Making web api testing easy

Who Am I?

- 28 years old, from Givatayim.
- Ex-8200.
- Worked in several startups.
- Was a digital nomad.
- Currently working at bluevine.
- Creator of drf-api-action.

Github: Ori-Roza

Linkedin: Ori Roza

Lecture Agenda

- **drf-api-action?**

- **Intro main components in django-rest-framework**
  - What is a Serializer?
  - What is the @action decorator?
  - What is a ViewSet?

- **drf-api-action internals**
  - Mocking Request object
  - Injecting our mocked request and serializer class
  - Put it all together

- **drf-api-action benefits & examples**
  - Explicitly & Simplicity
  - Method traceback is explicit
  - Pagination

drf-api-action?

- A pytest plugin for django-rest-framework endpoint testing.
- Giving the ability to call endpoint as they where a simple functions.
- Support in exception traceback, pagination...

drf-api-action?

```python
from rest_framework.test import RequestsClient
from tests.test_server.test_app.models import DummyModel


def test_call_as_api_fixture(db):
    dummy_model = DummyModel()
    dummy_model.dummy_int = 1
    dummy_model.save()
    client = RequestsClient()
    res = client.get('http://testserver/api/django-rest/api_dummy')
    assert res.json()['results'][0]['dummy_int']
```

drf-api-action?

```python
from tests.test_server.test_app.models import DummyModel
from tests.test_server.test_app.views import DummyViewSetFixture, DummyAPIViewSet


@pytest.mark.api_action(view_set_class=DummyViewSetFixture)
def test_call_as_api_fixture(db, api_action):
    dummy_model = DummyModel()
    dummy_model.dummy_int = 1
    dummy_model.save()
    res = api_action.api_dummy(pk=1)
    assert res["dummy_int"] == 1
```

What is a Serializer?

Intro main components in django-rest-framework

Based on docs:

“””

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types,
after first validating the incoming data.

“””

What is a Serializer?

Intro main components in django-rest-framework

```
class AddUserSerializer(Serializer):
    id = IntegerField(required=False)
    first_name = CharField(required=True, max_length=32)
    last_name = CharField(required=True, max_length=32)
    age = IntegerField(required=False, max_value=120)

    class Meta:
        model = User
        fields = ("*", )

    def validate(self, attrs):
        if 'age' not in attrs:
            return attrs

        age = attrs['age']

        if age and age > 120:
            raise ValidationError(detail='age must be between 0 and 120')
        return attrs

    def create(self, validated_data):
        instance = self.Meta.model(**validated_data)
        instance.save()
        return instance
```

What is a Serializer?

Intro main components in django-rest-framework

```
@action(detail=False,
        methods=['post'],
        url_path='/',
        url_name='users/',
        serializer_class=serializers.AddUserSerializer)
def add_user(self, request, **kwargs):
    """
    adds new user
    """
    serializer = self.get_serializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    serializer.save()
    return Response(data=serializer.data, status=status.HTTP_201_CREATED)
```

What is @action decorator?

Intro main components in django-rest-framework

Based on docs:

> “””
>
> If you have ad-hoc methods that should be routable,
> you can mark them as such with the action decorator.
>
> “””

This decorator is very straightforward.
It defines a route in a ViewSet and usually requires the following arguments:

- **detail**: whether the call expects to get PK or not.
- **methods**: which methods the call supports.
- **serializer_class**: the serializer that handles the specific view.

What is a ViewSet?

Intro main components in django-rest-framework

```python
class DummyAPIViewSet(ModelViewSet):

    @action(detail=True, methods=["get"], serializer_class=DummySerializer)
    def api_dummy(self, request, **kwargs):
        serializer = self.get_serializer(instance=self.get_object())
        return Response(data=serializer.data, status=status.HTTP_200_OK)

    @action(detail=False, methods=["post"], serializer_class=GetDummyByIntSerializer)
    def by_dummy_int(self, request, **kwargs):
        self.get_serializer(data=request.data).is_valid(raise_exception=True)
        queryset = DummyModel.objects.filter(dummy_int=request.data["dummy_int"])
        page = self.paginate_queryset(queryset)
        serializer = self.get_serializer(page, many=True)
        return self.get_paginated_response(serializer.data)
```

Mocking Request object

drf-api-action internals

- Each endpoint method expects an HTTPRequest object
- Request data commonly represented in request object by `.data` or `query_params` attributes.
- Request is passed to the endpoint & needs to be injected in ViewSet.
- We need minimum requirements to create our mocked Request object.

