Django REST Framework (Class-Based View)

APIView in DRF

APIView is the base class for class-based views in DRF. It gives you more control and flexibility compared to function-based @api_view.

- A class you extend to create your API endpoints as classes.
- It handles HTTP methods via class methods like get(), post(), put(), delete().
- It provides features like authentication, permissions, throttling, content negotiation.

serializers.Serializer

DRF Serializers convert complex data like Django models into JSON (or other formats) and validate input data.

Two main types:

- serializers.Serializer: Manual, flexible serializer.
- serializers.ModelSerializer: Auto-generates fields based on model.

Step 1 : Create Django Project and install djago

```
django-admin startproject bookStore
cd myproject
pip install django
```

step 2: Create Django App

```
python manage.py startapp books
```

Step 3 : install DRF

```
pip install djangorestframework
```

Step 4 : Add settings.py

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'books',  # your app name
]
```

Step 5 : Create a model inside app(book)

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=100)             # Book title
    author = models.CharField(max_length=100)            # Author name
    published_date = models.DateField()                  # Date of publication

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

Step 6 : Make Migrations & Migrate

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

Step 7 : Create Serializer

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

class BookSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    title = serializers.CharField()
    author = serializers.CharField()
    published_date = serializers.DateField()

    #create method
    def create(self, validated_data):
        return Book.objects.create(**validated_data)

    #update method
    def update(self, instance, validated_data):
        self.title = validated_data.get('title', instance.title)
        self.author = validated_data.get('author', instance.author)
        self.published_date = validated_data.get('published_date',
instance.published_date)
        instance.save()
        return instance
```

Step 8 : create ApiView

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from django.shortcuts import get_object_or_404
from .models import Book
from .serializers import BookSerializer

# List all books or create a new one
class BookListCreateView(APIView):
    def get(self, request):
        books = Book.objects.all()                             # Fetch all books
from DB
        serializer = BookSerializer(books, many=True)          # Serialize list of
books
        return Response(serializer.data)                       # Return serialized
data as JSON

    def post(self, request):
        serializer = BookSerializer(data=request.data)         # Deserialize and
validate incoming data
        if serializer.is_valid():                              # Check if data is
valid
            serializer.save()                                  # Save valid book
to DB
            return Response(serializer.data, status=status.HTTP_201_CREATED)  #
Return created book
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)  #
Return validation errors

# Retrieve, update or delete a single book
class BookDetailView(APIView):
    def get_object(self, pk):
        return get_object_or_404(Book, pk=pk)                  # Helper method to
get object or return 404

    def get(self, request, pk):
        book = self.get_object(pk)                             # Fetch specific
book
        serializer = BookSerializer(book)                      # Serialize the
book
        return Response(serializer.data)                       # Return book data

    def put(self, request, pk):
        book = self.get_object(pk)                             # Fetch book to
update
        serializer = BookSerializer(book, data=request.data)   # Deserialize and
validate new data
        if serializer.is_valid():
            serializer.save()                                  # Save updated book
            return Response(serializer.data)                   # Return updated
data
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)  #
Return validation errors

    def delete(self, request, pk):
        book = self.get_object(pk)                             # Fetch book to
delete
```

```
        book.delete()                                         # Delete from DB
        return Response(status=status.HTTP_204_NO_CONTENT)     # Return empty
response with 204 status
```

Step 9 : Register URLs (books/urls.py)

```
from django.urls import path
from .views import BookListCreateView, BookDetailView

urlpatterns = [
    path('books/', BookListCreateView.as_view()),
    path('books/<int:pk>/', BookDetailView.as_view()),
]
```

Step 10 : Include app into project(project/urls.py)

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('books.urls')),  # Prefix API routes with /api/
]
```

Step 11 : Run Development Server

```
python manage.py runserver
```