The Updated Serializer Is:

class ProductSerializer(serializers.ModelSerializer):

    class Meta:

        model = Product

        fields = ['id', 'title', 'description', 'slug', 'inventory', 'unit\_price', 'price\_with\_tax', 'collection']

    price\_with\_tax = serializers.SerializerMethodField(method\_name='calculate\_tax')

    def calculate\_tax(self, product: Product):

        return product.unit\_price \* Decimal(1.1)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Then To Save The Product, We Can Call save-Method Of Serializer:

@api\_view(['GET', 'POST'])

def product\_list(request):

    if request.method == 'GET':

        queryset = Product.objects.select\_related('collection').all()

        serializer = ProductSerializer(queryset, many=True, context={ 'request': request })

        return Response(serializer.data)

    elif request.method == 'POST':

        serializer = ProductSerializer(data = request.data)

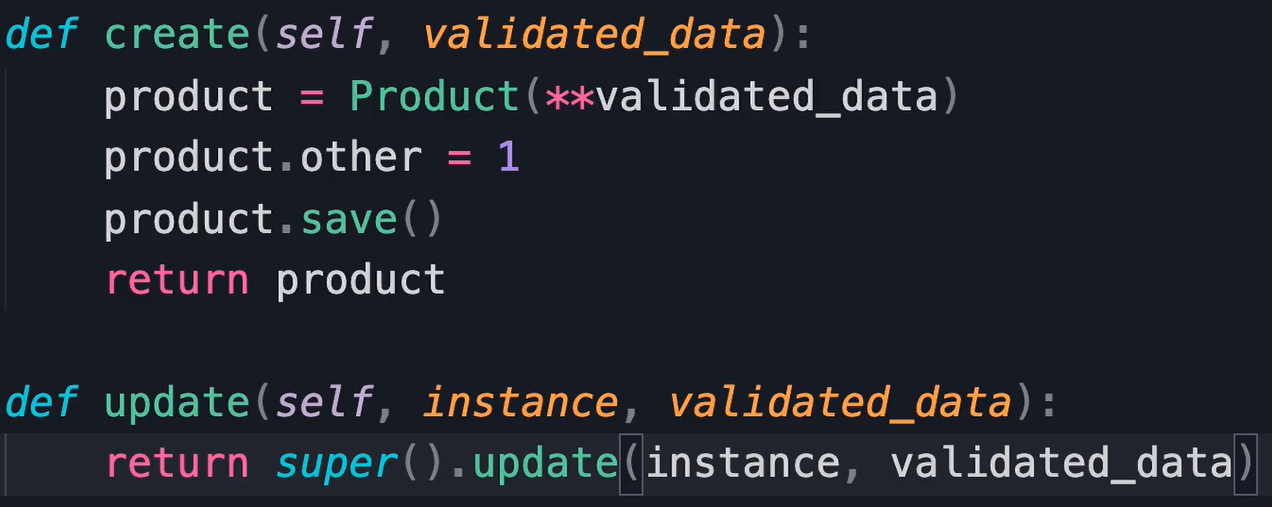
        serializer.is\_valid(raise\_exception=True)

        serializer.save()

        return Response('Ok')

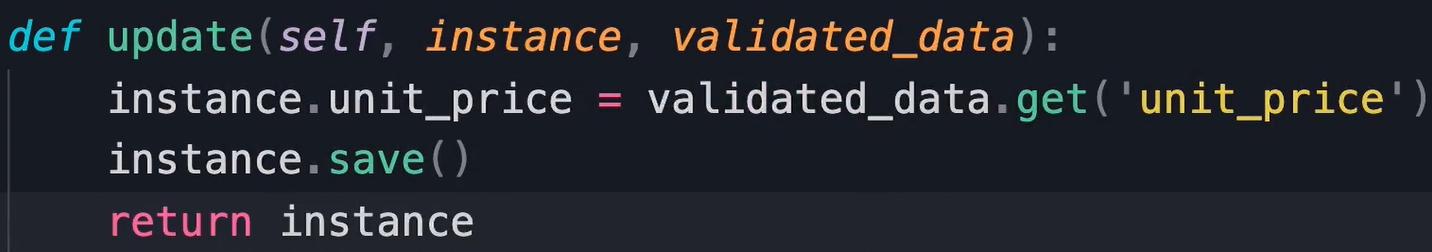
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If We Want To Override The Create Method Of Serializer:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If We Want To Override The Update-Method:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Update The Product Using PUT, OR PATCH, We Can Use:

