Django REST Witchcraft Documentation

Release 0.12.1

Serkan Hosca

Dec 21, 2022

Contents

Django REST Framework integration with SQLAlchemy

django-rest-witchcraft is an extension for Django REST Framework that adds support for SQLAlchemy. It aims to provide a similar development experience to building REST api's with Django REST Framework with Django ORM, except with SQLAlchemy.

CHAPTER 1

Installation

```
pip install django-rest-witchcraft
```

CHAPTER 2

Quick Start

First up, lets define some simple models:

```
import sqlalchemy as sa
import sqlalchemy.orm  # noqa
from sqlalchemy.ext.declarative import declarative_base

engine = sa.create_engine('sqlite:///:memory:', echo=True)
session = sa.orm.scoped_session(sa.orm.sessionmaker(bind=engine))

Base = declarative_base()
Base.query = session.query_property()


class Group(Base):
    __tablename__ = 'groups'

    id = sa.Column(sa.Integer(), primary_key=True, autoincrement=True)
    name = sa.Column(sa.String())


class User(Base):
    __tablename__ = 'users'

    id = sa.Column(sa.Integer(), primary_key=True, autoincrement=True)
    name = sa.Column(sa.String())
    fullname = sa.Column(sa.String())
    password = sa.Column(sa.String())

    _group_id = sa.Column('group_id', sa.Integer(), sa.ForeignKey('groups.id'))
    group = sa.orm.relationship(Group, backref='users')


class Address(Base):
    __tablename__ = 'addresses'
```

(continues on next page)

(continued from previous page)

```
    id = sa.Column(sa.Integer(), primary_key=True, autoincrement=True)
    email_address = sa.Column(sa.String(), nullable=False)

    _user_id = sa.Column(sa.Integer(), sa.ForeignKey('users.id'))
    user = sa.orm.relationship(User, backref='addresses')

Base.metadata.create_all(engine)
```

Nothing fancy here, we have a `User` class that can belongs to a `Group` instance and has many `Address` instances

This serializer can handle nested create, update or partial update operations.

Lets define a serializer for `User` with all the fields:

```
class UserSerializer(serializers.ModelSerializer):

    class Meta:
        model = User
        session = session
        fields = '__all__'
```

This will create the following serializer for us:

```
>>> serializer = UserSerializer()

>>> serializer
UserSerializer():
    id = IntegerField(allow_null=False, help_text=None, label='Id', required=True)
    name = CharField(allow_null=True, help_text=None, label='Name', max_length=None,␣
↪required=False)
    fullname = CharField(allow_null=True, help_text=None, label='Fullname', max_
↪length=None, required=False)
    password = CharField(allow_null=True, help_text=None, label='Password', max_
↪length=None, required=False)
    group = GroupSerializer(allow_null=True, is_nested=True, required=False):
        id = IntegerField(allow_null=False, help_text=None, label='Id',␣
↪required=False)
        name = CharField(allow_null=True, help_text=None, label='Name', max_
↪length=None, required=False)
    addresses = AddressSerializer(allow_null=True, many=True, required=False):
        id = IntegerField(allow_null=False, help_text=None, label='Id',␣
↪required=False)
        email_address = CharField(allow_null=False, help_text=None, label='Email_
↪address', max_length=None, required=True)
    url = UriField(read_only=True)
```

Lets try to create a `User` instance with our brand new serializer:

```
serializer = UserSerializer(data={
    'name': 'shosca',
    'password': 'swordfish',
})
serializer.is_valid()
serializer.save()

user = serializer.instance
```

This will create the following user for us:

```
>>> user
User(_group_id=None, id=1, name='shosca', fullname=None, password='swordfish')
```

Lets try to update our user `User` instance and change its password:

```
serializer = UserSerializer(user, data={
    'name': 'shosca',
    'password': 'password',
})
serializer.is_valid()
serializer.save()

user = serializer.instance
```

Our user now looks like:

```
>>> user
User(_group_id=None, id=1, name='shosca', fullname=None, password='password')
```

Lets try to update our `User` instance again, but this time lets change its password only:

```
serializer = UserSerializer(user, data={
    'password': 'swordfish',
}, partial=True)
serializer.is_valid()
serializer.save()

user = serializer.instance
```

