djangorest$_{a}lchemy$Documentation

Release 0.1.0

Ashish Gore

September 03, 2014

Contents

1 djangorest-alchemy **3**
- 1.1 Features . . . 3
- 1.2 Install dependencies . . . 3
- 1.3 Run tests . . . 3
- 1.4 Usage . . . 3
- 1.5 Advanced Usage . . . 4
- 1.6 Examples . . . 5

2 Installation **7**

3 API Documentation **9**
- 3.1 djangorest_alchemy package . . . 9

4 Contributing **15**
- 4.1 Types of Contributions . . . 15
- 4.2 Get Started! . . . 16
- 4.3 Pull Request Guidelines . . . 16

5 Credits **19**
- 5.1 Development Lead . . . 19
- 5.2 Contributors . . . 19

6 History **21**
- 6.1 0.1.0 (2014-09-03) . . . 21

7 Indices and tables **23**

Python Module Index **25**

Contents:

CHAPTER 1

djangorest-alchemy

A library to integrate the awesome frameworks Django REST Framework and SQLAlchemy

- Free software: MIT license
- Supports SQLAlchemy 0.7.8 and above

1.1 Features

- Provides GET verb implementation for SQLAlchemy models
- List, filter and paginate multiple rows
- Fetch single object with nested objects as complete URIs
- Supports multiple primary keys
- Provides ability to use 'Manager' like classes to work with SQLAlchemy models
- Supports both Declarative and Classical styles

1.2 Install dependencies

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

1.3 Run tests

```
` make test `
```

1.4 Usage

Getting Started

Assuming you have a SQLAlchemy model defined as below:

```
class DeclarativeModel(Base):
    __tablename__ = 'test_model'

    declarativemodel_id = Column(INTEGER, primary_key=True)
    field = Column(String)
    datetime = Column(DateTime, default=datetime.datetime.utcnow)
    floatfield = Column(Float)
    bigintfield = Column(BigInteger)
    child_model = relationship(ChildModel, uselist=False, primaryjoin=
                               (declarativemodel_id == ChildModel.parent_id))
```

Define the 'manager' class to work on above model:

```
class DeclarativeModelManager(SessionMixin, AlchemyModelManager):
    model_class = DeclarativeModel
```

SessionMixin just provides a convenient way to initialize the SQLAlchemy session. You can achieve the same by definining __init__ and setting ‘‘‘self.session‘‘‘ instance

Define the Django REST viewset and specify the manager class:

```
class DeclModelViewSet(AlchemyModelViewSet):
    manager_class = DeclarativeModelManager
```

Finally, register the routers as you would normally do using Django REST:

```
viewset_router = routers.SimpleRouter()
viewset_router.register(r'api/declmodels', DeclModelViewSet,
                        base_name='test-decl')
```

Pagination

Pagination works exactly like Django REST Framework (and Django). Provided your viewset has the `'paginate_by'` field set, pass page number in querystring:

```
class ModelViewSet(AlchemyModelViewSet):
    paginate_by = 25
```

- 5th page `'curl -v http://server/api/declmodels/?page=5'`
- Last page `'curl -v http://server/api/declmodels/?page=last'`
- First page `'curl -v http://server/api/declmodels/'`

Filters

Filters work exactly like Django REST Framework. Pass the field value pair in querystring.

`'curl -v http://server/api/declmodels/?field=value'`

1.5 Advanced Usage

Multiple primary keys

To use some sort of identifier in the URI, the library tries to use the following logic.

1. If a single primary key is found, use it! That was simple..
2. For multiple keys, try to find a field with convention 'model_id'
3. If not found, see if the model has 'pk_field' class variable

4. If not found, raise KeyNotFoundException

In addition, to support multiple primary keys which cannot be accomodated in the URI, the viewset needs to override the `'get_other_pks'` method and return back dictionary of primary keys. Example:

```
class ModelViewSet(AlchemyModelViewSet):
    manager_class = ModelManager
    def get_other_pks(self, request):
        pks = {
            'pk1': request.META.get('PK1'),
            'pk2': request.META.get('PK2'),
        }
        return pks
```

Manager factory

The base AlchemyModelViewSet viewset provides a way to override the instantiation of the manager. Example:

```
class ModelViewSet(AlchemyModelViewSet):
    def manager_factory(self, *args, **kwargs):
        return ModelManager()
```

Nested Models

This library recommends using the drf-nested-routers for implementing nested child models. Example:

```
child_router = routers.NestedSimpleRouter(viewset_router, r'api/declmodels',
                                          lookup='declmodels')
```

For more details, refer to the drf-nested-routers documentation.