Mocking Request object

drf-api-action internals

```
class CustomRequest:
    """
    Mock for a custom request
    """

    def __init__(self, data, query_params):
        self.data = data
        self.query_params = query_params

    def build_absolute_uri(self, _=None):
        """
        mocking django/http/request.py::HTTPRequest::build_absolute_uri
        It's irrelevant since we do not provide any web resource
        """
        return ''
```

Injecting our mocked request and serializer class

drf-api-action internals

- Each endpoint method uses a serializer from action decorator's `serializer_class` param or default one from viewset.
- DRF injects the right serializer to each endpoint in runtime.
- We want to follow this behavior and inject the corresponded serializer on each method run.

Injecting our mocked request and serializer class

drf-api-action internals

First, extract serializer_class parameter from endpoint's @action:

func - an endpoint (method) from our ViewSet (we will get to it later)

```
def run_function(self, func):
    def api_item(*args, **kwargs):
        # here we retrieve serializer_class from @action decorator in order to inject it
        serializer_class = func.kwargs['serializer_class']
        return run_as_api(self, func, serializer_class, *args, **kwargs)

    return api_item
```

Injecting our mocked request and serializer class

drf-api-action internals

Second, we will inject our objects into our viewset instance:

```
def run_as_api(self, func, serializer_class, *args, **kw):
    kw.update({"serializer_class": serializer_class})
    request = CustomRequest(kw, kw)
    self.kwargs = kw
    self.request = request
    ...
```

Injecting our mocked request and serializer class

drf-api-action internals

Third, we will override get_serializer method:

```
class APIRestMixin(viewsets.GenericViewSet):
    """
    creates our custom get_serializer in order to use our serializer injection
    """
    def get_serializer(self, query_set=None, *args, **kwargs):
        serializer_class = self.kwargs.get("serializer_class", self.serializer_class)
        return serializer_class(*args, **kwargs)
```

Put it All Together

drf-api-action internals

```
@pytest.fixture
def api_action(request):
    """
    Make Dango WebView endpoints accessible
    """
    from drf_api_action.mixins import APIRestMixin  # pylint: disable=import-outside-

    if request.keywords['api_action'].kwargs.get("view_set_class") is None:
        raise ActionsAPIException('using api_action fixture must require a view_set_c

    view_set_class = request.keywords['api_action'].kwargs["view_set_class"]

    class WrapperApiClass(APIRestMixin, view_set_class):
        def __getattribute__(self, item):
            class_attribute = super().__getattribute__(item)

            # running our logic on endpoints only
            if callable(class_attribute) and hasattr(class_attribute, 'detail'):
                return run_function(self, class_attribute)

            return class_attribute

    api = WrapperApiClass()
    return api
```

Put it All Together

drf-api-action internals

```
def run_function(self, func):
    def api_item(*args, **kwargs):
        # here we retrieve serializer_class from @action decorator in order to inject it
        serializer_class = func.kwargs['serializer_class']
        return run_as_api(self, func, serializer_class, *args, **kwargs)

    return api_item
```

Put it All Together

drf-api-action internals

```
def run_as_api(self, func, serializer_class, *args, **kw):
    # adding to the view the request & kwargs including the seria
    kw.update({"serializer_class": serializer_class})  # adding s
    # decorator into our instance
    request = CustomRequest(kw, kw)
    self.kwargs = kw  # adding our enhanced kwargs into instance
    self.request = request  # mocking request with our arguments

    ret = func(request, **kw)  # evaluating endpoint
    if isinstance(ret.data, list):  # multiple results
        results = [dict(res) for res in ret.data]
    else:  # only one json
        results = {k.lower(): v for k, v in ret.data.items()}

    return results
```

Explicitly & Simplicity

drf-api-action benefits & examples

- No code addition needed in ViewSet class.
- No need to use `reverse` function or use an explicit url.
- Calling endpoints is easy.

