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.

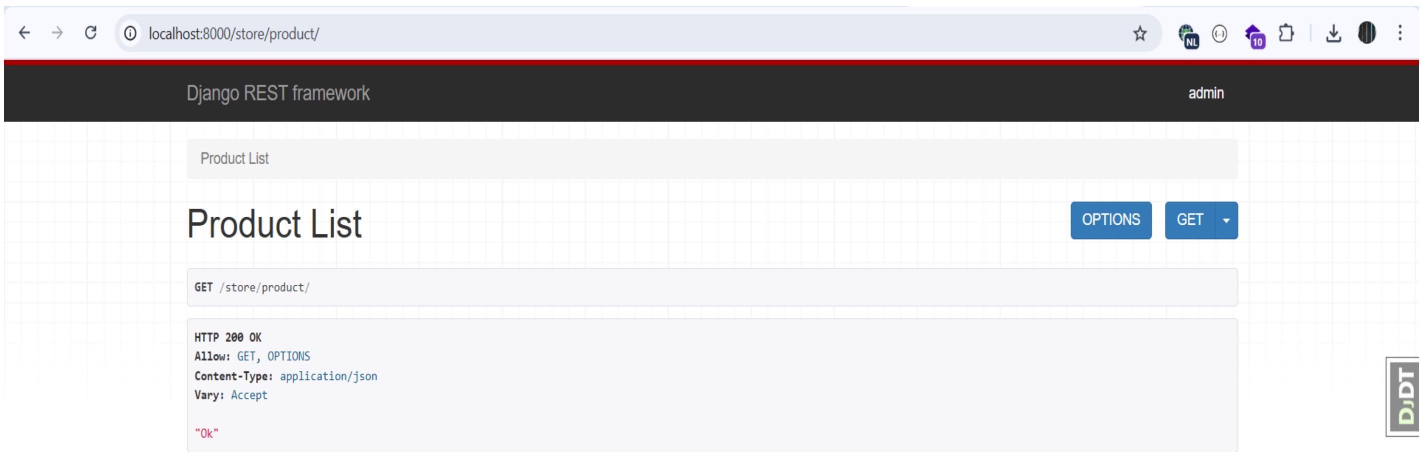
```
from rest_framework.decorators import api_view

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:



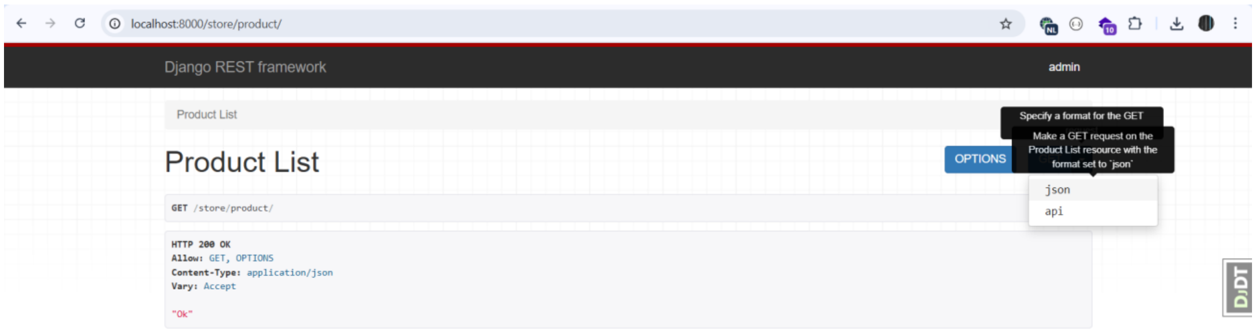
```
GET /store/product/
```

```
HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

"Ok"
```

****

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


localhost:8000/store/product/
Django REST framework
admin
Product List
OPTIONS
Specify a format for the GET
Make a GET request on the Product List resource with the format set to `json`
json
api

Product List

