ModelSerializer in DRF

ModelSerializer is a shortcut class provided by Django REST Framework that automatically creates a serializer based on a Django model.

It saves you from manually declaring fields, validators, and create/update methods.

Benefits:

- Automatically maps model fields
- Handles validation and object creation/update
- Saves boilerplate code
- Integrated with Django ORM

Example : simple Book API using Django REST Framework and ModelSerializer

Step 1 : Setup project

```
<!-- Install Required Packages -->
pip install django djangorestframework

<!-- Create a Django Project -->
django-admin startproject myapi

cd myapi

<!-- Create a Django App -->
python manage.py startapp books

<!-- Add rest_framework and books to INSTALLED_APPS in myapi/settings.py: -->
INSTALLED_APPS = [
    ...
    'rest_framework',
    'books',
]
```

Step 2 : Create a Model

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)

    def __str__(self):
        return self.title
```

Step 3 : Run the migration

```
python manage.py makemigrations
python manage.py migrate
```

Step 4 : Create a Serializer using ModelSerializer

```
from rest_framework import serializers
from .models import Author, Book

class AuthorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Author
        fields = ['id', 'name']

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = ['id', 'title', 'author']
```

Step 5 : Create API Views

```
from rest_framework import viewsets
from .models import Author, Book
from .serializers import AuthorSerializer, BookSerializer

class AuthorViewSet(viewsets.ModelViewSet):
    queryset = Author.objects.all()
    serializer_class = AuthorSerializer

class BookViewSet(viewsets.ModelViewSet):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

Step 6 : Set Up URLs

```
from django.contrib import admin
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from books.views import AuthorViewSet, BookViewSet

router = DefaultRouter()
router.register(r'authors', AuthorViewSet)
router.register(r'books', BookViewSet)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include(router.urls)),
]
```

Step 7 : Run the Server

```
python manage.py runserver

# http://127.0.0.1:8000/api/authors/ → Author list/create

# http://127.0.0.1:8000/api/books/ → Book list/create
```

HyperlinkedModelSerializer in DRF

HyperlinkedModelSerializer is a DRF serializer that represents relationships using URLs instead of primary keys. It's a more RESTful and navigable alternative to ModelSerializer.

In a REST API, one common pattern is to link related resources using URLs rather than showing only IDs. This lets clients easily follow relationships.

```
<!-- For example, instead of: -->
{
  "id": 1,
  "author": 2
}

<!-- You get: -->
{
  "url": "http://api.example.com/books/1/",
  "author": "http://api.example.com/authors/2/"
}
```

Example

```
from rest_framework import serializers
from .models import Book

class BookSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Book
        fields = ['url', 'title', 'author']
```

Run the application and perform the CRUD opertaion