DjangoRestless Documentation

Release 0.0.9

Senko Rasic

July 25, 2016

Contents

1 Installation **3**

2 Usage **5**
- 2.1 Views . . . 5
- 2.2 Model serialization . . . 6
- 2.3 Data deserialization and validation . . . 7
- 2.4 Generic views for CRUD operations on models . . . 7
- 2.5 RPC-style API for model views . . . 8

3 API Reference **9**
- 3.1 restless.views . . . 9
- 3.2 restless.modelviews . . . 9
- 3.3 restless.models . . . 11
- 3.4 restless.auth . . . 12
- 3.5 restless.http . . . 12

4 How to contribute **15**
- 4.1 Repository . . . 15
- 4.2 Tests and docs . . . 15

5 Indices and tables **17**

Python Module Index **19**

Django Restless is a lightweight set of tools for implementing JSON-based RESTful APIs in Django. It helps you to write APIs that loosely follow the RESTful paradigm, without forcing you to do so, and without imposing a full-blown REST framework.

Restless provides only JSON support. If you need to support XML or other formats, you probably want to take a look at some of the other frameworks out there (we recommend Django REST framework).

Here is a simple view implementing an API endpoint greeting the caller:

```
from restless.views import Endpoint

class HelloWorld(Endpoint):
    def get(self, request):
        name = request.params.get('name', 'World')
        return {'message': 'Hello, %s!' % name}
```

One of the main ideas behind Restless is that it's lightweight and reuses as much of functionality in Django as possible. For example, input parsing and validation is done using standard Django forms. This means you don't have to learn a whole new API to use Restless.

Besides giving you a set of tools to implement your own APIs, Restless comes with a few endpoints modelled after Django's generic class-based views for common use-cases.

For example, here's how to implement list and detail endpoints for *MyModel* class allowing the users to list, create, get details of, update and delete the models via API:

```
from restless.modelviews import ListEndpoint, DetailEndpoint
from myapp.models import MyModel

class MyList(ListEndpoint):
    model = MyModel

class MyDetail(DetailEndpoint):
    model = MyModel
```

CHAPTER 1

Installation

Django Restless is available from cheeseshop, so you can install it via pip:

```
pip install DjangoRestless
```

For the latest and the greatest, you can also get it directly from git master:

```
pip install -e git+ssh://github.com/dobarkod/django-restless/tree/master
```

The supported Python versions are 2.6, 2.7 and 3.3.

CHAPTER 2

Usage

After installation, first add *restless* to *INSTALLED_APPS* (this is not strictly necessary, as restless is just a bunch of helper classes and functions, but is good form nevertheless):

```
INSTALLED_APPS += ('restless',)
```

2.1 Views

To implement API endpoints (resources), write class-based views subclassing from *restless.views.Endpoint*:

```
from restless.views import Endpoint

class HelloWorld(Endpoint):
    def get(self, request):
        name = request.params.get('name', 'World')
        return {'message': 'Hello, %s!' % name}
```

This view will return a HTTP 200 response with a JSON payload greeting the caller. To return status code other than 200, use one of the `restless.views.JSONResponse` and pass the response in. Note that error responses take a string (error description) and optional error details keyword arguments.

Register the views in URLconf as you'd do with any other class-based view:

```
url(r'^hello/$', HelloWorld.as_view())
```

To require authentication, subclass from appropriate mixin as well. For example, if you're using HTTP Basic authentication, have all your views subclass *restless.auth.BasicHttpAuthMixin* and use *restless.auth.login_required* for requiring the user be authenticated:

```
from restless.auth import BasicHttpAuthMixin, login_required

class SecretGreeting(Endpoint, BasicHttpAuthMixin):
    @login_required
    def get(self, request):
        return {'message': 'Hello, %s!' % request.user}
```

If you're using session-based username/password authentication, you can use the *restless.auth.UsernamePasswordAuthMixin* in the above example, or just use *restless.auth.AuthenticateEndpoint* which will do the same, and return the serialized User object back to the authenticated user:

```
url(r'^login/$', restless.auth.AuthenticateEndpoint.as_view())
```