```
GET /store/product/

HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

"Ok"
```

****

To Make APIs With URL Params Like Id:

In urls.py Of Main APP:

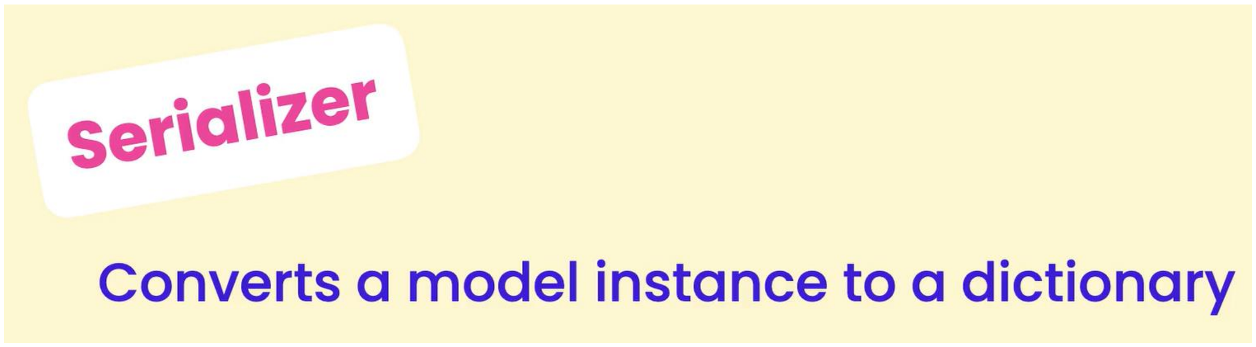
Note: In This Way Only Numeric Id Will Accepted

```
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:

```python
REST_FRAMEWORK = {
    'COERCE_DECIMAL_TO_STRING': False,
}
```

****

```python
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)
```

****

```python
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:

```python
@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

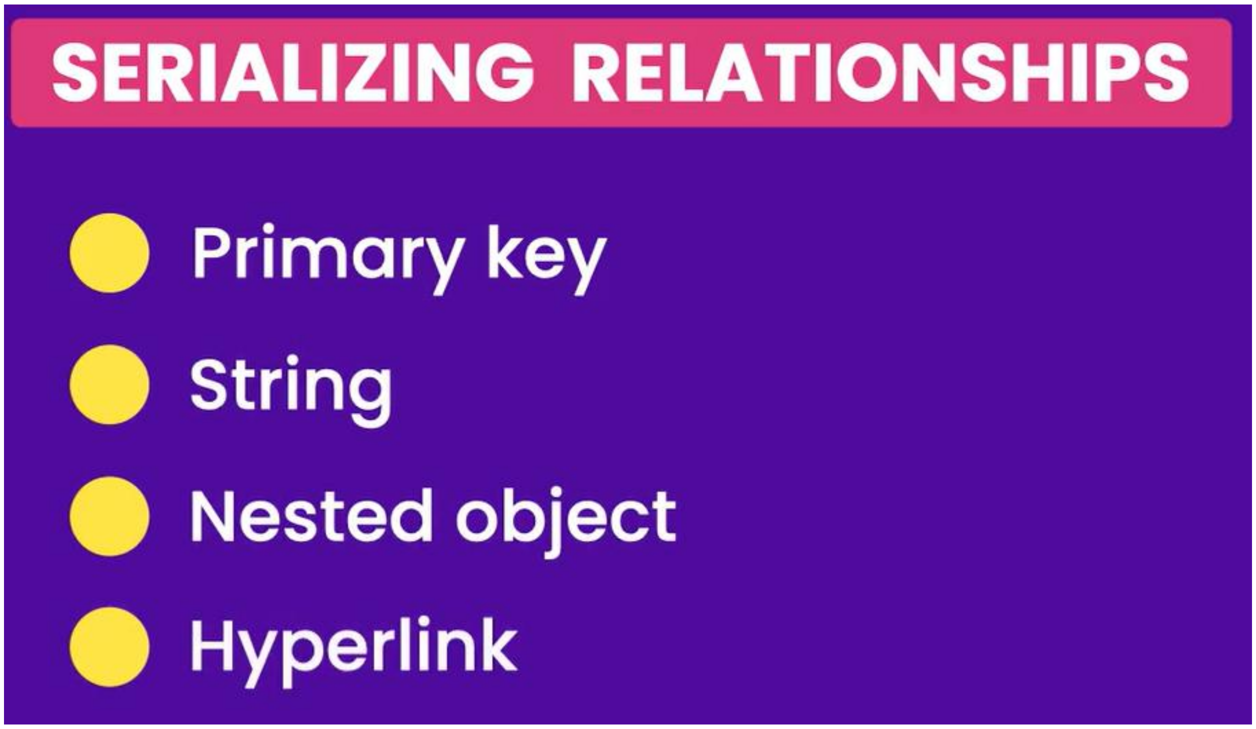
We Create The Nested Object Inside The Main Serializer.

```
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

****



SERIALIZING RELATIONSHIPS

- Primary key
- String
- Nested object
- Hyperlink

****

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')
```
****