django-rest-assured Documentation

Release 0.2.0

Yehonatan Daniv

Sep 27, 2017

Contents

Instantly test-cover your Django REST Framework based API.

Django-REST-Assured adds another layer on top of Django REST Framework's APITestCase which allows covering a set of RESTful resource's endpoints with a single class declaration.

This gives both a quick coverage of sanity tests to your API and a more DRY and more friendly platform for writing additional, more comprehensive tests.

CHAPTER 1

As easy as

```
class CategoryTestCase(ReadWriteRESTAPITestCaseMixin, BaseRESTAPITestCase):

    base_name = 'category'
    factory_class = CategoryFactory
    create_data = {'name': 'comedy'}
    update_data = {'name': 'horror'}
```

Django-REST-Assured is designed to work with factory_boy for mocking objects to test against. However, you can easily extend the `BaseRESTAPITestCase` to work directly with Django Models or any other factory.

CHAPTER 2

Main features

- Class-based declarative API for creating tests.
- Covers the stack through: `route > view > serializer > model`.
- Uses Django REST Framework's conventions to minimize configuration.
- All tests return the response object for more extensive assertions.
- Automatic mocking of authentication if a user factory is provided.

CHAPTER 3

Usage

The basic form of usage is simply to create a class that extends any mixin from `rest_assured.testcases`, according to the endpoints you wish to cover, and the `BaseRESTAPITestCase` class.

Then just set the required attributes, and continue extending it from there.

example

```
class CategoryAPITestCase(ReadWriteRESTAPITestCaseMixin, BaseRESTAPITestCase):

    base_name = 'category'
    factory_class = Category
    create_data = {'name', 'documentary'}
    update_data = {'name', 'horror'}
```

If your API requires authentication and/or authorization just add a user factory class. Assuming you use factory_boy:

example

```
# in some factories.py module in your accounts app
class User(factory.DjangoModelFactory):

    class Meta:
        model = User
        exclude = ('raw_password',)

    first_name = 'Robert'
    last_name = factory.Sequence(lambda n: 'Paulson the {0}'.format(n))
    email = factory.sequence(lambda n: 'account{0}@example.com'.format(n))
    username = 'mayhem'
    raw_password = '123'
    password = factory.PostGenerationMethodCall('set_password', raw_password)
    is_active = True
```

```
# now back in your tests.py module
class CategoryAPITestCase(ReadWriteRESTAPITestCaseMixin, BaseRESTAPITestCase):

    base_name = 'category'
    factory_class = Category
    # see here:
    user_factory = User
    create_data = {'name', 'documentary'}
    update_data = {'name', 'horror'}
```

CHAPTER 4

Supports

Tests run against:

- Django 1.6, 1.7, 1.8 & 1.9.
- Django REST Framework 2.4.3, 2.4.4, 3.0, 3.1, 3.2 & 3.3.
- Python 2.7, 3.3, 3.4 & 3.5 (3.2 should work but is not tested).

CHAPTER 5

Installation

PyPI: https://pypi.python.org/pypi/django-rest-assured

```
$ pip install django-rest-assured
```

Source: https://github.com/ydaniv/django-rest-assured

```
$ git clone https://github.com/ydaniv/django-rest-assured
$ python setup.py install
```

CHAPTER 6

Contributing

Issues are tracked in the github repository.

Pull requests are welcome!

CHAPTER 7

Running tests

```
$ pip install pytest pytest-django
$ py.test
```

CHAPTER 8

License

Django-REST-Assured is distributed under the BSD license.

CHAPTER 9

Table of Contents

Reference

class rest_assured.testcases.**BaseRESTAPITestCase**(*methodName='runTest'*)

Base test case class for testing REST API endpoints.

base_name = None

required: Base route name of the API endpoints to test.

factory_class = None

required: The factory class to use for creating the main object to test against.

LIST_SUFFIX = '-list'

Suffix for list endpoint view names. Defaults to `'-list'`.