2.2 Model serialization

Model serialization can be as simple or as complex as needed. In the simplest case, you just pass the object to *restless.models.serialize()*, and get back a dictionary with all the model fields (except related models) serialized:

```
from django.contrib.auth import get_user_model
from restless.models import serialize

User = get_user_model()

class GetUserProfileData(Endpoint):
    def get(self, request, user_id):
        profile = User.objects.get(pk=user_id).get_profile()
        return serialize(profile)
```

In some cases, you want to serialize only a subset of fields. Do this by passing a list of fields to serialize as the second argument:

```
class GetUserData(Endpoint):
    def get(self, request, user_id):
        fields = ('id', 'username', 'first_name', 'last_name', 'email')
        user = User.objects.get(pk=user_id)
        return serialize(user, fields)
```

Or you may only want to exclude a certain field:

```
class GetUserData(Endpoint):
    def get(self, request, user_id):
        user = User.objects.get(pk=user_id)
        return serialize(user, exclude=['password'])
```

Sometimes, you really need to complicate things. For example, for a book author, you want to retrieve all the books they've written, and for each book, all the user reviews, as well as the average rating for the author accross all their books:

```
class GetAuthorWithBooks(Endpoint):
    def get(self, request, author_id):
        author = Author.objects.get(pk=author_id)
        return serialize(author, include=[
            ('books', dict(  # for each book
                fields=[
                    'title',
                    'isbn',
                    ('reviews', dict()) # get a list of all reviews
                ]
            )),
            ('average_rating',
                lambda a: a.books.all().aggregate(
                    Avg('rating'))['avg_rating'])
        ])
```

Please see the *restless.models.serialize()* documentation for detailed description how this works.

Note: The *serialize* function changed in 0.0.4, and the *related* way of specifying sub-objects is now deprecated.

2.3 Data deserialization and validation

There is no deserialization support. Django already has awesome functionality in this regard called ModelForms, and that's the easiest way to go about parsing users' data and storing it into Django models.

Since Restless understands JSON payloads, it's easy to use Django forms to parse and validate client input, even in more complex cases where several models (or several forms) need to be parsed at once.

Let's say we have a Widget object that can be extended with Addon:

```
class Widget(models.Model):
    title = models.CharField(max_length=255)

class Addon(models.Model):
    parent = models.OneToOneField(Widget, related_name='addon')
    text = model.TextField()

class WidgetForm(forms.ModelForm):
    class Meta:
        model = Widget

class AddonForm(forms.ModelForm):
    class Meta:
        model = Addon
```

and the PUT request from user modifies the Widget object:

```
{ "title": "My widget!", "addon": { "text": "This is my addon" } }
```

A view handing the PUT request might look something like:

```
class ModifyWidget(Endpoint):
    def put(self, request, widget_id):
        widget = Widget.objects.get(pk=widget_id)
        widget_form = WidgetForm(request.data, instance=widget)
        addon_form = AddonForm(request.data.get('addon', {}),
            instance=widget.addon)
        if widget_form.is_valid() and addon_form.is_valid():
            widget_form.save()
            addon_form.save()
```

You can find more examples in the sample project used to test restless in the various files in the "testproject/testapp" folder of the source repository.

2.4 Generic views for CRUD operations on models

If you need a generic object CRUD operations, you can make use of the *restless.modelviews.ListEndpoint* and *restless.modelviews.DetailEndpoint* views. Here's an example of the list and detail views providing an easy way to list, create, get, update and delete a Book objects in a database:

```
# views.py
class BookList(ListEndpoint):
    model = Book

class BookDetail(DetailEndpoint):
    model = Book

# urls.py
urlpatterns += patterns('',
    url(r'^books/$', BookList.as_view(),
        name='book_list'),
    url(r'^books/(?P<pk>\d+)$', BookDetail.as_view(),
        name='book_detail'))
```

The *pk* parameter here was automatically used by the detail view. The parameter name can be customized if needed.

There are a number of ways to customize the generic views, explained in the API reference in more detail.

2.5 RPC-style API for model views

Sometimes a RPC-style API on models is needed (for example, to set a flag on the model). The *restless.modelviews.ActionEndpoint* provides an easy way to do it. ActionEndpoint is a subclass of *restless.modelviews.DetailEndpoint* allowing only *POST* HTTP request by default, which invoke the `restless.modelviews.DetailEndpoint.action()` method.