Explicitly & Simplicity

drf-api-action benefits & examples

```python
from tests.test_server.test_app.models import DummyModel
from tests.test_server.test_app.views import DummyViewSetFixture, DummyAPIViewSet


@pytest.mark.api_action(view_set_class=DummyViewSetFixture)
def test_call_as_api_fixture(db, api_action):
    dummy_model = DummyModel()
    dummy_model.dummy_int = 1
    dummy_model.save()
    res = api_action.api_dummy(pk=1)
    assert res["dummy_int"] == 1
```

Method traceback are explicit

drf-api-action benefits & examples

Instead of getting 500 or 400 we can catch the specific exception!

Method traceback are explicit

drf-api-action benefits & examples

```python
import pytest
from rest_framework.exceptions import ValidationError

from drf_api_action.utils import extract_page_number
from tests.test_server.test_app.models import DummyModel
from tests.test_server.test_app.views import DummyViewSetFixture, DummyAPIViewSet


@pytest.mark.api_action(view_set_class=DummyAPIViewSet)
def test_exceptions(db, api_action):
    dummy_model = DummyModel()
    dummy_model.dummy_int = 1
    dummy_model.save()
    with pytest.raises(ValidationError):
        _ = api_action.by_dummy_int(dummy_int=-1)
```

Method traceback are explicit

drf-api-action benefits & examples

```
@pytest.mark.api_action(view_set_class=DummyAPIViewSet)
def test_exceptions(db, api_action):
    dummy_model = DummyModel()
    dummy_model.dummy_int = 1
    dummy_model.save()
    #with pytest.raises(ValidationError):
>   _ = api_action.by_dummy_int(dummy_int=-1)

tests.py:51:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
../drf_api_action/plugin.py:11: in api_item
    return run_as_api(self, func, serializer_class, *args, **kwargs)
../drf_api_action/utils.py:29: in run_as_api
    ret = func(request, **kw)  # evaluating endpoint
test_server/test_app/views.py:61: in by_dummy_int
    self.get_serializer(data=request.data).is_valid(raise_exception=True)
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = GetDummyByIntSerializer(data={'dummy_int': -1, 'serializer_class': <class 'tests.
    id = IntegerField(required=False)
    dummy_int = IntegerField()

    def is_valid(self, *, raise_exception=False):
        assert hasattr(self, 'initial_data'), (
            'Cannot call `.is_valid()` as no `data=` keyword argument was '
            'passed when instantiating the serializer instance.'
        )

        if not hasattr(self, '_validated_data'):
            try:
                self._validated_data = self.run_validation(self.initial_data)
            except ValidationError as exc:
                self._validated_data = {}
                self._errors = exc.detail
            else:
                self._errors = {}

        if self._errors and raise_exception:
>           raise ValidationError(self.errors)
E           rest_framework.exceptions.ValidationError: {'error': [ErrorDetail(string=
'dummy_int must be greater equal than 0', code=400)]}
```

Pagination

drf-api-action benefits & examples

We can test pagination mechanism and get bulks!

Pagination

drf-api-action benefits & examples

```python
from drf_api_action.utils import extract_page_number
from tests.test_server.test_app.models import DummyModel
from tests.test_server.test_app.views import DummyViewSetFixture, DummyAPIViewSet


@pytest.mark.api_action(view_set_class=DummyAPIViewSet)
def test_pagination_data(db, api_action):
    for i in range(1, 3):
        dummy_model = DummyModel()
        dummy_model.dummy_int = 1
        dummy_model.save()

    response = api_action.by_dummy_int(dummy_int=1)
    assert extract_page_number(response['next']) == 2

    obj = response['results'][0]
    assert obj['dummy_int'] == 1

    response = api_action.by_dummy_int(dummy_int=1, page=2)
    assert extract_page_number(response['previous']) == 1
    assert extract_page_number(response['next']) is None
```

Thank You!!!

Any Questions?