To Install Django Rest Framework: *pip install djangorestframework*

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

Then We Need To Add It To INSTALLED\_APPS Like: 'rest\_framework'

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

To Use Rest Framework To Handle Request And Make Response:

Note 1: The api\_view-decorator Make The Request From Rest Framework Not Django, And It is More Powerful.

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

# Create your views here.

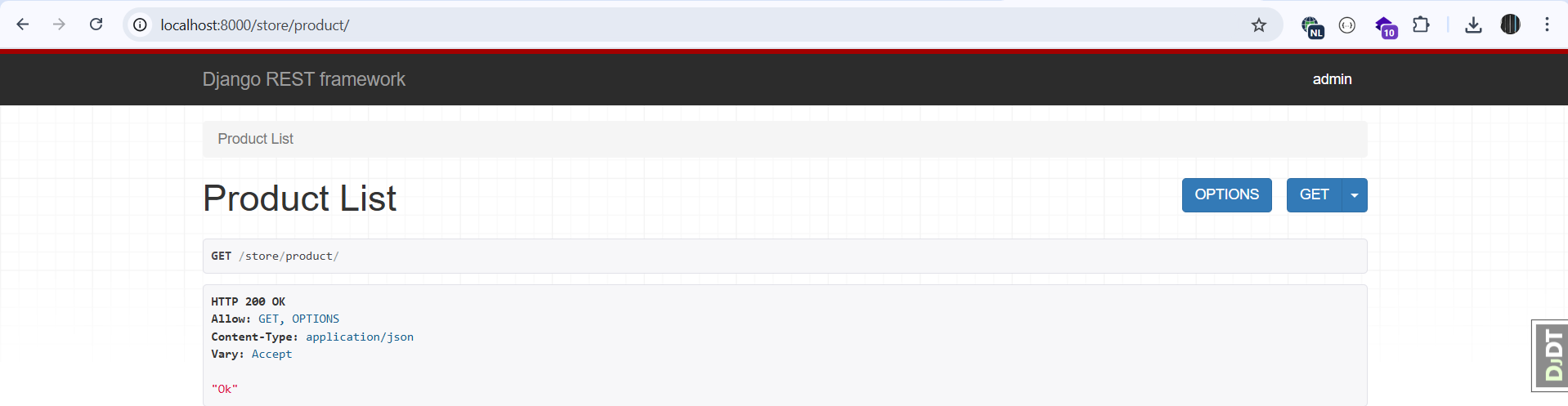
@api\_view()

def product\_list(request):

    return Response('Ok')

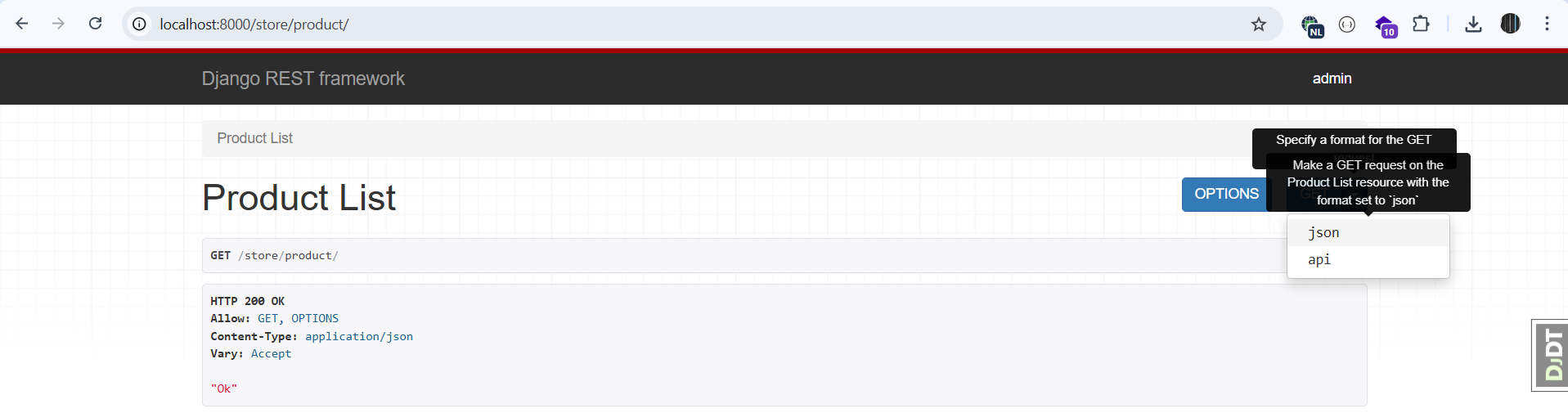
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The Output Of Using api\_view From Browser:



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

**Note**: This Window Will Show Only If We See The Output From Browser, If We Select Json, Then Will Show The API Output.