Custom methods

DRF allows to add custom methods other than the default list, retrieve, create, update and destroy using the @action decorator. However, if you have managers, then you can simply provide action methods on the manager and specify the action methods using *action_methods* field The methods have to return back appropriate status per below map.

STATUS_CODES = { 'created': status.HTTP_201_CREATED, 'updated': status.HTTP_200_OK, 'accepted': status.HTTP_202_ACCEPTED

}

class MyManager(AlchemyModelManager): action_methods = {'do_something': ['POST']}

def do_something(self, data, pk=None, **kwargs): # data is actual payload return {'status': 'created'}

class ModelViewSet(AlchemyModelViewSet): manager_class = MyManager

```
'curl -X POST http://server/api/declmodels/1/do_something/'
```

Read-only API

If you need only the GET method, and do not wish to expose/support POST/PUT/DELETE then you can use the *djangorest_alchemy.routers.ReadOnlyRouter* instead of the DefaultRouter

1.6 Examples

The examples folder demonstrates a real-world example using Cars and Parts as the object models.

Run the following command just as you would normally run a Django project:

```
` cd examples python manage.py runserver --settings=settings `
```

Then type the following in your favorite browser:

```
` http://localhost/api/cars/ `
```

CHAPTER 2

Installation

At the command line:

```
$ easy_install djangorest-alchemy
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv djangorest-alchemy
$ pip install djangorest-alchemy
```

CHAPTER 3

API Documentation

3.1 djangorest_alchemy package

3.1.1 Submodules

3.1.2 djangorest_alchemy.fields module

Relationship field

class djangorest_alchemy.fields.**AlchemyRelatedField**(**args*, ***kwargs*)
Bases: rest_framework.relations.RelatedField

to_native(*obj*)

class djangorest_alchemy.fields.**AlchemyUriField**(**args*, ***kwargs*)
Bases: rest_framework.relations.RelatedField

to_native(*obj*)

3.1.3 djangorest_alchemy.inspector module

Functions to reflect over SQLAlchemy models

exception djangorest_alchemy.inspector.**KeyNotFoundException**
Bases: exceptions.Exception

Primary key not found exception

djangorest_alchemy.inspector.**class_keys**(*cls*)
This is a utility function to get the attribute names for the primary keys of a class

>>> class_keys(Deal) # >>> ('dealer_code', 'deal_jacket_id', 'deal_id')

djangorest_alchemy.inspector.**primary_key**(*cls*)
Utility function to get the primary key of the class. In case of multiple primary keys, use the <classname>_id convention

3.1.4 djangorest_alchemy.managers module

Base for interfacing with SQLAlchemy Provides the necessary plumbing for CRUD using SA session

class djangorest_alchemy.managers.**AlchemyModelManager**(**args*, ***kwargs*)
Bases: object

list(*other_pks=None*, *filters=None*)
List returns back list of URI In case of multiple pks, We guess the pk by using '<modelname>_id' as the field convention

retrieve(*pks*, *other_pks=None*)
Retrieve fetches the object based on the following pk logic: if 'other' pks are not found, just use the pk list (coming from URLS) assuming their order is already correct if 'other' pks are found, then use the class keys to get the correct order of pks, look them up

3.1.5 djangorest_alchemy.mixins module

class djangorest_alchemy.mixins.**ManagerMeta**
Bases: type

Meta class to read action methods from manager and attach them to viewset This allows us to directly call manager methods without writing any action methods on viewsets

class djangorest_alchemy.mixins.**ManagerMixin**
Bases: object

Manager mixin allows to use a manager class to provide the actual CRUD implementation in addition to providing action methods

Example:

```
class MyManager(AlchemyModelManager):
    action_methods = {'my_method': ['POST']}

    def my_method(self, data, pk=None, **kwargs):
        # data is actual payload
        return {'status': 'created'}

class MyViewSet(viewset.Viewsets, ManagerMixin):
    manager_class = MyManager
```

manager_factory(**args*, ***kwargs*)
Factory method for instantiating manager class Override to return back your instance

class djangorest_alchemy.mixins.**MultipleObjectMixin**
Bases: object

SQLAlchemy analog to Django's MultipleObjectMixin.

allow_empty = True

filter_query_object(*query_object*)
Generic filtering.

This is a stub and has yet to be implemented.

get_allow_empty()
Returns True to display empty lists, False to 404.

get_page(*queryset*)
Add the object list to the template context.

get_paginate_by(*query_object*)
Get the number of items to paginate by. None for no pagination.

get_paginator(*query_object*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)
Get a paginator instance.