Here's an example of a Book endpoint on which a POST marks the book as borrowed by the current user:

```
class BorrowBook(ActionEndpoint):
    model = Book

    @login_required
    def action(self, request, obj, *args, **kwargs):
        obj.borrowed_by = request.user
        obj.save()
        return serialize(obj)
```

CHAPTER 3

API Reference

3.1 restless.views

Base classes for class-based views implementing the API endpoints.

class restless.views.**Endpoint**(***kwargs*)

Class-based Django view that should be extended to provide an API endpoint (resource). To provide GET, POST, PUT, HEAD or DELETE methods, implement the corresponding get(), post(), put(), head() or delete() method, respectively.

If you also implement authenticate(request) method, it will be called before the main method to provide authentication, if needed. Auth mixins use this to provide authentication.

The usual Django "request" object passed to methods is extended with a few more attributes:

- request.content_type - the content type of the request
- request.params - a dictionary with GET parameters
- **request.data - a dictionary with POST/PUT parameters, as parsed from** either form submission or submitted application/json data payload
- request.raw_data - string containing raw request body

The view method should return either a HTTPResponse (for example, a redirect), or something else (usually a dictionary or a list). If something other than HTTPResponse is returned, it is first serialized into *restless.http.JSONResponse* with a status code 200 (OK), then returned.

The authenticate method should return either a HttpResponse, which will shortcut the rest of the request handling (the view method will not be called), or None (the request will be processed normally).

Both methods can raise a *restless.http.HttpError* exception instead of returning a HttpResponse, to shortcut the request handling and immediately return the error to the client.

3.2 restless.modelviews

Generic class-based views providing CRUD API for the models.

class restless.modelviews.**ListEndpoint**(***kwargs*)

List *restless.views.Endpoint* supporting getting a list of objects and creating a new one. The endpoint exports two view methods by default: get (for getting the list of objects) and post (for creating a new object).

The only required configuration for the endpoint is the *model* class attribute, which should be set to the model you want to have a list (and/or create) endpoints for.

You can also provide a *form* class attribute, which should be the model form that's used for creating the model. If not provided, the default model class for the model will be created automatically.

You can restrict the HTTP methods available by specifying the *methods* class variable.

get(*request*, **args*, ***kwargs*)
Return a serialized list of objects in this endpoint.

get_query_set(*request*, **args*, ***kwargs*)
Return a QuerySet that this endpoint represents.

If *model* class attribute is set, this method returns the *all()* queryset for the model. You can override the method to provide custom behaviour. The *args* and *kwargs* parameters are passed in directly from the URL pattern match.

If the method raises a *restless.http.HttpError* exception, the rest of the request processing is terminated and the error is immediately returned to the client.

post(*request*, **args*, ***kwargs*)
Create a new object.

serialize(*objs*)
Serialize the objects in the response.

By default, the method uses the *restless.models.serialize()* function to serialize the objects with default behaviour. Override the method to customize the serialization.

class restless.modelviews.**DetailEndpoint**(***kwargs*)
Detail *restless.views.Endpoint* supports getting a single object from the database (HTTP GET), updating it (HTTP PUT) and deleting it (HTTP DELETE).

The only required configuration for the endpoint is the *model* class attribute, which should be set to the model you want to have the detail endpoints for.

You can also provide a *form* class attribute, which should be the model form that's used for updating the model. If not provided, the default model class for the model will be created automatically.

You can restrict the HTTP methods available by specifying the *methods* class variable.

delete(*request*, **args*, ***kwargs*)
Delete the object represented by this endpoint.

get(*request*, **args*, ***kwargs*)
Return the serialized object represented by this endpoint.

get_instance(*request*, **args*, ***kwargs*)
Return a model instance represented by this endpoint.

If *model* is set and the primary key keyword argument is present, the method attempts to get the model with the primary key equal to the url argument.

By default, the primary key keyword argument name is *pk*. This can be overridden by setting the *lookup_field* class attribute.

You can override the method to provide custom behaviour. The *args* and *kwargs* parameters are passed in directly from the URL pattern match.

