

25 ноября 2012 в 05:34

Web API с помощью Django REST framework

[Веб-разработка*](#), [Django*](#), [API*](#)

Веб-сервис (англ. web service) — идентифицируемая веб-адресом программная система со стандартизированными интерфейсами. Веб-службы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (XML, JSON и т. д.). Веб-служба является единицей модульности при использовании сервис-ориентированной архитектуры приложения.

Одним из подходов создания веб сервиса является rest.

Rest (сокр. англ. Representational State Transfer, «передача состояния представления») — стиль построения архитектуры распределенного приложения. Данные в REST должны передаваться в виде небольшого количества стандартных форматов (например HTML, XML, JSON). Сетевой протокол (как и HTTP) должен поддерживать кэширование, не должен зависеть от сетевого слоя, не должен сохранять информацию о состоянии между парами «запрос-ответ». Утверждается, что такой подход обеспечивает масштабируемость системы и позволяет ей эволюционировать с новыми требованиями.

Django REST framework — удобный инструмент для работы с rest основанный на идеологии фреймворка Django.

Требования к окружению:

Python (2.6, 2.7)

Django (1.3, 1.4, 1.5)

По желанию:

[Markdown](#)

[PyYAML](#)

[django-filter](#)

Установка

Установить можно привычной для нас командой pip:

```
pip install.djangorestframework
```

И можно поставить дополнительные пакеты:

```
pip install markdown
pip install pyyaml
pip install django-filter
```

Или же сделать клон проекта с Github:

```
git clone git@github.com:tomchristie/django-rest-framework.git
cd django-rest-framework
pip install -r requirements.txt
pip install -r optionals.txt
```

Не забываем прописать приложение в INSTALLED_APPS:

```
INSTALLED_APPS = (
    ...
    'rest_framework',
)
```

А также добавить запись в urls.py:

```
urlpatterns = patterns('',
    ...
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework'))
)
```

Url можно ставить любой на Ваш вкус, главное подключить файл с урлами rest framework (rest_framework.urls).

Пример использования

Создадим API для работы с пользователями и их группами.

Для начала нам нужно определить некоторые Serializers, которые мы будем использовать

```
from django.contrib.auth.models import User, Group, Permission
from rest_framework import serializers

class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = ('url', 'username', 'email', 'groups')

class GroupSerializer(serializers.HyperlinkedModelSerializer):
    permissions = serializers.ManySlugRelatedField(
        slug_field='codename',
        queryset=Permission.objects.all()
    )

    class Meta:
        model = Group
        fields = ('url', 'name', 'permissions')
```

Пропишем views.py

```
from django.contrib.auth.models import User, Group
from rest_framework import generics
from rest_framework.decorators import api_view
from rest_framework.reverse import reverse
from rest_framework.response import Response
from quickstart.serializers import UserSerializer, GroupSerializer

@api_view(['GET'])
def api_root(request, format=None):
    """
    The entry endpoint of our API.
    """
    return Response({
        'users': reverse('user-list', request=request),
        'groups': reverse('group-list', request=request),
    })

class UserList(generics.ListCreateAPIView):
    """
    API endpoint that represents a list of users.
    """
    model = User
    serializer_class = UserSerializer

class UserDetail(generics.RetrieveUpdateDestroyAPIView):
    """
    API endpoint that represents a single user.
    """
    model = User
    serializer_class = UserSerializer

class GroupList(generics.ListCreateAPIView):
    """
    API endpoint that represents a list of groups.
    """
    model = Group
    serializer_class = GroupSerializer

class GroupDetail(generics.RetrieveUpdateDestroyAPIView):
    """
    API endpoint that represents a single group.
    """
    model = Group
    serializer_class = GroupSerializer
```

Мы создали функцию `api_root`, которая будет отправной точкой для нашего API. И четыре класса, для связи с моделями и указали какие `serializers` нужно при этом использовать.

Добавим ссылки в `urls.py`

```
from django.conf.urls import patterns, url, include
from rest_framework.urlpatterns import format_suffix_patterns
from quickstart.views import UserList, UserDetails, GroupList, GroupDetail

urlpatterns = patterns('quickstart.views',
    url(r'^$', 'api_root'),
    url(r'^users/$', UserList.as_view(), name='user-list'),
    url(r'^users/(?P<pk>\d+)/$', UserDetails.as_view(), name='user-detail'),
    url(r'^groups/$', GroupList.as_view(), name='group-list'),
    url(r'^groups/(?P<pk>\d+)/$', GroupDetail.as_view(), name='group-detail'),
)

# Format suffixes
urlpatterns = format_suffix_patterns(urlpatterns, allowed=['json', 'api'])

# Default login/logout views
urlpatterns += patterns('',
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework'))
)
```

Важный момент использование `user-detail` и `group-detail`. Для корректной связи с `views.py` нужно использовать именование вида `{modelname}-detail`.

В `format_suffix_patterns` мы указали суффикс для наших `urls`.

Settings

```
INSTALLED_APPS = (
    ...
    'rest_framework',
)

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': ('rest_framework.permissions.IsAdminUser',),
    'PAGINATE_BY': 10
}
```

Результат

Используя `curl` в консоли испытаем что же получилось:

```
bash: curl -H 'Accept: application/json; indent=4' -u admin:password http://127.0.0.1:8000/users/
{
  "count": 2,
  "next": null,
  "previous": null,
  "results": [
    {
      "email": "admin@example.com",
      "groups": [],
      "url": "http://127.0.0.1:8000/users/1/",
      "username": "admin"
    },
    {
      "email": "tom@example.com",
      "groups": [
      ],
      "url": "http://127.0.0.1:8000/users/2/",
      "username": "tom"
    }
  ]
}
```

В браузере можно увидеть что то подобное:

Django REST framework v2.0.0

admin

User List

OPTIONSGET

API endpoint that represents a list of users.

