ADRest

Release 3.3.2

November 19, 2014

Contents

ADREST is an API framework for Django.

It supports REST and RPC paradigms.

.

Contents

CHAPTER 1

Requirements

- Python 2.7
- Django (1.5, 1.6, 1.7)

CHAPTER 2

Installation

ADRest should be installed using pip:

```
pip install adrest
```

CHAPTER 3

Quick start

```
from adrest import Api, ResourceView

api = Api('v1')

@api.register
class BookResource(ResourceView):
    class Meta:
        allowed_methods = 'get', 'post'
        model = 'app.book'

urlpatterns = api.urls
```

CHAPTER 4

User Guide

.

4.1 Configuration

You should add `adrest` to your `INSTALLED_APPS` in Django settings.

There are default **ADRest** settings are writen bellow.

`adrest.settings.`**ADREST_ACCESSKEY = False**
Create *adrest.models.AccessKey* models for authorisation by keys

`adrest.settings.`**ADREST_ACCESS_LOG = False**
Enable ADRest API logs. Information about requests and responses will be saved in database.

`adrest.settings.`**ADREST_ALLOW_OPTIONS = False**
We do not restrict access for OPTIONS request.

`adrest.settings.`**ADREST_AUTO_CREATE_ACCESSKEY = False**
Create AccessKey for Users automaticly

`adrest.settings.`**ADREST_DEBUG = False**
Dont parse a exceptions. Show standart Django 500 page.

`adrest.settings.`**ADREST_LIMIT_PER_PAGE = 50**
Set default number resources per page for pagination ADREST_LIMIT_PER_PAGE = 0 – Disable pagination

`adrest.settings.`**ADREST_MAIL_ERRORS = (500,)**
List of errors for ADRest's errors mails. Set ADREST_MAIL_ERRORS = None for disable this functionality

`adrest.settings.`**ADREST_MAP_TEMPLATE = 'api/map.html'**
Template path for ADRest map

`adrest.settings.`**ADREST_THROTTLE_AT = 120**
Set maximum requests per timeframe

`adrest.settings.`**ADREST_THROTTLE_TIMEFRAME = 60**
Set timeframe length

4.2 Creating Resources

TODO

4.3 API

You can use `adrest.Api` for bind multiple `adrest.ResourceView` together with version prefix.

4.3.1 Create API

Default use:

```
from adrest.api import Api
from myresources import Resource1, Resource2

api = Api('0.1')
```

4.3.2 Register a resource

After creation you can register some resources with that Api.

```
api.register(Resource1)
api.register(Resource2)
```

You can use *api.register* method as decorator:

```
@api.register
class SomeResource():
    ...
```

4.3.3 Enable API Map

You can enable API Map for quick reference on created resources. Use *api_map* param.

```
api = Api('1.0b', api_map=True)
```

By default access to map is anonimous. If you want use a custom authenticator register map resource by manualy.

```
from adrest.resources.map import MapResource

api = Api('1.0')
api.register(MapResource, authenticators=UserLoggedInAuthenticator)
```

4.3.4 Auto JSONRPC from REST

If you are created some REST api with adrest, you already have JSON RPC too. Use *api_rpc* param.

```
api = Api('1.0', api_rpc=True)
```

4.4 ADRest Mixins

ADRest based on mixin classes. You can use `adrest.views.ResourceView` as base for your REST controlers. Or you can use a ADRest's mixins separately.

Contents

4.4.1 Common Options

class adrest.utils.meta.**Meta**

Base options for all ADRest mixins.

With Meta options you can setup your resources.

model = None

Setup Django ORM model. Value could be a model class or string path like 'app.model'.

4.4.2 EmitterMixin

class adrest.mixin.emitter.**EmitterMixin**

Serialize response.

class Meta

Emitter options. Setup parameters for resource's serialization.

```
class SomeResource(EmitterMixin, View):

    class Meta:
        emitters = JSONEmitter
```

Meta.**emit_format = 'django'**

Serialization format. Set 'django' for django like view:

```
{
    'pk': ...,
    'model': ...,
    'fields': {
        'name': ...,
        ...
    }
}
```

Or set 'simple' for simpliest serialization:

```
{
    'id': ...,
    'name': ...,
}
```

Meta.**emit_models** = None
Dictionary with emitter's options for relations
•emit_models['fields'] – Set serialized fields by manual
•emit_models['exclude'] – Exclude some fields
•emit_models['include'] – Include some fields
•emit_models['related'] – Options for relations.
Example:

```
class SomeResource(EmitterMixin, View):
    class Meta:
        model = Role
        emit_models = dict(
             include = 'group_count',
             exclude = ['password', 'service'],
             related = dict(
                 user = dict(
                     fields = 'username'
                 )
             )

         )
```

