FREE eBook

LEARNING tastypie

#tastypie

Table of Contents

About

Chapter 1: Getting started with tastypie

Remarks

This section provides an overview of what tastypie is, and why a developer might want to use it.

It should also mention any large subjects within tastypie, and link out to the related topics. Since the Documentation for tastypie is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting tastypie set up or installed.

Read Getting started with tastypie online: https://riptutorial.com/tastypie/topic/9618/getting-started-with-tastypie

Chapter 2: Coding with Tastypie

Introduction

Tastypie is a webservice API framework for Django. It provides a convenient, yet powerful and highly customizable abstraction for creating REST-style interfaces. Tastypie makes exposing your models easy, but gives you full control over what you expose, letting you abstract away the database as much as needed. Tastypie also makes it easy to integrate with non-ORM data sources. Hence Tastypie can be used with ORM and Non-ORM databases.

Examples

Quick Start

1. Add tastypie to INSTALLED_APPS.
2. Create an api directory in your app with a bare **init**.py.
3. Create an <my_app>/api/resources.py file and place the following in it:

```
from tastypie.resources import ModelResource
from my_app.models import MyModel


class MyModelResource(ModelResource):
    class Meta:
        queryset = MyModel.objects.all()
        allowed_methods = ['get']
```

In your root URLconf, add the following code (around where the admin code might be):

```
from django.conf.urls import url, include
from tastypie.api import Api
from my_app.api.resources import MyModelResource

v1_api = Api(api_name='v1')
v1_api.register(MyModelResource())

urlpatterns = [
  # ...more URLconf bits here...
  # Then add:
  url(r'^api/', include(v1_api.urls)),
]
```

Tastypie Installation

Tastypie can be installed with python package management, ie, `pip` or we can directly checkout the code from Github

1. pip install django-tastypie
2. Checkout from Github

Why Tastypie?

There are other API frameworks out there for Django. You need to assess the options available and decide for yourself. That said, here are some common reasons for tastypie.

- You need an API that is RESTful and uses HTTP well.
- You want to support deep relations.
- You DON'T want to have to write your own serializer to make the output right.
- You want an API framework that has little magic, very flexible and maps well to the problem domain.
- You want/need XML serialization that is treated equally to JSON (and YAML is there too).

To develop API with Django, we have many options. The main options are **Tastypie** and **Django Rest Framework**(DRF).

What makes a decent API Framework?

These features:

1. pagination
2. posting of data with validation
3. Publishing of metadata along with querysets
4. API discovery
5. proper HTTP response handling
6. caching
7. serialization
8. throttling
9. permissions
10. authentication

Proper API frameworks also need:

11. Really good test coverage of their code

12. Decent performance

13. Documentation

14. An active community to advance and support the framework

If you take these factors, at this time there are only two API frameworks worth using (depends in user views), django-tastypie and django-rest-framework.

Which one is better? django-tastypie or django-rest-framework?

I say they are equal. You simply can't go wrong with either one. The authors and communities behind both of them are active, the code is solid and tested. And here are my specific thoughts about both of them:

Tastypie:

Advantages:

- Easy to get started with and provide basic functionalities OOB (out of the box)
- Most of the time you won't be dealing with Advanced Django concepts like CBVs, Forms etc
- More readable code and less of magic!
- If your models are NON-ORM, go for it.

Disadvantages:

- Doesn't strictly follow idiomatic Django (mind well python and django's philosophies are quite different)
- Probably bit tough to customize APIs once you go big
- No O-Auth

  DRF:

Advantages:

- Follow idiomatic django. (If you know django inside out, and very comfortable with CBV, Forms etc without any doubt go for it)

- Provides out of the box REST functionality using ModelViewSets. At the same time, provides greater control for customization using CustomSerializer, APIView, GenericViews etc.

- Better authentication.

- Easier to write custom permission classes.

- Work very well and importantly very easy to make it work with 3rd party libraries and OAuth.

- DJANGO-REST-AUTH is worth mentioning LIBRARY for Auth/SocialAuthentication/Registration.

Disadvantages:

- If you don't know Django very well, don't go for this.
- Magic! Some time very hard to understand magic.
- Because its been written on top of django's CBV which are in turn quite complex in nature.
- Has steep learning curve.

Getting Started with Tastypie

Tastypie is a reusable app (that is, it relies only on its own code and focuses on providing just a REST-style API) and is suitable for providing an API to any application without having to modify the sources of that app.

Not everyone's needs are the same, so Tastypie goes out of its way to provide plenty of hooks for overriding or extending how it works.

For example purposes, we'll be adding an API to a simple blog application.

*The only mandatory configuration is adding '**tastypie**' to your **INSTALLED_APPS**. This isn't strictly necessary, as Tastypie has only two non-required models, but may ease usage.*

Here is myapp/models.py:

```
from tastypie.utils.timezone import now
from django.contrib.auth.models import User
from django.db import models
from django.utils.text import slugify


class Entry(models.Model):
    user = models.ForeignKey(User)
    pub_date = models.DateTimeField(default=now)
    title = models.CharField(max_length=200)
    slug = models.SlugField(null=True, blank=True)
    body = models.TextField()
```

Creating Resources

REST-style architecture talks about resources, so unsurprisingly integrating with Tastypie involves creating *Resource classes*. For our simple application, we'll create a file for these in **myapp/api.py** , though they can live anywhere in your application:

myapp/api.py

```
from tastypie.resources import ModelResource
from myapp.models import Entry


class EntryResource(ModelResource):
    class Meta:
        queryset = Entry.objects.all()
        resource_name = 'entry'
```

In REST everything is a resource, means objects are described as resources. So for creating a Tastypie REST API involves in creating a resource class. To create a resource, we need to subclass ModelResource class. This EntryResource class will check all the non-relational fields on the Entry model and create its own Api fields. *This works just like the ModelForm of Django Forms.*

Hooking Up The Resource: Afrer creating the Resource, we need to tell the Django, that a resource is created by hooking our EntryResource to the URL.

urls.py

```
from django.conf.urls import url, include
from myapp.api import EntryResource

entry_resource = EntryResource()
```

```
urlpatterns = [
    url(r'^blog/', include('myapp.urls')),
    url(r'^api/', include(entry_resource.urls)),
]
```

Now if we check on **localhost:8000/api/entry/** we will get the API response.

*(The **resource_name** within the Meta class is optional. If not provided, it is automatically generated off the classname, removing any instances of Resource and lowercasing the string. So **EntryResource would become just entry**. That's why in the URL we specified **../entry/**)*

Tastypie Authorization

Up to now we tried get request. *(If you need to try other HTTP resquest, use some API testing tools like curl or postman)*

If you try sending a POST/PUT/DELETE to the resource, you find yourself getting “401 Unauthorized” errors. For safety, Tastypie ships with the authorization class set to ReadOnlyAuthorization. This makes it safe to expose on the web, but prevents us from doing POST/PUT/DELETE. Let’s enable those.

myapp/api.py

```
from tastypie.authorization import Authorization
from tastypie.resources import ModelResource
from myapp.models import Entry


class EntryResource(ModelResource):
    class Meta:
        queryset = Entry.objects.all()
        resource_name = 'entry'
        authorization = Authorization()
```

Read Coding with Tastypie online: https://riptutorial.com/tastypie/topic/10640/coding-with-tastypie

Credits

S. No	Chapters	Contributors
1	Getting started with tastypie	Community
2	Coding with Tastypie	AKHIL MATHEW