django-tastypie-mongoengine Documentation

Release 0.4.6

wlan slovenija

Jul 27, 2017

Contents

This Django application provides MongoEngine support for Tastypie.

CHAPTER 1

Contents

Installation

Using pip simply by doing:

```
pip install django-tastypie-mongoengine
```

or by installing from source with:

```
python setup.py install
```

In your settings.py add `tastypie` and `tastypie_mongoengine` to `INSTALLED_APPS`:

```
INSTALLED_APPS += (
    'tastypie',
    'tastypie_mongoengine',
)
```

You must also connect MongoEngine to the database:

```
MONGO_DATABASE_NAME = 'database'

import mongoengine
mongoengine.connect(MONGO_DATABASE_NAME)
```

Tests

You can run tests by doing:

```
./setup.py test
```

This will install necessary dependencies as well. In `tests` subdirectory there is a testing Django project with tests. You can check it to for usage examples, as well.

To test different versions at the same time, Travis CI is used.

Usage

Usage for simple cases is very similar as with Tastypie. You should read their tutorial first.

The main difference is when you are defining API resource files. There you must use `MongoEngineResource` instead of `ModelResource`.

Simple Example

```
from tastypie import authorization
from tastypie_mongoengine import resources
from test_app import documents

class PersonResource(resources.MongoEngineResource):
    class Meta:
        queryset = documents.Person.objects.all()
        allowed_methods = ('get', 'post', 'put', 'delete')
        authorization = authorization.Authorization()
```

Defining fields

Most document fields are automatically mapped to corresponding Tastypie fields but some are not. Of course, you can also manually define (override) those automatically mapped fields if, for example, you want to define some read-only.

Warning: When manually defining resource fields be careful to properly map MongoEngine attributes to Tastypie attributes. For example, `required` and `null` attributes are inversed in meaning, but both are by default `False`.

Warning: When manually defining resource fields be careful not to forget to set `attribute` to document's field name. It is not set automatically and if it is not set it is assumed that you will be processing this field manually (in for example, resource's `hydrate` method).

Some fields cannot be mapped automatically so you have to define them manually. Like *related and embedded fields*, but special fields like SequenceField as well:

```
sequence_field = tastypie_fields.IntegerField(attribute='sequence_field')
```

EmbeddedDocument

When you are using `EmbeddedDocument` in your MongoEngine documents, you must define `object_class` in Meta class of your resource declaration instead of queryset:

```
class EmbeddedPersonResource(resources.MongoEngineResource):
    class Meta:
        object_class = documents.EmbeddedPerson
    ...
```

When you are using normal MongoEngine `Document` you can use `queryset` or `object_class`.

Related and Embedded Fields

Related and embedded fields have to be defined manually always.

ReferenceField

```
from tastypie_mongoengine import fields

class CustomerResource(resources.MongoEngineResource):
    person = fields.ReferenceField(to='test_project.test_app.api.resources.
↪PersonResource', attribute='person', full=True)
    ...
```

EmbeddedDocumentField

Embeds a resource inside another resource just like you do in MongoEngine:

```
from tastypie_mongoengine import fields

class EmbeddedDocumentFieldTestResource(resources.MongoEngineResource):
    customer = fields.EmbeddedDocumentField(embedded='test_project.test_app.api.
↪resources.EmbeddedPersonResource', attribute='customer')
    ...
```

EmbeddedListField

If you are using `ListField` containing a `EmbeddedDocumentField` in MongoEngine document, it should be mapped to `EmbeddedListField`:

```
from tastypie_mongoengine import fields

class EmbeddedListFieldTestResource(resources.MongoEngineResource):
    embeddedlist = fields.EmbeddedListField(of='test_project.test_app.api.resources.
↪EmbeddedPersonResource', attribute='embeddedlist', full=True, null=True)
    ...
```

`EmbeddedListField` also exposes its embedded documents as subresources, so you can access them directly. For example, URI of the first element of the list above could be `/api/v1/embeddedlistfieldtest/4fb88d7549902817fe000000/embeddedlist/0/`. You can also manipulate subresources in the same manner as resources themselves.