DETAIL_SUFFIX = '-detail'

Suffix for detail endpoint view names. Defaults to `'-detail'`.

lookup_field = 'pk'

The field to use for DB and route lookups. Defaults to `'pk'`.

user_factory = None

User factory to use in case you need user authentication for testing. Defaults to `None`.

object = None

The main test subject.

user = None

The user instance created if the `user_factory` is set and used. Defaults to `None`.

get_factory_class()

Return the factory class for generating the main object (or model instance) of this test case.

By default this gets the `factory_class` attribute of this class.

Returns Factory class used for creating the mock objects.

get_object (*factory*)
Create and return the object (or model instance) of this test case.

By default this calls the `create()` method of the factory class, assuming a Django Model or a factory_boy's Factory.

Parameters **factory** – The factory class used for creating

Returns The main object of this test case.

setUp()
Generates the main object and user instance if needed.

The user instance will be created only if the `user_factory` attribute is set to the factory class.

If there is an available user instance, that user will be force authenticated.

class rest_assured.testcases.**ListAPITestCaseMixin**
Adds a list view test to the test case.

pagination_results_field = None
When using pagination set this attribute to the name of the property in the response data that holds the result set. Defaults to `None`.

get_list_url()
Return the list endpoint url.

Returns The url of list endpoint.

get_list_response(***kwargs*)
Send the list request and return the response.

Parameters **kwargs** – Extra arguments that are passed to the client's `get()` call.

Returns The response object.

test_list(***kwargs*)
Send request to the list view endpoint, verify and return the response.

Checks for a 200 status code and that there is a `results` property in the `response.data`.

You can extend it for more extensive checks.

example

```
class LanguageRESTAPITestCase(ListAPITestCaseMixin, BaseRESTAPITestCase):

    def test_list(self, **kwargs):
        response = super(LanguageRESTAPITestCase, self).test_list(**kwargs)
        results = response.data.get('results')
        self.assertEqual(results[0].get('code'), self.object.code)
```

Parameters **kwargs** – Extra arguments that are passed to the client's `get()` call.

Returns The view's response.

class rest_assured.testcases.**DetailAPITestCaseMixin**
Adds a detail view test to the test case.

get_detail_url()
Return the detail endpoint url.

Returns The url of detail endpoint.

get_detail_response(***kwargs*)
Send the detail request and return the response.

Parameters **kwargs** – Extra arguments that are passed to the client's `get()` call.

Returns The response object.

test_detail(***kwargs*)
Send request to the detail view endpoint, verify and return the response.

Checks for a 200 status code and that there is an `id` property in the `response.data` and that it equals the main object's id.

You can extend it for more extensive checks.

---

example

```
class LanguageRESTAPITestCase(DetailAPITestCaseMixin, BaseRESTAPITestCase):

    def test_list(self, **kwargs):
        response = super(LanguageRESTAPITestCase, self).test_list(**kwargs)
        self.assertEqual(response.data.get('code'), self.object.code)
```

Using a callable in `attributes_to_check`:

---

example

```
class TaggedFoodRESTAPITestCase(DetailAPITestCaseMixin, BaseRESTAPITestCase):

    attributes_to_check = ['name', ('similar', lambda obj: obj.tags.similar_
↪objects())]
```

Parameters **kwargs** – Extra arguments that are passed to the client's `get()` call.

Returns The view's response.

class rest_assured.testcases.**CreateAPITestCaseMixin**
Adds a create view test to the test case.

create_data = None
required: Dictionary of data to use as the POST request's body.

response_lookup_field = 'id'
The name of the field in the response data for looking up the created object in DB.

get_create_data()
Return the data used for the create request.

By default gets the `create_data` attribute of this class.

Returns The data dictionary.

get_create_url()
Return the create endpoint url.

Returns The url of create endpoint.

get_create_response(*data=None*, ***kwargs*)
Send the create request and return the response.