If the method raises a *restless.http.HttpError* exception, the rest of the request processing is terminated and the error is immediately returned to the client.

put(*request*, **args*, ***kwargs*)
Update the object represented by this endpoint.

serialize(*obj*)
Serialize the object in the response.

By default, the method uses the *restless.models.serialize()* function to serialize the object with default behaviour. Override the method to customize the serialization.

class restless.modelviews.**ActionEndpoint**(***kwargs*)
A variant of *DetailEndpoint* for supporting a RPC-style action on a resource. All the documentation for DetailEndpoint applies, but only the *POST* HTTP method is allowed by default, and it invokes the `ActionEndpoint.action()` method to do the actual work.

If you want to support any of the other HTTP methods with their default behaviour as in DetailEndpoint, just modify the *methods* list to include the methods you need.

3.3 restless.models

Model serialization helper.

restless.models.**serialize**(*src*, *fields=None*, *related=None*, *include=None*, *exclude=None*, *fixup=None*)
Serialize Model or a QuerySet instance to Python primitives.

By default, all the model fields (and only the model fields) are serialized. If the field is a Python primitive, it is serialized as such, otherwise it is converted to string in utf-8 encoding.

If *fields* is specified, it is a list of attribute descriptions to be serialized, replacing the default (all model fields). If *include* is specified, it is a list of attribute descriptions to add to the default list. If *exclude* is specified, it is a list of attribute descriptions to remove from the default list.

Each attribute description can be either:

- a string - includes a correspondingly named attribute of the object being serialized (eg. *name*, or *created_at*); this can be a model field, a property, class variable or anything else that's an attribute on the instance
- a tuple, where the first element is a string key and the second is a function taking one argument - function will be run with the object being serialized as the argument, and the function result will be included in the result, with the key being the first tuple element
- a tuple, where the first element is a related model attribute name and the second is a dictionary - related model instance(s) will be serialized recursively and added as sub-object(s) to the object being serialized; the dictionary may specify *fields*, *include*, *exclude* and *fixup* options for the related models following the same semantics as for the object being serialized.

The *fixup* argument, if defined, is a function taking two arguments, the object being serialized, and the serialization result dict, and returning the modified serialization result. It's useful in cases where it's neccessary to modify the result of the automatic serialization, but its use is discouraged if the same result can be obtained through the attribute descriptions.

The *related* argument (a different way of specifying related objects to be serialized) is deprecated and included only for backwards compatibility.

Example:

```
    serialize(obj, fields=[
        'name',    # obj.name
        'dob',     # obj.dob
        ('age', lambda obj: date.today() - obj.dob),
        ('jobs', dict(   # for job in obj.jobs.all()
            fields=[
```

```
                'title',  # job.title
                'from',   # job.from
                'to',     # job.to,
                ('duration', lambda job: job.to - job.from),
            ]
        ))
    ])
```

Returns: a dict (if a single model instance was serialized) or a list od dicts (if a QuerySet was serialized) with the serialized data. The data returned is suitable for JSON serialization using Django's JSON serializator.

`restless.models.`**`flatten`**(*attname*)
Fixup helper for serialize.

Given an attribute name, returns a fixup function suitable for serialize() that will pull all items from the sub-dict and into the main dict. If any of the keys from the sub-dict already exist in the main dict, they'll be overwritten.

3.4 restless.auth

Authentication helpers.

class `restless.auth.`**`UsernamePasswordAuthMixin`**
restless.views.Endpoint mixin providing user authentication based on username and password (as specified in "username" and "password" request GET params).

class `restless.auth.`**`BasicHttpAuthMixin`**
restless.views.Endpoint mixin providing user authentication based on HTTP Basic authentication.

class `restless.auth.`**`AuthenticateEndpoint`**(***kwargs*)
Session-based authentication API endpoint. Provides a GET method for authenticating the user based on passed-in "username" and "password" request params. On successful authentication, the method returns authenticated user details.

Uses *UsernamePasswordAuthMixin* to actually implement the Authentication API endpoint.

On success, the user will get a response with their serialized User object, containing id, username, first_name, last_name and email fields.