This will update the following user for us:

```
>>> user
User(_group_id=None, id=1, name='shosca', fullname=None, password='swordfish')
```

Our user does not belong to a `Group`, lets fix that:

```
group = Group(name='Admin')
session.add(group)
session.flush()

serializer = UserSerializer(user, data={
    'group': {'id': group.id}
})
serializer.is_valid()
serializer.save()

user = serializer.instance
```

Now, our user looks like:

```
>>> user
User(_group_id=1, id=1, name='shosca', fullname=None, password='swordfish')

>>> user.group
Group(id=1, name='Admin')
```

We can also change the name of our user's group through the user using nested updates:

```
class UserSerializer(serializers.ModelSerializer):

    class Meta:
        model = User
        session = session
        fields = '__all__'
        extra_kwargs = {
            'group': {'allow_nested_updates': True}
        }

serializer = UserSerializer(user, data={
    'group': {'name': 'Super User'}
}, partial=True)
serializer.is_valid()

user = serializer.save()
```

Now, our user looks like:

```
>>> user
User(_group_id=1, id=1, name='shosca', fullname=None, password='swordfish')

>>> user.group
Group(id=1, name='Super User')
```

We can use this serializer in a viewset like:

```
from rest_witchcraft import viewsets

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.query
    serializer_class = UserSerializer
```

And we can register this viewset in our `urls.py` like:

```
from rest_witchcraft import routers

router = routers.DefaultRouter()
router.register(r'users', UserViewSet)

urlpatterns = [
    ...
    url(r'^', include(router.urls)),
    ...
]
```

2.1 Changelog

2.1.1 0.12.1 (2022-12-21)

- Add support for DRF 3.14 and python 3.11 (#89) [Serkan Hosca]
- [pre-commit.ci] pre-commit autoupdate (#87) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#86) [pre-commit-ci[bot], pre- commit-ci[bot]]

- Add dj41 to build matrix (#85) [Serkan Hosca]
- [pre-commit.ci] pre-commit autoupdate (#84) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#83) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#82) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#81) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#80) [Serkan Hosca, pre-commit- ci[bot], pre-commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#79) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#76) [Serkan Hosca, pre-commit- ci[bot], pre-commit-ci[bot]]
- Multi python dockerfile for local dev (#78) [Serkan Hosca]

2.1.2 0.12.0 (2022-03-26)

- Add django 4 to tox matrix (#77) [Serkan Hosca]
- [pre-commit.ci] pre-commit autoupdate (#75) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#74) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#73) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#72) [pre-commit-ci[bot], pre- commit-ci[bot]]
- [pre-commit.ci] pre-commit autoupdate (#71) [pre-commit-ci[bot], pre- commit-ci[bot]]
- Sourcery fixes (#70) [Serkan Hosca]

2.1.3 0.11.1 (2021-05-06)

- Use suppress from contextlib (#69) [Serkan Hosca]

2.1.4 0.11.0 (2021-03-20)

- Add sqlalchemy 1.4 support (#65) [Serkan Hosca]
- Github actions build badge (#64) [Serkan Hosca]
- Adding github actions (#57) [Serkan Hosca]
- Fix build link. [Serkan Hosca]
- Add dj3.1 and drf3.11 to matrix (#62) [Serkan Hosca]
- Pre-commit imporanize pyupgrade and docformat (#61) [Serkan Hosca]
- Add django3 to build matrix (#58) [Serkan Hosca]
- Add drf 3.10 on build matrix (#56) [Serkan Hosca]

2.1.5 0.10.3 (2019-11-07)

- Checking manifest with pre-commit (#55) [Miroslav Shubernetskiy]

2.1.6 0.10.2 (2019-10-31)

- Accounting for all expandable fields (#54) [Miroslav Shubernetskiy]

2.1.7 0.10.1 (2019-10-30)

- Expandable serializer uses selectinload for *tomany (#53) [Miroslav Shubernetskiy]

2.1.8 0.10.0 (2019-08-31)

- Drop py2 support (#51) [Serkan Hosca]
- Pytest and black configs (#49) [Serkan Hosca]
- Add SearchFilter (#47) [Serkan Hosca]
- Use python/black (#46) [Serkan Hosca]