@api\_view(['GET', 'PUT', 'PATCH'])

def product\_detail(request, id):

    product = get\_object\_or\_404(Product, pk=id)

    if request.method == 'GET':

        serializer = ProductSerializer(product, context={ 'request': request })

        return Response(serializer.data)

    elif request.method == 'PUT':

        serializer = ProductSerializer(product, data = request.data);

        serializer.is\_valid(raise\_exception=True)

        serializer.save()

        return Response(serializer.data)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Delete The Product, After We Check All Constraint:

@api\_view(['GET', 'PUT', 'DELETE'])

def product\_detail(request, id):

    product = get\_object\_or\_404(Product, pk=id)

    if request.method == 'GET':

        serializer = ProductSerializer(product, context={ 'request': request })

        return Response(serializer.data)

    elif request.method == 'PUT':

        serializer = ProductSerializer(product, data = request.data);

        serializer.is\_valid(raise\_exception=True)

        serializer.save()

        return Response(serializer.data)

    elif request.method == 'DELETE':

        if product.orderitems.count() > 0:

            return Response({'error': 'product cannot be deleted'}, status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

        product.delete()

        return Response(status=status.HTTP\_204\_NO\_CONTENT)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Note (From Me)*: Always Check The Number Of Queries That Are Executed For Each Serializer OR Model, And Use: ***select\_related***, ***prefetch\_related***, ***annotated***.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@api\_view(['GET', 'POST'])

def collection\_list(request):

    if request.method == 'GET':

        queryset = Collection.objects.annotate(products\_count=Count('product')).all();

        serializer = CollectionSerializer(queryset, many=True)

        return Response(serializer.data)

    elif request.method == 'POST':

        serializer = CollectionSerializer(data=request.data)

        serializer.is\_valid(raise\_exception=True)

        serializer.save()

        return Response(serializer.data, status=status.HTTP\_201\_CREATED)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class CollectionSerializer(serializers.ModelSerializer):

    class Meta:

        model = Collection

        fields = ['id', 'title', 'products\_count']

    products\_count = serializers.SerializerMethodField(method\_name='get\_products\_count')

    def get\_products\_count(self, collection: Collection):

        return collection.products\_count

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Create Class Based Views, We Can Use APIView-Class, That Represent The Main Class For Views-Based-Classes.

Note 1: In Class Based Views, Here We Don’t Have Many If-Statements.

from rest\_framework.views import APIView

class ProductList(APIView):

    def get(self, request):

        queryset = Product.objects.select\_related('collection').all()

        serializer = ProductSerializer(queryset, many=True, context={ 'request': request })

        return Response(serializer.data)

    def post(self, request):

        serializer = ProductSerializer(data = request.data)

        serializer.is\_valid(raise\_exception=True)

        serializer.save()

        return Response(serializer.data, status=status.HTTP\_201\_CREATED)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Set The URL Param:

class CollectionDetail(APIView):

    def get(self, request, pk: int):

        collection = get\_object\_or\_404(

            Collection.objects.annotate(products\_count=Count('products')),

            pk=pk

        )

        serializer = CollectionSerializer(collection)

        return Response(serializer.data)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Note*: To Encapsulate The Logic Of Creating, Listing, Updating, And Deleting We Can Use Mixins.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Note*: The Best Way To Use Mixins, is That Using Generic Views, That Encapsulate The Works Of Business Layer.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Use Generic Views For List OR Create Objects We Can Use:

Note: We Can Use This Way With Methods If We Have Logic Inside The Methods.

from rest\_framework.generics import ListCreateAPIView

class ProductList(ListCreateAPIView):

    def get\_queryset(self):

        return Product.objects.select\_related('collection').all()

    def get\_serializer\_class(self):

        return ProductSerializer

    def get\_serializer\_context(self):

        return { 'request': self.request }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In This Way We Can Use Attributes And Methods For Defining The QuerySet And SerializerClass.

class ProductList(ListCreateAPIView):

    queryset = Product.objects.select\_related('collection').all()

    serializer\_class = ProductSerializer

    def get\_serializer\_context(self):

        return { 'request': self.request }

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class CollectionSerializer(serializers.ModelSerializer):

    products\_count = serializers.IntegerField(read\_only=True)

    class Meta:

        model = Collection

        fields = ['id', 'title', 'products\_count']

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class CollectionDetail(RetrieveUpdateDestroyAPIView):

    queryset = Collection.objects.annotate(products\_count=Count('products')).all()

    serializer\_class = CollectionSerializer

    # lookup\_field = 'id'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For using ModelViewSet To Implement The Logic That We Want:

***Note***: In This Way, We Delete The Duplication In Generic Views

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.select\_related('collection').all()

    serializer\_class = ProductSerializer

    def get\_serializer\_context(self):

        return { 'request': self.request }

    def delete(self, request, pk: int):

        product = get\_object\_or\_404(Product, pk=pk)

        if product.orderitems.count() > 0:

            return Response({'error': 'product cannot be deleted'}, status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

        product.delete()

        return Response(status=status.HTTP\_204\_NO\_CONTENT)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