Parameters

- **data** – A dictionary of the data to use for the create request.
- **kwargs** – Extra arguments that are passed to the client's `post()` call.

Returns The response object.

get_lookup_from_response(*data*)
Return value for looking up the created object in DB.

Note The created object will be looked up using the `lookup_field` attribute as key, which defaults to `pk`.

Parameters **data** – A dictionary of the response data to lookup the field in.

Returns The value for looking up the

test_create(*data=None*, ***kwargs*)
Send request to the create view endpoint, verify and return the response.

Also verifies that the object actually exists in the database.

Parameters

- **data** – A dictionary of the data to use for the create request.
- **kwargs** – Extra arguments that are passed to the client's `post()` call.

Returns A tuple `response, created` of the view's response the created instance.

class rest_assured.testcases.**DestroyAPITestCaseMixin**
Adds a destroy view test to the test case.

get_destroy_url()
Return the destroy endpoint url.

Returns The url of destroy endpoint.

get_destroy_response(***kwargs*)
Send the destroy request and return the response.

Parameters **kwargs** – Extra arguments that are passed to the client's `delete()` call.

Returns The view's response.

test_destroy(***kwargs*)
Send request to the destroy view endpoint, verify and return the response.

Also verifies the object does not exist anymore in the database.

Parameters **kwargs** – Extra arguments that are passed to the client's `delete()` call.

Returns The view's response.

class rest_assured.testcases.**UpdateAPITestCaseMixin**
Adds an update view test to the test case.

use_patch = True
Whether to send a PATCH request instead of PUT. Defaults to `True`.

update_data = None
required: Dictionary of data to use as the update request's body.

update_results = None
Dictionary mapping attributes to values to check against the updated instance in the database. Defaults to `update_data`.

relationship_lookup_field = 'id'
The name of the field in the response data for looking up the created object in DB.

get_update_url()
Return the update endpoint url.

Returns The url of update endpoint.

get_update_response(*data=None*, *results=None*, *use_patch=None*, ***kwargs*)
Send the update request and return the response.

Parameters

- **data** – Data dictionary for the update request.
- **results** – Dictionary mapping instance properties to expected values.
- **kwargs** – Extra arguments that are passed to the client's `put()` or `patch()` call.

Returns The response object.

get_update_data()
Return the data used for the update request.

By default gets the `update_data` attribute of this class.

Returns Data dictionary for the update request.

get_update_results(*data=None*)
Return a dictionary of the expected results of the instance.

By default gets the `update_results` attribute of this class. If that isn't set defaults to the data.

Parameters **data** – The update request's data dictionary.

Returns Dictionary mapping instance properties to expected values.

get_relationship_value(*related_obj*, *key*)
Return a value representing a relation to a related model instance.

By default gets the `relationship_lookup_field` attribute of this class which defaults to `id`, and converts it to a `string`.

Parameters

- **related_obj** – The related model instance to convert to a value.
- **key** – A `string` representing the name of the relation, or the key on the updated object.

Returns Value representing the relation to assert against.

test_update(*data=None*, *results=None*, *use_patch=None*, ***kwargs*)
Send request to the update view endpoint, verify and return the response.

Parameters

- **data** – Data dictionary for the update request.
- **results** – Dictionary mapping instance properties to expected values.
- **kwargs** – Extra arguments that are passed to the client's `put()` or `patch()` call.

Returns A tuple `response, updated` of the view's response the updated instance.

class rest_assured.testcases.**ReadRESTAPITestCaseMixin**
Adds the read CRUD operations tests to the test case.

Includes: *ListAPITestCaseMixin*, *DetailAPITestCaseMixin*.

class rest_assured.testcases.**WriteRESTAPITestCaseMixin**
Adds the write CRUD operations tests to the test case.

Includes: *CreateAPITestCaseMixin*, *UpdateAPITestCaseMixin*, *DestroyAPITestCaseMixin*.

class rest_assured.testcases.**ReadWriteRESTAPITestCaseMixin**
A complete API test case that covers all successful CRUD operation requests.