2.1.9 0.9.0 (2019-06-28)

- Drop enumfield and update importanize config (#45) [Serkan Hosca]

2.1.10 0.8.3 (2019-06-27)

- Fix module test runner target (#44) [Serkan Hosca]
- Switching to tox-travis and tox matrix (#43) [Miroslav Shubernetskiy]
- Run tests with pg (#42) [Serkan Hosca]
- Update pre-commit (#41) [Serkan Hosca]

2.1.11 0.8.2 (2019-02-11)

- Fix Unicode type column mapping (#40) [Serkan Hosca]

2.1.12 0.8.1 (2019-01-08)

- Allowing to overwrite fields and exclude on serializer init (#38) [Miroslav Shubernetskiy]

2.1.13 0.8.0 (2019-01-05)

- Grab composite meta info from parent model (#37) [Serkan Hosca]
- Coersion fixes from django-sorcery (#36) [Serkan Hosca]

2.1.14 0.7.20 (2018-12-13)

- Fix enum field custom kwargs (#35) [Serkan Hosca]

2.1.15 0.7.19 (2018-11-28)

- Pop widget from args (#34) [Serkan Hosca]

2.1.16 0.7.18 (2018-11-26)

- Stop using deprecated functions (#33) [Serkan Hosca]

2.1.17 0.7.17 (2018-11-24)

- Fix enum field and make it more generic (#32) [Serkan Hosca]

2.1.18 0.7.16 (2018-11-19)

- Fix composite source (#31) [Serkan Hosca]
- Remove pipenv (#30) [Serkan Hosca]

2.1.19 0.7.15 (2018-11-14)

- Handling ValidationError in update on set attribute (#28) [Miroslav Shubernetskiy]
- Bump pre-commit check versions (#27) [Serkan Hosca]

2.1.20 0.7.14 (2018-11-07)

- Fixing typo referencing session which does not exist (#26) [Miroslav Shubernetskiy]

2.1.21 0.7.13 (2018-11-06)

- Adding query_model hook (#24) [Miroslav Shubernetskiy]

2.1.22 0.7.12 (2018-11-05)

- Remove url default field from ModelSerializer (#25) [Serkan Hosca]
- Update lock. [Serkan Hosca]

2.1.23 0.7.11 (2018-11-01)

- Hook for how model is created (#22) [Miroslav Shubernetskiy]
- Fix serializer tests (#23) [Serkan Hosca]
- Relock (#20) [Serkan Hosca]
- Drop py3.5 build. [Serkan Hosca]

2.1.24 0.7.10 (2018-08-13)

- Partial by pk (#19) [Miroslav Shubernetskiy]
- Allowing to overwrite extra_kwargs in Serializer.__init__ (#18) [Miroslav Shubernetskiy]

2.1.25 0.7.9 (2018-08-08)

- ExpandableModelSerializer (#17) [Miroslav Shubernetskiy]
- Fixing saving serializer with source=* (#16) [Miroslav Shubernetskiy]

2.1.26 0.7.5 (2018-07-24)

- Correctly removing composite when validated data is None (#15) [Miroslav Shubernetskiy]

2.1.27 0.7.4 (2018-07-20)

- Fixing enum field choices (#14) [Miroslav Shubernetskiy]

2.1.28 0.7.3 (2018-07-16)

- Fixing updating model when field.field_name != field.source (#13) [Miroslav Shubernetskiy]
- Add nested update test (#12) [Serkan Hosca]

2.1.29 0.7.2 (2018-06-28)

- Merge pull request #10 from shosca/composite-labels. [Serkan Hosca]
- Fixing uri field for multiple pk models. fixed tests. [Miroslav Shubernetskiy]
- Honoring lookup_field iin querying model in generics.py when single pk. [Miroslav Shubernetskiy]
- Normalizing django validation errors in apis. [Miroslav Shubernetskiy]
- Fixing composite serializer field labels to use compose fields vs column names. [Miroslav Shubernetskiy]

2.1.30 0.7.1 (2018-06-26)

- Merge pull request #11 from shosca/relation-null-set. [Serkan Hosca]
- Fix many-to-one or one-to-one relation null set. [Serkan Hosca]