GET /users/

HTTP 200 OK

Vary: Accept

Content-Type: text/html

Allow: GET, POST, HEAD, OPTIONS

```
{
  "count": 2,
  "next": null,
  "previous": null,
  "results": [
    {
      "email": "admin@example.com",
      "groups": [],
      "url": "http://127.0.0.1:8000/users/1/",
      "username": "admin"
    },
    {
      "email": "tom@example.com",
      "groups": [],
      "url": "http://127.0.0.1:8000/users/2/",
      "username": "tom"
    }
  ]
}
```

Username

Email

Groups

POST

Ссылки:

[Сайт проекта](#)
[Github](#)

P.S. Для работы с rest в django есть еще одно не плохое приложение именуемое как [django-piston](#). При желании можно и о нем написать, хотя в использовании оно довольно не сложное.

[api](#), [django](#), [Django REST framework](#) изменить

+18

12564

168

int22h^{23,4}

комментарии (8) ☐ отслеживать новые: ☐ в почте ☒ в треке

 **MechanisM**


25 ноября 2012 в 05:44

#

+5

Для работы с rest в django есть еще одно не плохое приложение именуемое как [django-piston](#).

Одним из самых популярных для этих целей является [django-tastypie](#).

 **int22h**

25 ноября 2012 в 05:46

#

↑

0

Благодарю, обязательно рассмотрю.

 **xmdy**

25 ноября 2012 в 13:42

#

↑

0

да, тоже смотрел в сторону tastypie, плюс легко интегрировать с backbone)

 **MechanisM**

25 ноября 2012 в 16:25 (комментарий был изменён)

#

↑

0

А есть полно всяких интересных решений. На самом деле [есть из чего выбирать](#).

Я просто был удивлен что тут написали только о django-piston и django-rest-framework, а про самый популярный, и наиболее часто используемый django-tastypie ничего не сказали)



Sellec, 25 ноября 2012 в 09:49 #

0

Недавно [TVT](#) упомянули REST, заинтересовался для своего проекта — а тут и эта статья подкатила)



liaren, 25 ноября 2012 в 11:23 (комментарий был изменён) #

0

Если это всё относится только к Python + Django, то и назвали бы пост соответственно... Спасибо.



xSkyFoXx, 25 ноября 2012 в 20:15 (комментарий был изменён) #

0

Самая большая проблема состоит в том, что подобная схема не позволяет хранить модели заведомо неизвестной длины. Это, правда, ограничение реляционной базы данных, а не выбранной связки. Такие API я отношу к «внутренним», т.к. таким, которые можно использовать для своих же приложений.

Питон, благодаря универсальным функциям (`def foo(*args, **kwargs)`) позволяет обрабатывать сколь угодно сложные вложенные последовательности. Другое дело, что они не переключаются на структуру реляционных баз. Зато на Key-Value NoSQL-хранилища — за милую душу. Например на MongoDB. Для Django даже есть backend. Т.е. можно будет написать некоторую универсальную модель, которая будет принимать на вход последовательности произвольной вложенности. И спокойно класть их в базу. Кроме того, замечательно когда можно описывать модели данных вне моделей django. Это очень упрощает доработку и изменение схемы данных для API. Особенно если это делать в замечательном формате, который вы и используете — `yaml`.



fledgling, 26 ноября 2012 в 01:43 (комментарий был изменён) #



+2

MongoDB — это не kv-хранилище, а документо-ориентированное.

NoSQL — это не более, чем маркетинговое название всех хранилищ (и баз данных в целом), в которых, если высказаться грубо, нет «сложного» sql (на самом деле, not only sql, хотя я практически присутствовал во время, когда posql всплыл как нечто, что не более, чем очередной интернето-твиторо-мем): иерархические, документо-, объектно-ориентированные (но тут возможен страшный OQL), те же kv.

Опять же, никто не запрещает использовать те же mysql/postgresql/any sql как k/v хранилища. Индексируйте данные модным elasticsearch'ем и получайте их по ключу «select * from table where id = ?» (или и вовсе пройдите за пачкой данных напрямую в хранилище).

Хипсторство не ра-бо-та-ет, если вы не понимаете, для чего оно вам нужно.

А я, в свою очередь, вижу общую путаницу на тему БД, СУБД, SQL, NoSQL, KV, документо-ориентированные и хипста-ориентированные хранилища, базы данных и систем управления базами данных. Что, в целом, меня немного расстраивает.

Проблемы, с которым сталкивается человечество при разработке архитектуры данных начинаются где-то около твитора или фейсбука — это многочисленные счётчики (замена update на append-only + моментальная агрегация), различные ссылки объектов туда-сюда, друг на друга, етц, в следствие возникают проблемы с расширяемостью, придумывают map-reduce и аналоги. И решают эти задачи отнюдь не на дедике в hetzner.net, а на, как минимум, десятке таких.

Глянуть на тот же достаточно взрывной Pinterest:

— Март 2010: MySQL

— Январь 2011: MongoDB

— Весна-лето 2012: MySQL — по сей день

На этом внезапно закончу, а то потянуло на графоманство. Думаю, Вы поняли о чём я :-)

[Fakebook. Концепт
здравомыслия](#)

[Petrovich просклоняет русские
имена](#)

[Вода горит! А также ЭГЭ и
волны-убийцы](#)

[Все мозги в одном месте](#)

[Заказы для фрилансеров](#)

[Вакансии для айтишников](#)

[Уютная и дружелюбная](#)