***Note 1***: With ReadOnlyModelViewSet We Can Only Use list, And Retrieve By Field.

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

For creating Nested Routers, We Use: ***pip install drf-nested-routers***

Then We Import It, And Use Its Routers-Classes:

from rest\_framework\_nested import routers

router = routers.DefaultRouter()

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

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

products\_router = routers.NestedDefaultRouter(router, 'products', lookup='product')

products\_router.register('reviews', views.ReviewViewSet, basename='product-reviews')

urlpatterns = router.urls + products\_router.urls

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

To Avoid Problems of Parent Pk Not Used For Child **OR** Related Objects:

**Note**: Here We Use Context to Avoid Read the Parent Data from User Input, Then We Override The ***create-Method*** Of ReviewSerializer

class ReviewViewSet(ModelViewSet):

    # queryset = Review.objects.all()

    serializer\_class = ReviewSerializer

    def get\_queryset(self):

        return Review.objects.filter(product\_id = self.kwargs['product\_pk'])

    def get\_serializer\_context(self):

        return { 'product\_id': self.kwargs['product\_pk'] }

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

To Implement Filtering Using Query Params, We Must:

* Override The *get\_queryset-Method*
* Use *get-Method of Dict*
* Set ***basename in urls.py*** For Target Class

class ProductViewSet(ModelViewSet):

    serializer\_class = ProductSerializer

    def get\_queryset(self):

        queryset = Product.objects.all()

        collection\_id = self.request.query\_params.get('collection\_id')

        if collection\_id is not None:

            queryset = queryset.filter(collection\_id=collection\_id)

        return queryset

router = routers.DefaultRouter()

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

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

To Use Generic Filtering In Django: We Use: ***pip install django-filter***

Then We Register It in installed APPs, AS: ***django\_filters*** And Before The ***rest\_framework***

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

Then To Implement The Filters For Products:

***Note***: In This Way We Can Return ***queryset-Attribute***

from django\_filters.rest\_framework import DjangoFilterBackend

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.all()

    serializer\_class = ProductSerializer

    filter\_backends = [DjangoFilterBackend]

    filterset\_fields = ['collection\_id']

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

***Note***: We Must Be Careful When Using django-filter, Because We May Have Duplicate Query

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

To Implement Custom Filter Using django-filter:

from django\_filters import FilterSet

from .models import Product

class ProductFilter(FilterSet):

    class Meta:

        model = Product

        fields = {

            'collection\_id': ['exact'],

            'unit\_price': ['lt', 'gt'],

        }

Then In views.py:

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.all()

    serializer\_class = ProductSerializer

    filter\_backends = [DjangoFilterBackend]

    filterset\_class = ProductFilter

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

To Implement Search Filters, We Use Filters of *rest\_framework*:

from rest\_framework.filters import SearchFilter

Then We Add it to Filters Array:

filter\_backends = [DjangoFilterBackend, SearchFilter]

Then We Define The Search Array Fields:

search\_fields = ['title', 'description']

**Note 1**: The Search Is *Case Insensitive*

**Note 2**: This Will Use search query Param: *http://localhost:8000/store/products/?search=coffee*

The Complete Implementation:

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.all()

    serializer\_class = ProductSerializer

    filter\_backends = [DjangoFilterBackend, SearchFilter]

    filterset\_class = ProductFilter

    search\_fields = ['title', 'description']

    def get\_serializer\_context(self):

        return { 'request': self.request }

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

To implement ordering using specific fields:

from rest\_framework.filters import SearchFilter, OrderingFilter

Then We Add It To Filters Backend Array:

filter\_backends = [DjangoFilterBackend, SearchFilter, OrderingFilter]

Then We Define The Ordering Fields: ordering\_fields = ['unit\_price', 'last\_update']

The Complete Implementation:

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.all()

    serializer\_class = ProductSerializer

    filter\_backends = [DjangoFilterBackend, SearchFilter, OrderingFilter]

    filterset\_class = ProductFilter

    search\_fields = ['title', 'description']

    ordering\_fields = ['unit\_price', 'last\_update']

    def get\_serializer\_context(self):

        return { 'request': self.request }

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

**Note**: To Order Using Multiple Fields: *http://localhost:8000/store/products/?ordering=-unit\_price,last\_update&search=*

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

To implement the pagination in rest\_framework:

from rest\_framework.pagination import PageNumberPagination

Then We Define The Pagination Class Inside The ViewSet:

pagination\_class = PageNumberPagination

Then to define the Page Size, We Go To settings.py Of Main Project:

REST\_FRAMEWORK = {

    'COERCE\_DECIMAL\_TO\_STRING': False,

    'PAGE\_SIZE': 10,

    'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.PageNumberPagination'

}

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.all()

    serializer\_class = ProductSerializer

    filter\_backends = [DjangoFilterBackend, SearchFilter, OrderingFilter]

    filterset\_class = ProductFilter

    search\_fields = ['title', 'description']

    ordering\_fields = ['unit\_price', 'last\_update']

    pagination\_class = PageNumberPagination # This Can Be Deleted

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

By Using The Default Pagination Class, We Can Delete The ***pagination\_class*** From Product ViewSet

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

**Note**: If We Want to Use Limit and Offset for Pagination, We Have *LimitOffsetPagination-Class*.

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

The Best Way, If We Don’t Want To use Pagination for all ViewSets, Is By Implementing Our Paginator Class And Define Its Settings:

from rest\_framework.pagination import PageNumberPagination

class DefaultPagination(PageNumberPagination):

    page\_size = 10

Then We Set The ***pagination\_class*** Only For ViewSets, That Needed It:

from .pagination import DefaultPagination

class ProductViewSet(ModelViewSet):

    queryset = Product.objects.all()

    serializer\_class = ProductSerializer

    filter\_backends = [DjangoFilterBackend, SearchFilter, OrderingFilter]

    filterset\_class = ProductFilter

    search\_fields = ['title', 'description']

    ordering\_fields = ['unit\_price', 'last\_update']

    pagination\_class = DefaultPagination

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

from uuid import uuid4

To use UUID AS ***Id Field***: id = models.UUIDField(primary\_key=True, default=uuid4)

*Note*: Here We Don’t Call ***uuid4-Function***

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

If Any Problems Happened When Changing Id Field From Bigint-8 To UUID: (From Stackoverflow)

So, to handle this in django, do the following:

**1)** revert migrations to a working graph

**2)** add **temp\_id = models.UUIDField(default=uuid.uuid4)** to your model, then run **makemigrations**

**3)** \* add **primary\_key=True** to the **temp\_id** field, then run **makemigrations** again

**4)** rename the field to **id** (or to whatever you want), then run **makemigrations** a third time

**5)** push the migrations to the database via **python3 manage.py migrate**

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

Note (To Remember):

from rest\_framework.mixins import CreateModelMixin, RetrieveModelMixin

from rest\_framework.viewsets import ModelViewSet, GenericViewSet

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

Note: To create simple serializer that can be used instead of full one:

class SimpleProductSerialzier(serializers.ModelSerializer):

    class Meta:

        model = Product

        fields = [ 'id', 'title', 'unit\_price' ]

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

class CartItemSerializer(serializers.ModelSerializer):

    product = SimpleProductSerialzier()

    total\_price = serializers.SerializerMethodField(method\_name='get\_total\_price')

    def get\_total\_price(self, cart\_item: CartItem):

        return cart\_item.quantity \* cart\_item.product.unit\_price

    class Meta:

        model = CartItem

        fields= ['id', 'product', 'quantity', 'total\_price']

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

class CartSerializer(serializers.ModelSerializer):

    id = serializers.UUIDField(read\_only=True)

    items = CartItemSerializer(many=True, read\_only = True)

    total\_price = serializers.SerializerMethodField(method\_name='get\_total\_price')

    def get\_total\_price(self, cart: Cart):

        return sum([item.quantity \* item.product.unit\_price for item in cart.items.all()])

    class Meta:

        model = Cart

        # Here We Define items Inside The CartItem-Model AS The Related Name

        fields = [ 'id', 'items', 'total\_price']

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

*Note (To remember):* serializer objects has method for validation: **is\_valid(…)**

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

class AddCartItemSerialzier(serializers.ModelSerializer):

    id = serializers.UUIDField(read\_only=True)

    product\_id = serializers.IntegerField() # Here we set product\_id

                                       # because it is populated only in runtime

    # here will call self.is\_valid

    def save(self, \*\*kwargs):

        cart\_id = self.context['cart\_id']

        product\_id = self.validated\_data['product\_id']

        quantity = self.validated\_data['quantity']

        try:

            cart\_item = CartItem.objects.get(cart\_id = cart\_id, product\_id = product\_id)

            cart\_item.quantity += quantity

            cart\_item.save()

            self.instance = cart\_item

        except CartItem.DoesNotExist:

            self.instance = CartItem.objects.create(cart\_id=cart\_id, \*\*self.validated\_data)

        return self.instance

    class Meta:

        model = CartItem

        fields = ['id', 'product\_id', 'quantity']

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