2.1.31 0.7.0 (2018-06-10)

- Merge pull request #9 from shosca/use-sorcery. [Serkan Hosca]
- Add sorcery as dependency. [Serkan Hosca]

2.1.32 0.6.2 (2018-02-23)

- Merge pull request #8 from shosca/packaging. [Serkan Hosca]
- Fix packaging. [Serkan Hosca]

2.1.33 0.6.1 (2018-01-08)

Fix

- Adjust build_nested_field signature. [Serkan Hosca]

Other

- Version 0.6.1. [Serkan Hosca]
- Merge pull request #7 from shosca/relation-info. [Serkan Hosca]

2.1.34 0.6.0 (2018-01-05)

- Version 0.6.0. [Serkan Hosca]
- Merge pull request #5 from shosca/build-field-signature. [Serkan Hosca]
- Add model_class to build_field. [Serkan Hosca]

2.1.35 0.5.6 (2017-12-21)

- Merge pull request #3 from nickswiss/enum-mapping. [Serkan Hosca]
- Adding enums to field mapping dict. [Nick Arnold]

2.1.36 0.5.5 (2017-11-02)

Fix

- Declared fields. [Serkan Hosca]

Other

- 0.5.5. [Serkan Hosca]
- Merge pull request #2 from shosca/fix-declared-fields. [Serkan Hosca]

2.1.37 0.5.4 (2017-10-23)

Fix

- Super for py2. [Serkan Hosca]

Refactor

- Separate out session flush. [Serkan Hosca]

2.1.38 0.5.2 (2017-10-21)

Fix

- Deepcopy composite and model serializers. [Serkan Hosca]

2.1.39 0.5.1 (2017-10-04)

Refactor

- Handle session passing around. [Serkan Hosca]

Other

- Merge pull request #1 from shosca/session-distribution. [Serkan Hosca]

2.1.40 0.5.0 (2017-10-03)

Refactor

- Make enums use values instead of names. [Serkan Hosca]
- Use relationship mapper to get target model class. [Serkan Hosca]

Other

- Add LICENSE. [Serkan Hosca]
- Pipfile lock. [Serkan Hosca]

2.1.41 0.4.3 (2017-07-06)

Fix

- Allow_null is not allowed in boolean fields. [Serkan Hosca]

2.1.42 0.4.2 (2017-07-02)

Fix

- Handle composite pks when one pk is None. [Serkan Hosca]

2.1.43 0.4.1 (2017-07-01)

Fix

- Nested model primary key field generation. [Serkan Hosca]

Other

- Fix readme. [Serkan Hosca]

2.1.44 0.4.0 (2017-06-28)

Fix

- Field label generation. [Serkan Hosca]

Refactor

- Lots of minor pylint and pycharm linter fixes. [Serkan Hosca]

Other

- Update gitchangelog.rc. [Serkan Hosca]

2.1.45 0.3.5 (2017-06-18)

Fix

- Increase coverage. [Serkan Hosca]

Refactor

- Dedup update attribute logic. [Serkan Hosca]
- Run pre-commit as part of build. [Serkan Hosca]

2.1.46 0.3.4 (2017-06-14)

Refactor

- Better route name handling and nullable boolean field tests. [Serkan Hosca]

Documentation

- Update gitchangelog config. [Serkan Hosca]

2.1.47 0.3.3 (2017-06-13)

Fix

- Add pipenv for setup. [Serkan Hosca]

Documentation

- Fix versioning. [Serkan Hosca]

2.1.48 0.3.2 (2017-06-13)

Fix

- Stop passing around is_nested and fix autoincrement value check. [Serkan Hosca]

2.1.49 0.3.1 (2017-06-11)

- Delete tests and coverall config. [Serkan Hosca]

2.1.50 0.3.0 (2017-06-11)

Fix

- Nested list serializer flags. [Serkan Hosca]
- Generic destroy with sqlalchemy. [Serkan Hosca]
- Handle autoincrement and nested update with existing instance. [Serkan Hosca]

Refactor

- Model_info changes and added docstrings. [Serkan Hosca]

Other

- Initial doc setup. [Serkan Hosca]