Includes: *ReadRESTAPITestCaseMixin*, *WriteRESTAPITestCaseMixin*.

class rest_assured.contrib.drf_fsm_transitions.**TransitionAPITestCaseMixin**
Adds the `transition()` method for testing state transition API endpoints.

This is a handy extension for quickly test-covering API endpoints that are generated using the DRF-FSM-Transition library.

transition(*result*, *route*, *attribute='status'*, *from_state=None*, *data=None*)
Send request to a transition view endpoint, verify and return the response.

Parameters

- **result** – The expected value of the instance's `attribute`.
- **route** – The addition to the route, usually the name of the transition action's name.
- **attribute** – Name of the instance's attribute that holds the state.
- **from_state** – A state to update the object to, to initialize the "from" state.

Returns The view's response.

Tutorial

Note: You can clone this example and run the tests yourself from: https://github.com/ydaniv/django-rest-assured-demo.

Let's take a look at an example from the Django documentation of a Weblog application:

```
from django.db import models


class Blog(models.Model):

    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __str__(self):              # __unicode__ on Python 2
        return self.name


class Author(models.Model):

    name = models.CharField(max_length=50)
```

```
    email = models.EmailField()

    def __str__(self):              # __unicode__ on Python 2
        return self.name


class Entry(models.Model):

    blog = models.ForeignKey(Blog)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    mod_date = models.DateField(auto_now=True)
    authors = models.ManyToManyField(Author)
    n_comments = models.IntegerField()
    n_pingbacks = models.IntegerField()
    rating = models.IntegerField()

    def __str__(self):              # __unicode__ on Python 2
        return self.headline
```

The above file will serve as the `models.py` file in this example application.

Say that we want to have a RESTful API endpoint for the `Entry` model. We'll need a serializer for `Entry` objects, so this will serve as our `serializers.py`:

```
from rest_framework import serializers
from . import models


class Entry(serializers.ModelSerializer):

    class Meta:
        model = models.Entry
```

Now we're ready to define our views. The following shall serve as `views.py`:

```
from rest_framework import viewsets
from . import models, serializers


class Entries(viewsets.ModelViewSet):

    queryset = models.Entry.objects.all()
    serializer_class = serializers.Entry
```

And hooking that viewset with URL's, we'll add a `urls.py` and define a router:

```
from django.conf.urls import url, include
from rest_framework import routers
from . import views


router = routers.DefaultRouter()
router.register(r'entries', views.Entries)


urlpatterns = [
```

```
    url(r'^', include(router.urls)),
]
```

And we'll assume the pattern above is added to the project's root `urlpatterns` under the prefix `/api/`, so that our endpoint will look like `/api/entries/`.

Now we have an API endpoint we can test. Yay!

To make things even easier we'll create a `factories.py` file that will include factories for our models using Factory Boy:

```
import datetime
import factory
from factory import fuzzy
from . import models


class Blog(factory.DjangoModelFactory):

    class Meta:
        model = models.Blog

    name = factory.Sequence(lambda n: 'Blog {0}'.format(n))
    tagline = factory.Sequence(lambda n: 'Blog {0} tag line'.format(n))


class Author(factory.DjangoModelFactory):

    class Meta:
        model = models.Author

    name = factory.Sequence(lambda n: 'Author {0}'.format(n))
    email = factory.Sequence(lambda n: 'author{0}@example.com'.format(n))


class Entry(factory.DjangoModelFactory):

    class Meta:
        model = models.Entry

    blog = factory.SubFactory(Blog)
    headline = factory.Sequence(lambda n: 'OMG Headline {0}!'.format(n))
    body_text = fuzzy.FuzzyText(length=100)
    pub_date = datetime.date(2014, 11, 12)
    mod_date = datetime.date(2014, 11, 12)
    rating = fuzzy.FuzzyInteger(low=1, high=5, step=1)
    n_pingbacks = 0
    n_comments = 0

    @factory.post_generation
    def authors(self, create, extracted, **kwargs):
        if not create:
            return

        if extracted:
            for author in extracted:
                self.authors.add(author)
```

This will make testing fun.