The class used is overridable by setting the paginator_class attribute.

paginate_by = None

paginate_query_object(*query_object*, *page_size*)
Paginate the query object.

paginator_class
alias of `Paginator`

query_object = None

djangorest_alchemy.mixins.**make_action_method**(*name*, *methods*, ***kwargs*)

3.1.6 djangorest_alchemy.routers module

class djangorest_alchemy.routers.**ReadOnlyRouter**(*trailing_slash=True*)
Bases: `rest_framework.routers.DefaultRouter`

A router for read-only APIs, which USES trailing slashes.

routes = [Route(url='^{prefix}{trailing_slash}$', mapping={'get': 'list'}, name='{basename}-list', initkwargs={'suffix':

3.1.7 djangorest_alchemy.serializers module

Base AlchemyModelSerializer which provides the mapping between SQLALchemy and DRF fields to serialize/deserialize objects

class djangorest_alchemy.serializers.**AlchemyListSerializer**(**args*, ***kwargs*)
Bases: djangorest_alchemy.serializers.AlchemyModelSerializer

base_fields = {}

get_default_fields()

class djangorest_alchemy.serializers.**AlchemyModelSerializer**(**args*, ***kwargs*)
Bases: `rest_framework.serializers.Serializer`

Alchemy -> DRF field serializer

base_fields = {}

field_mapping = {<class 'sqlalchemy.types.INTEGER'>: <class 'rest_framework.fields.IntegerField'>, <class 'sqlalch

get_default_fields()

3.1.8 djangorest_alchemy.settings module

3.1.9 djangorest_alchemy.viewsets module

Base AlchemyViewSet which provides the necessary plumbing to interface with AlchemyModelSerializer and AlchemyModelManager

class djangorest_alchemy.viewsets.**AlchemyModelViewSet**(***kwargs*)
Bases: djangorest_alchemy.mixins.MultipleObjectMixin, djangorest_alchemy.mixins.ManagerMixin, `rest_framework.viewsets.ViewSet`

Generic SQLAlchemy viewset which calls methods over the specified manager_class and uses specified serializer_class

create(*request*)

destroy(*request*, *pk=None*)

get_other_pks(*request*)
Return default empty {} Override to return back your primary keys from other source (possibly from headers)

Parameters **request** – REST request object

Returns dict

get_pks(*request*, ***kwargs*)
Return list of pks from the keyword args e.g. /models/pk1/childmodel/pk2 return back [pk1, pk2]

Parameters **request** – REST request object

Kwargs kwargs URI keyword args

Returns List e.g. [pk1, pk2]

list(*request*, ***kwargs*)
Returns back serialized list of objects URIs in the *results* key

Returns

json {

"results": [

{ "href": "http://server/api/models/pk/"

}

]

}

Note:

```
* URI contains the same pk field
* Complete URI with server/port is returned back
```

retrieve(*request*, ***kwargs*)
Retrieve returns back serialized object.

Returns

json {

"href": "http://server/api/models/pk/", "field": "value" "childmodel": "http://serv/api/parentmodels/pk/childmodels/pk"

}

Note:

```
As of now, only SQLAlchemy mapper properties are returned
No other fields or properties are serialized. You will need
to override retrieve and provide your own implementation to
query those additional properties for now.
```

serializer_factory(*multiple*, *queryset*, *model_class*, *context*)
Factory method to instantiate appropriate serializer class Override to return back your instance

update(*request*, *pk=None*)

3.1.10 Module contents

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/Dealertrack/djangorest-alchemy/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

djangorest-alchemy could always use more documentation, whether as part of the official djangorest-alchemy docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Dealertrack/djangorest-alchemy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *djangorest-alchemy* for local development.

1. Fork the *djangorest-alchemy* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/djangorest-alchemy.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv djangorest-alchemy
   $ cd djangorest-alchemy/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 djangorest-alchemy tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/iagore/djangorest-alchemy/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

5.1 Development Lead

- Ashish Gore <ashish.gore@dealertrack.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2014-09-03)

- First release

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*

d

djangorest_alchemy, 13
djangorest_alchemy.fields, 9
djangorest_alchemy.inspector, 9
djangorest_alchemy.managers, 9
djangorest_alchemy.mixins, 10
djangorest_alchemy.routers, 11
djangorest_alchemy.serializers, 11
djangorest_alchemy.settings, 11
djangorest_alchemy.viewsets, 11