And To Register View Sets Inside The URLs File Of APP:

*Note*: Here We Can Use Include-Function With *router.urls*.

from rest\_framework.routers import SimpleRouter

from pprint import pprint

router = SimpleRouter()

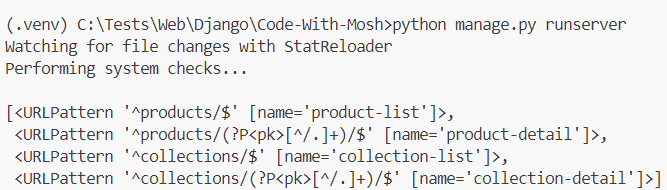
router.register('products', views.ProductViewSet)

router.register('collections', views.CollectionViewSet)

pprint(router.urls)

urlpatterns = router.urls

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

By Using DefaultRouter We Have Two Additional Features:

1. The Listing Of URLs Inside The Browser; Ex: <http://localhost:8000/store/>
2. If We Want Only The Data In JSON Format; Ex: http://localhost:8000/store/products.json

from rest\_framework.routers import DefaultRouter

from . import views

router = DefaultRouter()

router.register('products', views.ProductViewSet)

router.register('collections', views.CollectionViewSet)

urlpatterns = router.urls

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

When We Use View Set, And We Want To Use Delete Method, We Must Override The destroy-Method:

class CollectionViewSet(ModelViewSet):

    queryset = Collection.objects.annotate(products\_count=Count('products')).all()

    serializer\_class = CollectionSerializer

    def destroy(self, request, pk: int):

        collection = get\_object\_or\_404(

            Collection.objects.annotate(products\_count=Count('products')),

            pk=pk

        )

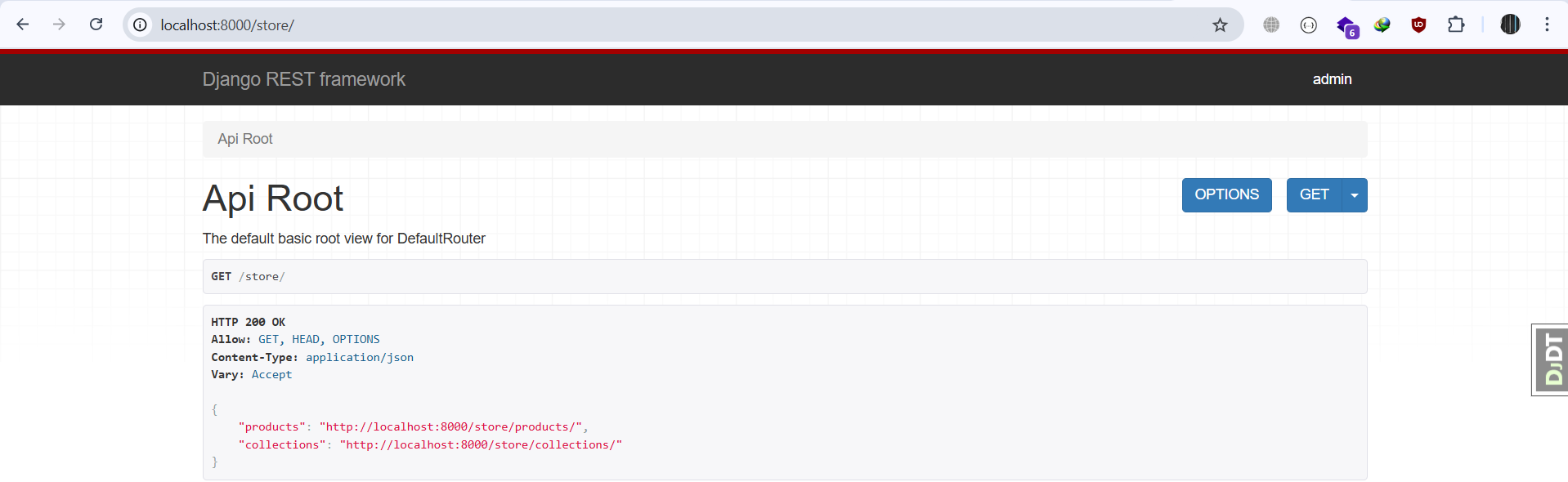
        if collection.products.count() > 0:

            return Response({'error': 'Collection Cannot Be Deleted'}, status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