Let's write the tests! This shall be our `tests.py` file:

```
from rest_assured.testcases import ReadWriteRESTAPITestCaseMixin, BaseRESTAPITestCase
from . import factories


class EntryAPITestCase(ReadWriteRESTAPITestCaseMixin, BaseRESTAPITestCase):

    base_name = 'entry' # this is the base_name generated by the DefaultRouter
    factory_class = factories.Entry
    update_data = {'rating': 5}

    def setUp(self):
       self.author = factories.Author.create()
       super(EntryAPITestCase, self).setUp()

    def get_object(self, factory):
        return factory.create(authors=[self.author])

    def get_create_data(self):
       return {'headline': 'Lucifer Sam',
               'body_text': 'is a song by British psychedelic rock band Pink Floyd.',
               'authors': [self.author.pk],
               'rating': 4,
               'n_pingbacks': 0,
               'n_comments': 0,
               'pub_date': datetime.date(2014, 11, 12),
               'blog': self.object.blog.pk}
```

And that's it!

This simple class will make 5 tests if we'll run:

```
$ python manage.py test
```

And will produce an output like such:

```
user@machine:~/project$ python manage.py test
Creating test database for alias 'default'...
.....
----------------------------------------------------------------------
Ran 5 tests in 0.155s

OK
Destroying test database for alias 'default'...
```

You can see the above example is not entirely trivial. We had to do some setup work to ensure we have a ready made Author instance. We also created dynamic getters for the main test object and the data dict used for the create request. In both cases this was required to obtain a lazy reference to the Author instance we created in `setUp()`.

Say now our API is not public and requires authentication (token, session, etc.). We'll need a user factory to mock authenticated requests. Let's create that factory:

```
from django.contrib import auth


class User(factory.DjangoModelFactory):

    class Meta:
```

```
        model = auth.get_user_model()
        exclude = ('raw_password',)

    first_name = 'Robert'
    last_name = factory.Sequence(lambda n: 'Paulson the {0}'.format(n))
    email = factory.sequence(lambda n: 'account{0}@example.com'.format(n))
    username = 'mayhem'
    raw_password = '123'
    password = factory.PostGenerationMethodCall('set_password', raw_password)
    is_active = True
```

Our tests now will fail, since all responses will return a `HTTP_401_UNAUTHORIZED` status code. Which is great.

Assuming that `User` factory resides in the previous `factories.py` module, we add a `user_factory` attribute to our test case:

```
...
user_factory = factories.User
...
```

The full version of our `tests.py` now look like:

```
from rest_assured.testcases import ReadWriteRESTAPITestCaseMixin, BaseRESTAPITestCase
from . import factories


class EntryAPITestCase(ReadWriteRESTAPITestCaseMixin, BaseRESTAPITestCase):

    base_name = 'entry' # this is the base_name generated by the DefaultRouter
    factory_class = factories.Entry
    user_factory = factories.User # this is the user that will be authenticated for␣
↪testing
    update_data = {'rating': 5}

    def setUp(self):
       self.author = factories.Author.create()
       super(EntryAPITestCase, self).setUp()

    def get_object(self, factory):
        return factory.create(authors=[self.author])

    def get_create_data(self):
       return {'headline': 'Lucifer Sam',
               'body_text': 'is a song by British psychedelic rock band Pink Floyd.',
               'authors': [self.author.pk],
               'rating': 4,
               'n_pingbacks': 0,
               'n_comments': 0,
               'pub_date': datetime.date(2014, 11, 12),
               'blog': self.object.blog.pk}
```

And our tests pass again.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

S

T

U

W