GenericAPIView

This class extends REST framework's APIView class, adding commonly required behavior for standard list and detail views.

Attributes

queryset - The queryset that should be used for returning objects from this view. Typically, you must either set this attribute, or override the get_queryset() method. If you are overriding a view method, it is important that you call get_queryset() instead of accessing this property directly, as queryset will get evaluated once, and those results will be cached for all subsequent requests.

serializer_class - The serializer class that should be used for validating and deserializing input, and for serializing output. Typically, you must either set this attribute, or override the get_serializer_class() method.

GenericAPIView

lookup_field - The model field that should be used to for performing object lookup of individual model instances. Defaults to 'pk’.

lookup_url_kwarg - The URL keyword argument that should be used for object lookup. The URL conf should include a keyword argument corresponding to this value. If unset this defaults to using the same value as lookup_field.

pagination_class - The pagination class that should be used when paginating list results. Defaults to the same value as the DEFAULT_PAGINATION_CLASS setting, which is 'rest_framework.pagination.PageNumberPagination’. Setting pagination_class=None will disable pagination on this view.

filter_backends - A list of filter backend classes that should be used for filtering the queryset. Defaults to the same value as the DEFAULT_FILTER_BACKENDS setting.

GenericAPIView

Methods

get_queryset(self) - It returns the queryset that should be used for list views, and that should be used as the base for lookups in detail views. Defaults to returning the queryset specified by the queryset attribute.

This method should always be used rather than accessing self.queryset directly, as self.queryset gets evaluated only once, and those results are cached for all subsequent requests.

get_object(self) - It returns an object instance that should be used for detail views. Defaults to using the lookup_field parameter to filter the base queryset.

get_serializer_class(self) - It returns the class that should be used for the serializer. Defaults to returning the serializer_class attribute.

GenericAPIView

get_serializer_context(self) – It returns a dictionary containing any extra context that should be supplied to the serializer. Defaults to including 'request', 'view' and 'format' keys.

get_serializer(self, instance=None, data=None, many=False, partial=False) – It returns a serializer instance.

get_paginated_response(self, data) – It returns a paginated style Response object.

paginate_queryset(self, queryset) - Paginate a queryset if required, either returning a page object, or None if pagination is not configured for this view.

GenericAPIView

filter_queryset(self, queryset) - Given a queryset, filter it with whichever filter backends are in use, returning a new queryset.

Mixins

One of the big wins of using class-based views is that it allows us to easily compose reusable bits of behaviour.

The create/retrieve/update/delete operations that we've been using so far are going to be pretty similar for any model-backed API views we create.

Those bits of common behaviour are implemented in REST framework's mixin classes.

The mixin classes provide the actions that are used to provide the basic view behavior.

Note that the mixin classes provide action methods rather than defining the handler methods, such as get() and post(), directly. This allows for more flexible composition of behavior.

The mixin classes can be imported from rest_framework.mixins

Mixins

- ListModelMixin
- CreateModelMixin
- RetrieveModelMixin
- UpdateModelMixin
- DestroyModelMixin

ListModelMixin

It provides a list(request, *args, **kwargs) method, that implements listing a queryset.

If the queryset is populated, this returns a 200 OK response, with a serialized representation of the queryset as the body of the response. The response data may optionally be paginated.

from rest_framework.mixins import ListModelMixin

from rest_framework.generics import GenericAPIView

ListModelMixin

```
from rest_framework.mixins import ListModelMixin
from rest_framework.generics import GenericAPIView
class StudentList(ListModelMixin, GenericAPIView):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)
```

CreateModelMixin

It provides a create(request, *args, **kwargs) method, that implements creating and saving a new model instance.

If an object is created this returns a 201 Created response, with a serialized representation of the object as the body of the response. If the representation contains a key named url, then the Location header of the response will be populated with that value.

If the request data provided for creating the object was invalid, a 400 Bad Request response will be returned, with the error details as the body of the response.

CreateModelMixin

```
from rest_framework.mixins import CreateModelMixin
from rest_framework.generics import GenericAPIView
class StudentCreate(CreateModelMixin, GenericAPIView):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
    def post(self, request, *args, **kwargs):
        return self.create(request, *args, **kwargs)
```

RetrieveModelMixin

It provides a retrieve(request, *args, **kwargs) method, that implements returning an existing model instance in a response.

If an object can be retrieved this returns a 200 OK response, with a serialized representation of the object as the body of the response. Otherwise it will return a 404 Not Found.

RetrieveModelMixin

```
from rest_framework.mixins import RetrieveModelMixin
from rest_framework.generics import GenericAPIView
class StudentRetrieve(RetrieveModelMixin, GenericAPIView):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
    def get(self, request, *args, **kwargs):
        return self.retrieve(request, *args, **kwargs)
```

UpdateModelMixin

It provides a update(request, *args, **kwargs) method, that implements updating and saving an existing model instance.

It also provides a partial_update(request, *args, **kwargs) method, which is similar to the update method, except that all fields for the update will be optional. This allows support for HTTP PATCH requests.

If an object is updated this returns a 200 OK response, with a serialized representation of the object as the body of the response.

If the request data provided for updating the object was invalid, a 400 Bad Request response will be returned, with the error details as the body of the response.

UpdateModelMixin

```
from rest_framework.mixins import UpdateModelMixin
from rest_framework.generics import GenericAPIView
class StudentUpdate(UpdateModelMixin, GenericAPIView):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
    def put(self, request, *args, **kwargs):
        return self.update(request, *args, **kwargs)
```

DestroyModelMixin

It provides a destroy(request, *args, **kwargs) method, that implements deletion of an existing model instance.

If an object is deleted this returns a 204 No Content response, otherwise it will return a 404 Not Found.

DestroyModelMixin

```
from rest_framework.mixins import DestroyModelMixin
from rest_framework.generics import GenericAPIView
class StudentDestroy(DestroyModelMixin, GenericAPIView):
    queryset = Student.objects.all()
    serializer_class = StudentSerializer
    def delete(self, request, *args, **kwargs):
        return self.destroy(request, *args, **kwargs)
```