`restless.auth.`**`login_required`**(*fn*)
Decorator for *restless.views.Endpoint* methods to require authenticated, active user. If the user isn't authenticated, HTTP 403 is returned immediately (HTTP 401 if Basic HTTP authentication is used).

3.5 restless.http

HTTP responses with JSON payload.

class `restless.http.`**`JSONResponse`**(*data*, ***kwargs*)
HTTP response with JSON body ("application/json" content type)

class `restless.http.`**`JSONErrorResponse`**(*reason*, ***additional_data*)
HTTP Error response with JSON body ("application/json" content type)

exception `restless.http.`**`HttpError`**(*code*, *reason*, ***additional_data*)
Exception that results in returning a JSONErrorResponse to the user.

class `restless.http.`**`Http200`**(*data*, ***kwargs*)
HTTP 200 OK

class restless.http.**Http201**(*data*, ***kwargs*)
HTTP 201 CREATED

class restless.http.**Http400**(*reason*, ***additional_data*)
HTTP 400 Bad Request

class restless.http.**Http401**(*typ='basic'*, *realm='api'*)
HTTP 401 UNAUTHENTICATED

class restless.http.**Http403**(*reason*, ***additional_data*)
HTTP 403 FORBIDDEN

CHAPTER 4

How to contribute

You've found (and hopefully fixed) a bug, or have a great idea you think should be added to Django Restless? Patches welcome! :-)

Bugs (and feature requests) are reported via the GitHub Issue tracker: https://github.com/dobarkod/django-restless/issues/

If you have a bug fix or a patch for a feature you'd like to include, here's how to submit the patch:

- Fork the https://github.com/dobarkod/django-restless.git repository
- Make the changes in a branch in your fork
- Make a pull request from the branch in your fork to dobarkod/django-restless master

If you're suggesting adding a feature, please file a feature request first before implementing it, so we can discuss your proposed solution.

When contributing code, please adhere to the Python coding style guide (PEP8). Both bug fixes and new feature implementations should come with corresponding unit/functional tests. For bug fixes, the test should exhibit the bug if the fix is not applied.

You can see the list of the contributors in the AUTHORS.md file in the Django Restless source code.

4.1 Repository

Restless is hosted on GitHub, using git version control. When checking for bugs, always try the git master first:

```
git clone https://github.com/dobarkod/django-restless.git
```

4.2 Tests and docs

To run the tests:

```
make test
```

To run the tests and get coverage report:

```
make coverage
```

To build Sphinx docs:

```
make docs
```

To build everything, run the tests with coverage support, and build the docs:

```
make
```

To clean the build directory:

```
make clean
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

r

restless.auth, 12
restless.http, 12
restless.models, 11
restless.modelviews, 9
restless.views, 9

A

ActionEndpoint (class in restless.modelviews), 11
AuthenticateEndpoint (class in restless.auth), 12

B

BasicHttpAuthMixin (class in restless.auth), 12

D

delete() (restless.modelviews.DetailEndpoint method), 10
DetailEndpoint (class in restless.modelviews), 10

E

Endpoint (class in restless.views), 9

F

flatten() (in module restless.models), 12

G

get() (restless.modelviews.DetailEndpoint method), 10
get() (restless.modelviews.ListEndpoint method), 10
get_instance() (restless.modelviews.DetailEndpoint method), 10
get_query_set() (restless.modelviews.ListEndpoint method), 10

H

Http200 (class in restless.http), 12
Http201 (class in restless.http), 12
Http400 (class in restless.http), 13
Http401 (class in restless.http), 13
Http403 (class in restless.http), 13
HttpError, 12

J

JSONErrorResponse (class in restless.http), 12
JSONResponse (class in restless.http), 12

L

ListEndpoint (class in restless.modelviews), 9
login_required() (in module restless.auth), 12

P

post() (restless.modelviews.ListEndpoint method), 10
put() (restless.modelviews.DetailEndpoint method), 10

R

restless.auth (module), 12
restless.http (module), 12
restless.models (module), 11
restless.modelviews (module), 9
restless.views (module), 9

S

serialize() (in module restless.models), 11
serialize() (restless.modelviews.DetailEndpoint method), 10
serialize() (restless.modelviews.ListEndpoint method), 10

U

UsernamePasswordAuthMixin (class in restless.auth), 12