2.1.51 0.2.1 (2017-06-10)

- Initial doc setup. [Serkan Hosca]

2.1.52 0.2.0 (2017-06-10)

- Refactor field mapping and object fetching and more tests. [Serkan Hosca]

2.1.53 0.1.4 (2017-06-09)

- Respect allow_null. [Serkan Hosca]

2.1.54 0.1.2 (2017-06-08)

- Mark all columns read only when allow_nested_updates is false. [Serkan Hosca]

2.1.55 0.1.1 (2017-06-07)

- Fix composite serializer. [Serkan Hosca]

2.1.56 0.1.0 (2017-06-06)

- Add more tests and generic api fixes. [Serkan Hosca]

2.1.57 0.0.6 (2017-06-05)

- Add missing dep and add pypi badge. [Serkan Hosca]
- Add more tests for composite routes. [Serkan Hosca]

2.1.58 0.0.5 (2017-06-05)

- Add route tests. [Serkan Hosca]

2.1.59 0.0.4 (2017-06-05)

- Add pre-commit. [Serkan Hosca]
- Move GenericAPIView. [Serkan Hosca]
- Fix Readme. [Serkan Hosca]

2.1.60 0.0.2 (2017-06-02)

- Fix setup publish and make clean. [Serkan Hosca]
- Added viewsets and version bump. [Serkan Hosca]
- Update readme. [Serkan Hosca]

2.1.61 0.0.1 (2017-06-02)

- Fix readme. [Serkan Hosca]
- Added initial readme. [Serkan Hosca]
- Add travis. [Serkan Hosca]
- Initial commit. [Serkan Hosca]

2.2 API Documentation

2.2.1 rest_witchcraft

rest_witchcraft package

Submodules

rest_witchcraft.field_mapping module

Field mapping from SQLAlchemy type's to DRF fields.

`rest_witchcraft.field_mapping.`**`get_detail_view_name`**(*model*)
Given a model class, return the view name to use for URL relationships that rever to instances of the model.

`rest_witchcraft.field_mapping.`**`get_field_type`**(*column*)
Returns the field type to be used determined by the sqlalchemy column type or the column type's python type.

`rest_witchcraft.field_mapping.`**`get_url_kwargs`**(*model*)
Gets kwargs for the UriField.

rest_witchcraft.fields module

Some SQLAlchemy specific field types.

class `rest_witchcraft.fields.`**`CharMappingField`**(***kwargs*)
Bases: `rest_framework.fields.DictField`

Used for Postgresql HSTORE columns for storing key-value pairs.

`child = CharField(allow_null=True)`

class `rest_witchcraft.fields.`**`HyperlinkedIdentityField`**(*view_name=None*, ***kwargs*)
Bases: `rest_framework.relations.HyperlinkedIdentityField`

`get_url`(*obj*, *view_name*, *request*, *format*)
Given an object, 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.

class `rest_witchcraft.fields.`**`ImplicitExpandableListField`**(***kwargs*)
Bases: `rest_framework.fields.ListField`

List field which implicitly expands parent field when child field is expanded assuming parent field is also expandable by being one of the choices.

`to_internal_value`(*data*)
List of dicts of native values <- List of dicts of primitive datatypes.

class `rest_witchcraft.fields.`**`SkippableField`**(**, read_only=False, write_only=False, required=None, default=<class 'rest_framework.fields.empty'>, initial=<class 'rest_framework.fields.empty'>, source=None, label=None, help_text=None, style=None, error_messages=None, validators=None, allow_null=False*)

Bases: rest_framework.fields.Field

Field which is always skipped on to_representation.

