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.

Let's take a look at how we can compose the views by using the mixin classes. Here's our views.py module again.

from snippets.models import Snippet

from snippets.serializers import SnippetSerializer

from rest\_framework import mixins

from rest\_framework import generics

class SnippetList(mixins.ListModelMixin, mixins.CreateModelMixin, generics.GenericAPIView):

queryset = Snippet.objects.all()

serializer\_class = SnippetSerializer

def get(self, request, \*args, \*\*kwargs):

return self.list(request, \*args, \*\*kwargs)

def post(self, request, \*args, \*\*kwargs):

return self.create(request, \*args, \*\*kwargs)

We'll take a moment to examine exactly what's happening here. We're building our view using GenericAPIView, and adding in ListModelMixin and CreateModelMixin.

The base class provides the core functionality, and the mixin classes provide the .list() and .create() actions. We're then explicitly binding the get and post methods to the appropriate actions. Simple enough stuff so far.

class SnippetDetail(mixins.RetrieveModelMixin, mixins.UpdateModelMixin,.DestroyModelMixin,.GenericAPIView):

queryset = Snippet.objects.all()

serializer\_class = SnippetSerializer

def get(self, request, \*args, \*\*kwargs):

return self.retrieve(request, \*args, \*\*kwargs)

def put(self, request, \*args, \*\*kwargs):

return self.update(request, \*args, \*\*kwargs)

def delete(self, request, \*args, \*\*kwargs):

return self.destroy(request, \*args, \*\*kwargs)

Pretty similar. Again we're using the GenericAPIView class to provide the core functionality, and adding in mixins to provide the .retrieve(), .update() and .destroy() actions.

Using the mixin classes we've rewritten the views to use slightly less code than before, but we can go one step further. REST framework provides a set of already mixed-in generic views that we can use to trim down our views.py module even more.

from snippets.models import Snippet

from snippets.serializers import SnippetSerializer

from rest\_framework import generics

class SnippetList(generics.ListCreateAPIView):

queryset = Snippet.objects.all()

serializer\_class = SnippetSerializer

class SnippetDetail(generics.RetrieveUpdateDestroyAPIView):

queryset = Snippet.objects.all()

serializer\_class = SnippetSerializer

LIST OF DIFFERENT API VIEWS

Generic API view

1. mixins.ListModelMixin

2. mixins.CreateModelmixin

3. mixins.RetrieveModelMixin

4. mixins.UpdateModelMixin

5. mixins.destroyModelMixin

Advanced

6. mixins.ListCreateAPIView

7. mixins.RetrieveUpdateDestroyAPIView