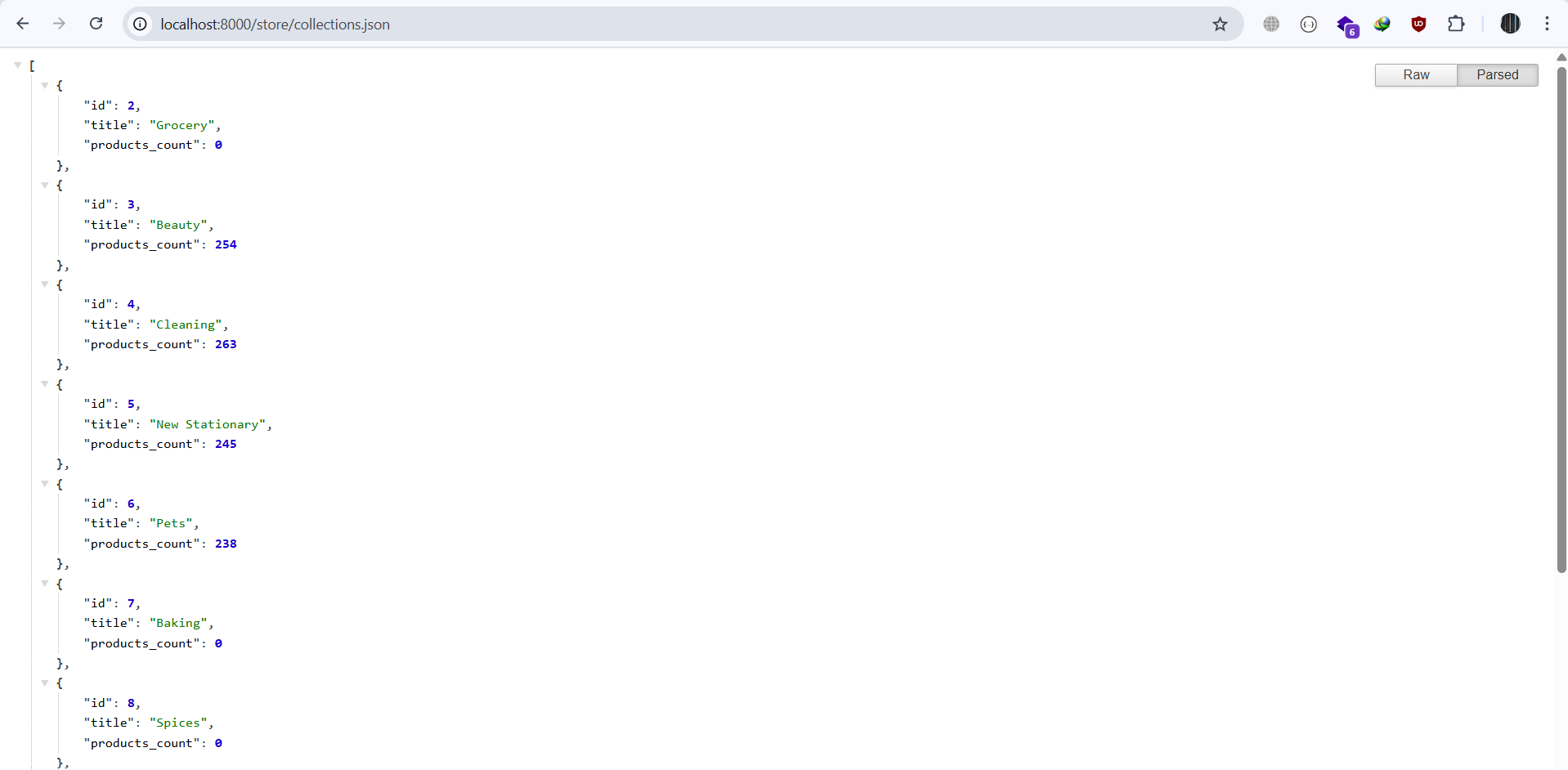
        collection.delete()

        return Response(status=status.HTTP\_204\_NO\_CONTENT)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If We Want To Implement Delete Method Using View Sets, We Must Override The Destroy Method:

class CollectionViewSet(ModelViewSet):

    queryset = Collection.objects.annotate(products\_count=Count('products')).all()

    serializer\_class = CollectionSerializer

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

        if Product.objects.filter(collection\_id=kwargs['pk']).count() > 0:

            return Response({'error': 'Collection Cannot Be Deleted'}, status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

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

--------------------------------------------------------------------------------------------

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.all()

    serializer\_class = ProductSerializer

    def get\_serializer\_context(self):

        return { 'request': self.request }

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

        if OrderItem.objects.filter(product\_id = kwargs['pk']).count() > 0:

            return Response({'error': 'product cannot be deleted'}, status=status.HTTP\_405\_METHOD\_NOT\_ALLOWED)

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*