Useful when used together with ExpandableModelSerializer since it allows to completely skip expandable field when it is not being expanded. Especially useful for OneToMany relations since by default nested serializer cannot be rendered as none of the PKs of the "many" items are known unlike ManyToOne when nested serializer can be rendered with PK. For example:

```
class FooSerializer(ExpandableModelSerializer):
    bar = BarSerializer(many=True)

    class Meta:
        model = Foo
        session = session
        fields = "__all__"
        expandable_fields = {
            "bar": SkippableField()
        }
```

get_attribute(*instance*)
Given the *outgoing* object instance, return the primitive value that should be used for this field.

class rest_witchcraft.fields.**UriField**(*view_name=None*, ***kwargs*)
Bases: *rest_witchcraft.fields.HyperlinkedIdentityField*

Represents a uri to the resource.

get_url(*obj*, *view_name*, *request*, *format*)
Same as basic HyperlinkedIdentityField except return uri vs full url.

rest_witchcraft.filters module

Provides generic filtering backends that can be used to filter the results returned by list views.

class rest_witchcraft.filters.**SearchFilter**
Bases: rest_framework.filters.BaseFilterBackend

filter_queryset(*request*, *queryset*, *view*)
Return a filtered queryset.

get_expression(*model*, *field*, *term*)

get_schema_fields(*view*)

get_schema_operation_parameters(*view*)

get_search_fields(*view*, *request*)

get_search_terms(*request*)

lookup_prefixes = {'': <function SearchFilter.<lambda>>, '=': <function SearchFilter

search_description = 'A search term.'

search_param = 'search'

search_title = 'Search'

template = 'rest_framework/filters/search.html'

to_html(*request*, *queryset*, *view*)

rest_witchcraft.generics module

class rest_witchcraft.generics.**GenericAPIView**(***kwargs*)
Bases: rest_framework.generics.GenericAPIView

Base class for sqlalchemy specific views.

classmethod get_model()
Returns the model class.

get_object()
Returns the object the view is displaying.

We ignore the *lookup_field* and *lookup_url_kwarg* values only when tere are multiple primary keys

get_session()
Returns the session.

rest_witchcraft.mixins module

class rest_witchcraft.mixins.**DestroyModelMixin**
Bases: rest_framework.mixins.DestroyModelMixin

Deletes a model instance.

perform_destroy(*instance*)

class rest_witchcraft.mixins.**ExpandableQuerySerializerMixin**
Bases: *rest_witchcraft.mixins.QuerySerializerMixin*

Adds expandable query serializer validation logic to viewset as well as automatic eager load of expanded fields on the serializer.

The query serializer is expected to be generated by *rest_witchcraft.serializers.ExpandableModelSerializer.get_query_serializer_class()*.

expand_queryset(*queryset*, *values*)

get_queryset()

get_serializer_context()

class rest_witchcraft.mixins.**QuerySerializerMixin**
Bases: object

Adds query serializer validation logic to viewset.

Query will be validated as part of query viewset initialization therefore query will be validated before any of the viewset actions are executed.

In addition query serializer will be included in serializer context for standard viewset serializers. That

check_query()

get_query_serializer(**args*, ***kwargs*)

get_query_serializer_class()

get_query_serializer_context()

initial(*request*, **args*, ***kwargs*)

query_serializer

query_serializer_class = None

class rest_witchcraft.mixins.**ToLoadField**(*field*, *direction*)
Bases: tuple

direction
Alias for field number 1

field
Alias for field number 0

rest_witchcraft.routers module

class rest_witchcraft.routers.**DefaultRouter**(**args*, ***kwargs*)
Bases: rest_framework.routers.DefaultRouter

get_default_base_name(*viewset*)

get_default_basename(*viewset*)

get_lookup_regex(*viewset*, *lookup_prefix=''*)
Given a viewset, return the portion of the url regex that is used to match against a single instance.

Can be overwritten by providing a *lookup_url_regex* on the viewset.

rest_witchcraft.serializers module

class rest_witchcraft.serializers.**BaseSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)
Bases: rest_framework.serializers.Serializer

build_standard_field(*field_name*, *column_info*)
Create regular model fields.

build_standard_field_kwargs(*field_name*, *field_class*, *column_info*)
Analyze model column to generate field kwargs.

create(*validated_data*)

get_field_type(*column_info*)
Returns the field type to be used determined by the sqlalchemy column type or the column type's python type.

include_extra_kwargs(*kwargs*, *extra_kwargs=None*)
Include any 'extra_kwargs' that have been included for this field, possibly removing any incompatible existing keyword arguments.

is_nested

serializer_choice_field
alias of rest_framework.fields.ChoiceField

update(*instance*, *validated_data*)

update_attribute(*instance*, *field*, *value*)
Performs update on the instance for the given field with value.