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

To Make APIs With URL Params Like Id:

In urls.py Of Main APP:

Note: In This Way Only Numeric Id Will Accepted

urlpatterns = [

    path('product/', views.product\_list),

    path('product/<int:id>/', views.product\_detail),

];

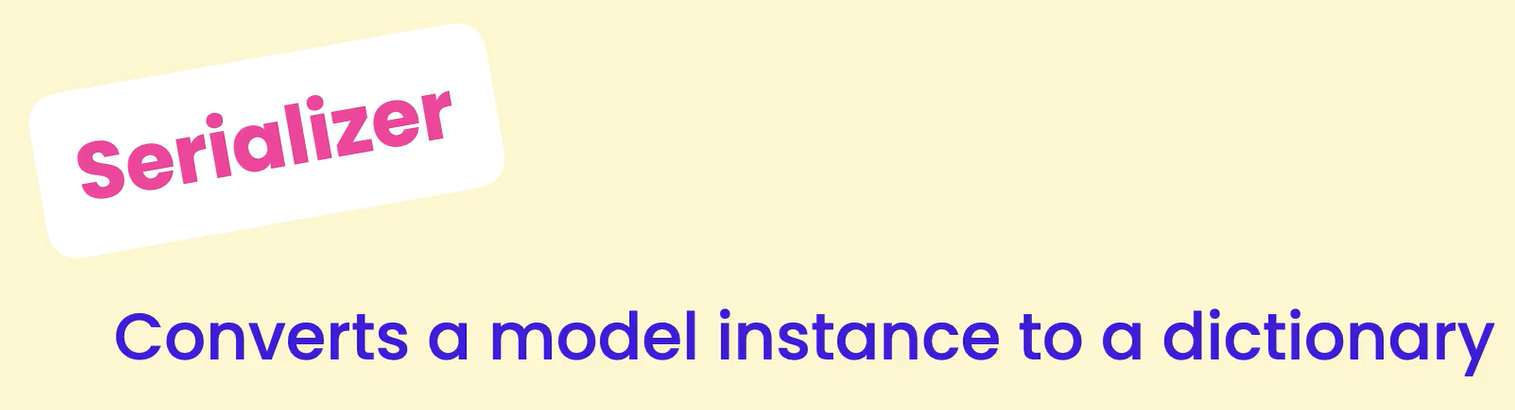
In Store APP:

@api\_view()

def product\_detail(request, id):

    return Response(id)

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



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

To Define Serializer, We Add serializers.py To APP Folder.

Then We Define:

from rest\_framework import serializers

class ProductSerializer(serializers.Serializer):

    id = serializers.IntegerField()

    title = serializers.CharField(max\_length=255)

    unit\_price = serializers.DecimalField(max\_digits=6, decimal\_places=2)

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

**Note**: We Define The Validation Using Serializer, So When We Sent The Data Using API, Then We Can Validate It Before Any Operation, Like: Save.

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

Then To Use Serializer With Model:

@api\_view()

def product\_detail(request, id):

    product = Product.objects.get(pk=id)

    serializer = ProductSerializer(product)

    return Response(serializer.data)

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

To Override The Django Repr Of Decimal When Return Data, Inside The settings.py Of Main App:

REST\_FRAMEWORK = {

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

}

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

from rest\_framework import status

@api\_view()

def product\_detail(request, id):

    try:

        product = Product.objects.get(pk=id)

        serializer = ProductSerializer(product)

        return Response(serializer.data)

    except Product.DoesNotExist:

        return Response(status=status.HTTP\_404\_NOT\_FOUND)

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

from django.shortcuts import get\_object\_or\_404

@api\_view()

def product\_detail(request, id):

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

    serializer = ProductSerializer(product)

    return Response(serializer.data)

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

To Get List Of Data:

@api\_view()

def product\_list(request):

    queryset = Product.objects.all()[:10]

    serializer = ProductSerializer(queryset, many=True)

    return Response(serializer.data)

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

To Define Custom Parameter For Our API We can Use *serializers.SerializerMethodField*:

**Note 1**: Here We Define Custom API Parameter Called price, And Set The Source Value For Product Model Is: *unit\_price*

from rest\_framework import serializers

from decimal import Decimal

from .models import Product

class ProductSerializer(serializers.Serializer):

    id = serializers.IntegerField()

    title = serializers.CharField(max\_length=255)

    price = serializers.DecimalField(max\_digits=6, decimal\_places=2, source='unit\_price')

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

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

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

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

In This way We Handle The Relationships Between Models:

from rest\_framework import serializers

from decimal import Decimal

from .models import Product, Collection

class ProductSerializer(serializers.Serializer):

    id = serializers.IntegerField()

    title = serializers.CharField(max\_length=255)

    price = serializers.DecimalField(max\_digits=6, decimal\_places=2, source='unit\_price')