You can use a shortcuts for *emit_models* option, as is *emit_fields* or *emit_include*. That same as bellow:

```
class SomeResource(EmitterMixin, View):
    class Meta:
        model = Role
        emit_include = 'group_count'
        emit_exclude = 'password', 'service'
        emit_related = dict(
            user = dict(
                    fields = 'username'
            )
        )
```

Meta.**emit_options** = None
Options for low-level serialization Example for JSON serialization

```
class SomeResource(EmitterMixin, View):
    class Meta:
        emit_options = dict(indent=2, sort_keys=True)
```

Meta.**emit_template** = None
Define template for template-based emitters by manualy Otherwise template name will be generated from resource name (or resource.Meta.model)

Meta.**emitters**
adrest.utils.Emitter (or collection of them) Defined available emitters for resource.

```
class SomeResource(EmitterMixin, View):
    class Meta:
        emitters = JSONEmitter, XMLEmitter
```

alias of JSONEmitter

Example:

```
class SomeResource():
    class Meta:
        emit_fields = ['pk', 'user', 'customfield']
        emit_related = {
            'user': {
                fields: ['username']
            }
        }

    def to_simple__customfield(self, user):
        return "I'm hero! " + user.username
```

class EmitterMixin.**Meta**
Emitter options. Setup parameters for resource's serialization.

```
class SomeResource(EmitterMixin, View):

    class Meta:
        emitters = JSONEmitter
```

emitters
alias of JSONEmitter

classmethod EmitterMixin.**determine_emitter**(*request*)
Get emitter for request.

Return emitter Instance of adrest.utils.emitters.BaseEmitter

EmitterMixin.**emit**(*content*, *request=None*, *emitter=None*)
Serialize response.

Return response Instance of django.http.Response

static EmitterMixin.**to_simple**(*content*, *simple*, *serializer=None*)
Abstract method for modification a structure before serialization.

Parameters

- **content** – response from called method
- **simple** – structure is prepared to serialization
- **serializer** – current serializer

Return object structure for serialization

```
class SomeResource(ResourceView):
    def get(self, request, **resources):
        return dict(true=False)

    def to_simple(self, content, simple, serializer):
        simple['true'] = True
        return simple
```

4.4.3 ParserMixin

class adrest.mixin.parser.**ParserMixin**
Parse user data.

4.4.4 ThrottleMixin

class adrest.mixin.throttle.**ThrottleMixin**
Throttle request.

4.4.5 AuthMixin

class adrest.mixin.auth.**AuthMixin**(**args*, ***kwargs*)
Adds pluggable authentication behaviour.

4.4.6 DynamicMixin

class adrest.mixin.handler.**DynamicMixin**(**args*, ***kwargs*)
Implement filters and sorting.

ADRest DynamicMixin supports filtering and sorting collection from query params.

4.4.7 HandlerMixin

class adrest.mixin.handler.**HandlerMixin**(**args*, ***kwargs*)
Implement REST API.

class Meta
Handler options. Setup parameters for REST implementation.

```
class SomeResource(HandlerMixin, View):

    class Meta:
        allowed_methods = 'get', 'post'
        model = 'app.model'
```

Meta.**allowed_methods = ('GET',)**
List of allowed methods (or simple one)

Meta.**callmap = {'HEAD': 'head', 'GET': 'get', 'PATCH': 'patch', 'PUT': 'put', 'POST': 'post', 'OPTIONS': 'op**
Map HTTP methods to handler methods

Meta.**form = None**
Set form for resource by manual

Meta.**form_exclude = None**
Exclude field's names for automatic a model form

Meta.**form_fields = None**
Specify field's names for automatic a model form

Example:

```
class SomeResource(HandlerMixin, View):

    class Meta:
        allowed_methods = 'get', 'post'
        model = 'app.model'

    def dispatch(self, request, **resources):

        self.check_method_allowed(request)
```

```
        resources = self.get_resources(request, **resources)

        return self.handle_request(request, **resources)
```

Note: Documentation in construction.

CHAPTER 5

Bug tracker

If you have any suggestions, bug reports or annoyances please report them to the issue tracker at https://github.com/klen/adrest/issues

CHAPTER 6

Contributing

Development of adrest happens at github: https://github.com/klen/adrest

CHAPTER 7

Contributors

- klen (Kirill Klenov)

CHAPTER 8

License

Licensed under a GNU lesser general public license.

a