class rest_witchcraft.serializers.**CompositeSerializer**(**args*, ***kwargs*)
Bases: *rest_witchcraft.serializers.BaseSerializer*

This class is useful for generating a serializer for sqlalchemy's *composite* model attributes.

create(*validated_data*)

get_fields()
Return the dict of field names -> field instances that should be used for *self.fields* when instantiating the serializer.

get_object(*validated_data*, *instance=None*)

perform_update(*instance*, *validated_data*, *errors*)

update(*instance*, *validated_data*)

class rest_witchcraft.serializers.**ExpandableModelSerializer**(**args*, ***kwargs*)
Bases: *rest_witchcraft.serializers.ModelSerializer*

Same as `ModelSerializer` but allows to conditionally recursively expand specific fields.

Serializer by default renders with all fields collapsed however validates data with expanded fields.

To expand fields, either:

- `request.GET` should request to expand field by `?expand=<field>`. Field names can be recursive `?expand=<field>__<nested_field>`.
- One of expandable fields was updated which will cause `to_representation()` to render expanded field.

By default serializer should define "expanded" fields. `ModelSerializer` already does it by default for all relations. This allows introspection of not rendered serializer to pick up all fields. This is especially useful when generating schema for the serializer such as for coreapi docs. Collapsed fields are specified in `Meta.expandable_fields` where keys are field names and values are replacement field instances.

In addition expandable query key can be specified via `Meta.expandable_query_key`.

For example:

```
class BarJustIDSerializer(Serializer):
    id = serializers.IntegerField(source="bar_id")

    class Meta:
        model = Bar
        session = session
        fields = ["id"]

class FooSerializer(ExpandableModelSerializer):
    class Meta:
        model = Foo
        session = session
        exclude = ["bar_id"]
        expandable_fields = {
            "bar": BarJustIDSerializer(source="*", read_only=True)
        }
        expandable_query_key = "include"
```

Additionally, query serializer can be autogenerated to be used to either validate request query or generate documentation:

```
FooSerializer().get_query_serializer_class()
FooSerializer().get_query_serializer_class(exclude=["bar"])
FooSerializer().get_query_serializer_class(disallow=["bar"])
```

Exclude excludes given expand paths. Useful for generating documentation.

Disallow leaves the expand field in serializer however removes given paths from valid choices. Useful for validating user input within viewset.

get_query_serializer_class(*exclude=()*, *disallow=()*, *implicit_expand=True*)
Generate serializer to either validate request querystring or generate documentation.

to_representation(*instance*)
Switch expandable fields to collapsed fields if not explicitly asked to be expanded or field was updated.

update_attribute(*instance*, *field*, *value*)
Mark which attributes are updated so that during representation of the resource, we can expand those fields even if not explicitly asked for.

Fields are marked on root serializer since child serializers should not contain any state.

class rest_witchcraft.serializers.**ModelSerializer**(**args*, ***kwargs*)
Bases: *rest_witchcraft.serializers.BaseSerializer*

ModelSerializer is basically like a drf model serializer except that it works with sqlalchemy models:

- A set of default fields are automatically populated by introspecting a sqlalchemy model
- Default *.create()* and *.update()* implementations provided by mostly reducing the problem to update.

The process of automatically determining a set of serializer fields is based on the model's fields, components and relationships.

If the *ModelSerializer* does not generate the set of fields that you need, you can explicitly declare them.

build_composite_field(*field_name*, *composite*)
Builds a *CompositeSerializer* to handle composite attribute in model.

build_field(*field_name*, *info*, *model_class*, *nested_depth*)
Return a field or a nested serializer for the field name.

build_nested_field(*field_name*, *relation_info*, *nested_depth*)
Builds nested serializer to handle relationshipped model.

build_primary_key_field(*field_name*, *column_info*)
Builds a field for the primary key of the model.

build_property_field(*field_name*, *info*)

build_unknown_field(*field_name*, *info*)
Raise an error on any unknown fields.

build_url_field(*field_name*, *info*)
Create a field representing the object's own URL.

create(*validated_data*)
Creates a model instance using validated_data.

create_model(*validated_data*)
Hook to allow to customize how model is created in create flow.

default_error_messages = {'not_found': 'No instance found with primary keys'}

get_default_field_names(*declared_fields*, *info*)
Return the default list of field names that will be used if the *Meta.fields* option is not specified.

get_extra_kwargs(***additional_kwargs*)
Return a dictionary mapping field names to a dictionary of additional keyword arguments.

get_field_names(*declared_fields*, *info*)
Returns the list of all field names that should be created when instantiating this serializer class.