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

    collection = serializers.PrimaryKeyRelatedField(queryset = Collection.objects.all())

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

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

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

To Include The String Representation For Related Models:

**Note**: In This Way We Must Use *select\_related-Method*, Else The Lazy Loading Will Make Above 1000-Query

class ProductSerializer(serializers.Serializer):

    id = serializers.IntegerField()

    title = serializers.CharField(max\_length=255)

    price = serializers.DecimalField(max\_digits=6, decimal\_places=2, source='unit\_price')

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

    collection = serializers.StringRelatedField()

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

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

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

@api\_view()

def product\_list(request):

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

    serializer = ProductSerializer(queryset, many=True)

    return Response(serializer.data)

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

To Create Nested Objects Inside The Main Serializer, First; We Make New Serializer For Nested Object, Then We Create The Nested Object Inside The Main Serializer.

class CollectionSerializer(serializers.Serializer):

    id = serializers.IntegerField()

    title = serializers.CharField(max\_length= 255)

class ProductSerializer(serializers.Serializer):

    id = serializers.IntegerField()

    title = serializers.CharField(max\_length=255)

    price = serializers.DecimalField(max\_digits=6, decimal\_places=2, source='unit\_price')

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

    collection = CollectionSerializer()

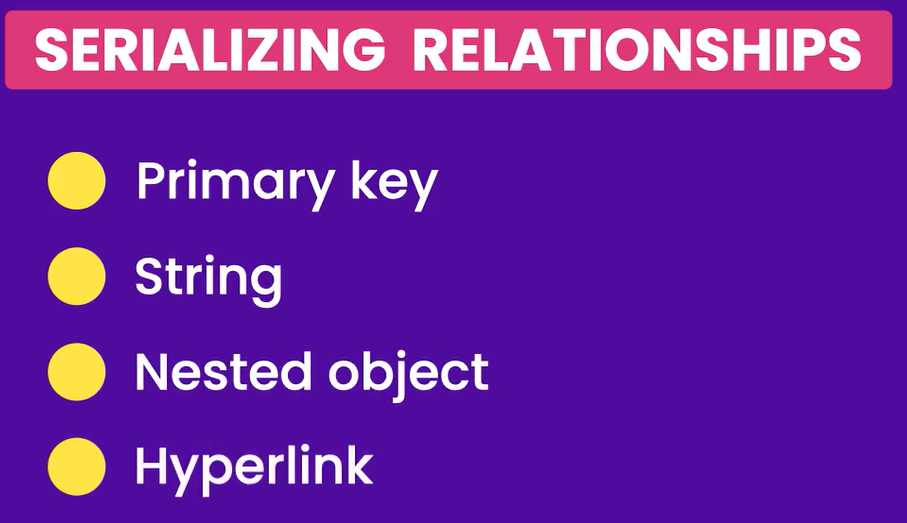
def calculate\_tax(self, product: Product):

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

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

**Note**: For The Previous State We Use, Also; select\_related

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



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

To Make Hyperlink To Another Related Objects, We Must Change These Files Inside The APP: *urls.py*, *views.py*, *serializer.py*

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

class ProductSerializer(serializers.Serializer):

    id = serializers.IntegerField()

    title = serializers.CharField(max\_length=255)

    price = serializers.DecimalField(max\_digits=6, decimal\_places=2, source='unit\_price')

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

    collection = serializers.HyperlinkedRelatedField(

        queryset = Collection.objects.all(),

        view\_name = 'collection-detail',

    )

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

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

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

***Note***: Here We Must Set The Lookup Field As: ***pk***, Not ***id***

urlpatterns = [

    path('product/', views.product\_list),

    path('product/<int:id>/', views.product\_detail),

    path('collection/<int:pk>/', views.collection\_detail, name='collection-detail')

];

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

@api\_view()

def collection\_detail(request, pk:int):

    return Response('Ok')

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

To Create Model Serializer, For Avoiding Duplicate Codes, Ex: For Validation Rules:

class ProductSerializer(serializers.ModelSerializer):

    class Meta:

        model = Product

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

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

To Change The Fields Name:

**Note**: Also We Can Include The Hyper Link Related Field Also.

class ProductSerializer(serializers.ModelSerializer):

    class Meta:

        model = Product

        fields = ['id', 'title', 'price', 'collection']

    price = serializers.DecimalField(max\_digits=6, decimal\_places=2, source='unit\_price')

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

class ProductSerializer(serializers.ModelSerializer):

    class Meta:

        model = Product

        fields = ['id', 'title', '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)

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

To Create View That Accepts GET, POST:

@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)

        return Response('Ok')

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

To Deserialize The Data From Request, We Can Use data-Attribute Of SerializerObject.

To Validate The Data We Can Use serializer.errors OR serializer.is\_valid(raise\_exception=True)

@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)

        print(serializer.validated\_data)

        return Response('Ok')

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