.drf.fields), 46  
FloatField (class in audoma.django.db.fields), 44  
FloatField (class in audoma.drf.fields), 46  
formfield() (audoma.django.db.fields.MoneyField method), 45

## G

generate\_value() (audoma.examples.Example method), 42  
generate\_value() (audoma.examples.NumericExample method), 42  
generate\_value() (audoma.examples.RegexExample method), 42  
GenericAPIView (class in audoma.drf.generics), 47  
GenericIPAddressField (class in audoma.django.db.fields), 44

GenericViewSet (class in *audoma.drf.viewsets*), 50

get\_audoma\_action\_config() (audoma.drf.generics.GenericAPIView method), 47

get\_audoma\_action\_serializer\_class() (audoma.drf.generics.GenericAPIView method), 47

get\_description() (audoma.openapi.AudomaAutoSchema method), 44

get\_object() (audoma.drf.mixins.BulkUpdateModelMixin method), 48

get\_operation() (audoma.openapi.AudomaAutoSchema method), 44

get\_operation\_id() (audoma.openapi.AudomaAutoSchema method), 44

get\_paginated\_response() (audoma.drf.mixins.ListModelMixin method), 48

get\_paginated\_response\_schema() (audoma.drf.viewsets.AudomaPagination method), 50

get\_response\_serializers() (audoma.openapi.AudomaAutoSchema method), 44

get\_result\_serializer() (audoma.drf.generics.GenericAPIView method), 47

get\_serializer() (audoma.drf.generics.GenericAPIView method), 47

get\_serializer\_class() (audoma.drf.generics.GenericAPIView method), 47

get\_success\_headers() (audoma.drf.mixins.ActionModelMixin method), 48

get\_value() (audoma.examples.Example method), 42

## H

handle\_exception() (audoma.drf.viewsets.GenericViewSet method), 50

HStoreField (class in *audoma.drf.fields*), 46

## I

IntegerField (class in *audoma.drf.fields*), 46

IntegerField (class in *audoma.django.db.fields*), 45

IntegerField (class in *audoma.drf.fields*), 46

IPAddressField (class in *audoma.django.db.fields*), 45

IPAddressField (class in *audoma.drf.fields*), 46

## J

JSONField (class in *audoma.django.db.fields*), 45

JSONField (class in *audoma.drf.fields*), 46

## L

list() (audoma.drf.mixins.ListModelMixin method), 48

ListField (class in *audoma.drf.fields*), 46

ListModelMixin (class in *audoma.drf.mixins*), 48

ListSerializer (class in *audoma.drf.serializers*), 49

## M

MACAddressField (class in *audoma.django.db.fields*), 45

MACAddressField (class in *audoma.drf.fields*), 46

make\_choices() (in module *audoma.choices*), 41

max\_page\_size (audoma.drf.viewsets.AudomaPagination attribute), 50

message (audoma.drf.validators.ExclusiveFieldsValidator attribute), 49

message\_required (audoma.drf.validators.ExclusiveFieldsValidator attribute), 49

ModelSerializer (class in *audoma.drf.serializers*), 49

module

audoma.choices, 41

audoma.django.db.fields, 44

audoma.django.forms.fields, 45

audoma.drf.decorators, 45

audoma.drf.fields, 46

audoma.drf.filters, 47

audoma.drf.generics, 47

audoma.drf.mixins, 47

audoma.drf.serializers, 49

audoma.drf.validators, 49

audoma.drf.viewsets, 50

audoma.hooks, 42

audoma.mixins, 42

audoma.openapi, 43

MoneyField (class in *audoma.django.db.fields*), 45

MoneyField (class in *audoma.django.forms.fields*), 45

MoneyField (class in *audoma.drf.fields*), 46

MultipleChoiceField (class in *audoma.drf.fields*), 46

## N

NullBooleanField (class in *audoma.django.db.fields*), 45

NullBooleanField (class in *audoma.drf.fields*), 46

NumericExample (class in *audoma.examples*), 42

NumericExampleMixin (class in *audoma.mixins*), 42

## P

page\_size (audoma.drf.viewsets.AudomaPagination attribute), 50

`pagination_class` (*audoma.drf.viewsets.GenericViewSet* attribute), 50

`partial_bulk_update()` (*audoma.drf.mixins.BulkUpdateModelMixin* method), 48

`perform_action()` (*audoma.drf.mixins.ActionModelMixin* method), 48

`perform_bulk_create()` (*audoma.drf.mixins.BulkCreateModelMixin* method), 48

`perform_bulk_update()` (*audoma.drf.mixins.BulkUpdateModelMixin* method), 48

`perform_update()` (*audoma.drf.mixins.BulkUpdateModelMixin* method), 48

`PhoneNumberField` (class in *audoma.django.db.fields*), 45

`PhoneNumberField` (class in *audoma.drf.fields*), 46

`PositiveBigIntegerField` (class in *audoma.django.db.fields*), 45

`PositiveIntegerField` (class in *audoma.django.db.fields*), 45

`PositiveSmallIntegerField` (class in *audoma.django.db.fields*), 45

`postprocess_common_errors_section()` (in module *audoma.hooks*), 42

`prepare_value()` (*audoma.django.forms.fields.MoneyField* method), 45

`preprocess_include_path_format()` (in module *audoma.hooks*), 42

`serializer_choice_field` (*audoma.drf.serializers.ModelSerializer* attribute), 49

`SerializerMethodField` (class in *audoma.drf.fields*), 46

`SlugField` (class in *audoma.django.db.fields*), 45

`SlugField` (class in *audoma.drf.fields*), 46

`SmallAutoField` (class in *audoma.django.db.fields*), 45

`SmallIntegerField` (class in *audoma.django.db.fields*), 45

**T**

`TextField` (class in *audoma.django.db.fields*), 45

`TimeField` (class in *audoma.django.db.fields*), 45

`TimeField` (class in *audoma.drf.fields*), 46

`to_representation()` (*audoma.examples.Example* method), 42

**U**

`update()` (*audoma.drf.mixins.UpdateModelMixin* method), 48

`UpdateModelMixin` (class in *audoma.drf.mixins*), 48

`URLField` (class in *audoma.django.db.fields*), 45

`URLField` (class in *audoma.drf.fields*), 46

`UUIDField` (class in *audoma.django.db.fields*), 45

`UUIDField` (class in *audoma.drf.fields*), 46

## R

`ReadOnlyField` (class in *audoma.drf.fields*), 46

`RegexExample` (class in *audoma.examples*), 42

`RegexExampleMixin` (class in *audoma.mixins*), 42

`RegexField` (class in *audoma.drf.fields*), 46

`result_serializer_class()` (in module *audoma.drf.serializers*), 49

`ResultSerializerClassMixin` (class in *audoma.drf.serializers*), 49

`retrieve()` (*audoma.drf.mixins.RetrieveModelMixin* method), 48

`retrieve_instance()` (*audoma.drf.mixins.ActionModelMixin* method), 48

`RetrieveModelMixin` (class in *audoma.drf.mixins*), 48

## S

`Serializer` (class in *audoma.drf.serializers*), 49