This is based on the default set of fields, but also takes into account the *Meta.fields* or *Meta.exclude* options if they have been specified.

get_fields()
Return the dict of field names -> field instances that should be used for *self.fields* when instantiating the serializer.

get_nested_relationship_fields(*relation_info*, *depth*)
Get the field names for the nested serializer.

get_object(*validated_data*, *instance=None*)
Returns model object instance using the primary key values in the *validated_data*.

If the instance is not found, depending on serializer's *allow_create* value, it will create a new model instance or raise an error.

get_primary_keys(*validated_data*)
Returns the primary key values from validated_data.

get_relationship_kwargs(*relation_info*, *depth*)
Figure out the arguments to be used in the *NestedSerializer* for the relationship.

model

perform_flush()
Perform session flush changes.

perform_update(*instance*, *validated_data*, *errors*)
The main nested update logic implementation using nested fields and serializer.

query_model(*pks*)
Hook to allow to customize how model is queried when serializer is nested and needs to query the model by its primary keys.

queryset

save(***kwargs*)
Save and return a list of object instances.

serializer_url_field
alias of *rest_witchcraft.fields.UriField*

session

to_internal_value(*data*)
Same as in DRF but also handle `partial_by_pk` by making all non- pk fields optional.

Even though flag name implies it will make serializer partial, that is currently not possible in DRF as partial flag is checked on root serializer within serializer validation loops. As such, individual serializers cannot be marked partial. Therefore when flag is provided and primary key is provided in validated data, we physically mark all other fields as not required to effectively make them partial without using `partial` flag itself. To make serializer behave more or less like real partial serializer, only passed keys in input data are preserved in validated data. If they are not stripped, it is possible to remove some existing data.

update(*instance*, *validated_data*)
Updates an existing model instance using validated_data with suspended autoflush.

url_field_name = None

rest_witchcraft.utils module

rest_witchcraft.utils.**django_to_drf_validation_error**(*e*)

rest_witchcraft.viewsets module

class rest_witchcraft.viewsets.**ExpandableModelViewSet**(***kwargs*)
Bases: *rest_witchcraft.mixins.ExpandableQuerySerializerMixin*, *rest_witchcraft.viewsets.ModelViewSet*

A viewset that provides automatically eagerloadsany subfields that are expanded via querystring.

For queryset to be expanded, either *rest_witchcraft.serializers.ExpandableModelSerializer* needs to be used in `serializer_class` or `query_serializer_class` can be manually provided.

class rest_witchcraft.viewsets.**GenericViewSet**(***kwargs*)
Bases: `rest_framework.viewsets.ViewSetMixin`, *rest_witchcraft.generics.GenericAPIView*

The GenericViewSet class does not provide any actions by default, but does include the base set of generic view behavior, such as the *get_object* and *get_queryset* methods.

class rest_witchcraft.viewsets.**ModelViewSet**(***kwargs*)
Bases: `rest_framework.mixins.CreateModelMixin`, `rest_framework.mixins.RetrieveModelMixin`, `rest_framework.mixins.UpdateModelMixin`, *rest_witchcraft.mixins.DestroyModelMixin*, `rest_framework.mixins.ListModelMixin`, *rest_witchcraft.viewsets.GenericViewSet*

A viewset that provides default *create()*, *retrieve()*, *update()*, *partial_update()*, *destroy()* and *list()* actions.

class rest_witchcraft.viewsets.**ReadOnlyViewModelViewSet**(***kwargs*)
Bases: `rest_framework.mixins.RetrieveModelMixin`, `rest_framework.mixins.ListModelMixin`, *rest_witchcraft.viewsets.GenericViewSet*

A viewset that provides default *list()* and *retrieve()* actions.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

G

H

I

L