ReferencedListField

If you are using `ListField` containing a `ReferenceField` in MongoEngine document, it should be mapped to `ReferencedListField`:

```
from tastypie_mongoengine import fields

class ReferencedListFieldTestResource(resources.MongoEngineResource):
    referencedlist = fields.ReferencedListField(of='test_project.test_app.api.
↪resources.PersonResource', attribute='referencedlist', full=True, null=True)
    ...
```

Polymorphism

MongoEngine supports document inheritance and you can normally add such inherited documents to your RESTful API. But sometimes it is useful to have only one API endpoint for family of documents so that they are all listed together but that you can still create different variations of the document. To enable this, you have to define mapping between types and resources. For example, if documents are defined as:

```
class Person(mongoengine.Document):
    meta = {
        'allow_inheritance': True,
    }

    name = mongoengine.StringField(max_length=200, required=True)
    optional = mongoengine.StringField(max_length=200, required=False)

class StrangePerson(Person):
    strange = mongoengine.StringField(max_length=100, required=True)
```

You might define your resources as:

```
class StrangePersonResource(resources.MongoEngineResource):
    class Meta:
        queryset = documents.StrangePerson.objects.all()

class PersonResource(resources.MongoEngineResource):
    class Meta:
        queryset = documents.Person.objects.all()
        allowed_methods = ('get', 'post', 'put', 'patch', 'delete')
        authorization = authorization.Authorization()

        polymorphic = {
            'person': 'self',
            'strangeperson': StrangePersonResource,
        }
```

Use `self` keyword to refer back to the current (main) resource. Authorization and other similar parameters are still taken from the main resource.

Then, when you want to use some other type than default, you should pass it as an additional parameter `type` to `Content-Type` in your payload request:

```
Content-Type: application/json; type=strangeperson
```

Alternatively, you can pass a query string parameter.

All this works also for embedded documents in list.

Polymorphic resource_uri

By default, polymorphic resources are exposed through the API with a common `resource_uri`.

In the previous case, `PersonResource` and `StrangePersonResource` are both exposed through the `/<api_version>/person/` resource URI.

But in some cases, you may want to expose your resources through the polymorphic resource uri. To use this behaviour, you should set the `prefer_polymorphic_resource_uri` meta variable to `True`.

You might define your resources as:

```
class IndividualResource(resources.MongoEngineResource):
    class Meta:
        queryset = documents.Individual.objects.all()
        allowed_methods = ('get', 'post', 'put', 'patch', 'delete')
        authorization = tastypie_authorization.Authorization()
        paginator_class = paginator.Paginator

class CompanyResource(resources.MongoEngineResource):
    class Meta:
        queryset = documents.Company.objects.all()
        allowed_methods = ('get', 'post', 'put', 'patch', 'delete')
        authorization = tastypie_authorization.Authorization()
        paginator_class = paginator.Paginator

class ContactResource(resources.MongoEngineResource):
    class Meta:
        queryset = documents.Contact.objects.all()
        allowed_methods = ('get', 'post', 'put', 'patch', 'delete')
        authorization = tastypie_authorization.Authorization()

        prefer_polymorphic_resource_uri = True
        polymorphic = {
            'individual': IndividualResource,
            'company': CompanyResource,
        }
```

You might now reference both resources:

```
class ContactGroupResource(resources.MongoEngineResource):
    contacts = fields.ReferencedListField(of='test_project.test_app.api.resources.
↪ContactResource', attribute='contacts', null=True)

    class Meta:
        queryset = documents.ContactGroup.objects.all()
        allowed_methods = ('get', 'post', 'put', 'patch', 'delete')
        authorization = tastypie_authorization.Authorization()
```

And for each contact listed, the:

- `IndividualResource` would be dehydrated to `/<api_version>/individual/<id>/`
- `CompanyResource` to `/<api_version>/company/<id>/`

Warning: The `ContactResource` could not be registered but be careful to register all the resources present in the `polymorphic` *dict* otherwise the dehydrated `resource_uri` will point to the parent resource.

CHAPTER 2

Source Code and Issue Tracker

For development GitHub is used, so source code and issue tracker is found there.

CHAPTER 3

Indices and tables

- genindex
- search