🚀 DJANGO REST FRAMEWORK (MODELVIEWSET)

Complete Step-by-Step Guide (With Intro + Code)

🟢 STEP 0: Environment Setup

📘 Introduction

We first create an isolated Python environment and install required frameworks to avoid dependency conflicts.

🔧 Commands

```
python -m venv venv

source venv/bin/activate     # Windows: venv\Scripts\activate

pip install django djangorestframework
```

🟢 STEP 1: Create Django Project

📘 Introduction

A Django project is the **root container** of your backend system.
It holds settings, URLs, and app registrations.

🔧 Commands

```
django-admin startproject core

cd core
```

🟢 STEP 2: Create Django App

📘 Introduction

A Django app handles **one feature** of your system.
Here, users manages user-related logic.

🔧 Command

```
python manage.py startapp users
```

🟢 STEP 3: Register Apps in Settings

📘 Introduction

Django only recognizes apps that are registered in INSTALLED_APPS.

Code (core/settings.py)

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'rest_framework',   # DRF
    'users',            # User app
]
```

STEP 4: Create Model

Introduction

Models define **database tables** using Python classes.

Code (users/models.py)

```
from django.db import models


class User(models.Model):
    ROLE_CHOICES = (
        ('admin', 'Admin'),
        ('instructor', 'Instructor'),
        ('student', 'Student'),
    )

    name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=255)
    role = models.CharField(max_length=20, choices=ROLE_CHOICES)

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

🟢 STEP 5: Create Migrations

📘 Introduction

Migrations generate **instructions** to create or modify database tables.

🔧 Command

```
python manage.py makemigrations
```

🟢 STEP 6: Apply Migrations

📘 Introduction

This step executes the migration instructions and creates tables in the database.

🔧 Command

```
python manage.py migrate
```

🟢 STEP 7: Create Serializer

📘 Introduction

Serializers validate data and safely convert between:

- JSON ↔ Python ↔ Database

🧩 Code (users/serializers.py)

```
from rest_framework import serializers
from .models import User
from django.contrib.auth.hashers import make_password

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'name', 'email', 'password', 'role']
        extra_kwargs = {
            'password': {'write_only': True}
        }

    def create(self, validated_data):
        validated_data['password'] = make_password(
            validated_data['password']
        )
        return super().create(validated_data)
```

🟢 STEP 8: Create ViewSet (MAIN PART)

📘 Introduction

A ViewSet handles **request logic**.
ModelViewSet automatically provides full CRUD functionality.

🧩 Code (users/views.py)

```
from rest_framework.viewsets import ModelViewSet
from .models import User
from .serializers import UserSerializer


class UserViewSet(ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

🟢 STEP 9: Create Router

📘 Introduction

A Router automatically generates URL endpoints for ViewSets.

🧩 Code (users/urls.py)

```
from rest_framework.routers import DefaultRouter
from .views import UserViewSet

router = DefaultRouter()
router.register('users', UserViewSet)

urlpatterns = router.urls
```

🟢 STEP 10: Connect App URLs to Project

📘 Introduction

This step exposes your app APIs to the project-level routing system.

🧩 Code (core/urls.py)

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

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

🟢 STEP 11: Run Server

📘 Introduction

Starts Django’s development server to test APIs.

🔧 Command

```
python manage.py runserver
```

🟢 STEP 12: Test API Endpoints

📘 Introduction

Testing ensures CRUD operations work correctly.

🌐 Available APIs

POST	/api/users/	→ Create user
GET	/api/users/	→ List users
GET	/api/users/1/	→ Retrieve user
PUT	/api/users/1/	→ Update user
DELETE	/api/users/1/	→ Delete user

🧠 ONE-LINE MASTER FLOW (MEMORIZE THIS)

Model → Migration → Serializer → ViewSet → Router → URL → API

⭐ WHY MODELVIEWSET IS USED IN INDUSTRY

- Less boilerplate code
- Fewer bugs
- Faster development
- Cleaner architecture
- Easy maintenance

🧠 FINAL PATH COMPARISON (IMPORTANT)

Feature	**APIView**	**ModelViewSet**
URL definition	path()	router.register()
Method mapping	Manual	Automatic
Best for	Custom APIs	CRUD APIs

🧩 PART 1 — ADDING APIView (WITH PATH & CODE)

So far, we focused on **ModelViewSet** (recommended).
Now let's **also include APIView**, so you clearly understand:

- How URLs differ
- When APIView is used
- Minimal working example

🔵 WHAT IS APIView? (SHORT INTRO)

📘 Introduction

APIView is the **base-level class** in DRF.

- You manually define:
  - get()
  - post()
  - put()
  - delete()
- You manually map URLs to methods

👉 **More control, more code**

🔁 Comparison (Quick Recap)

Feature	APIView	ModelViewSet
CRUD auto	❌	✅
URL auto	❌	✅
Code length	Long	Short
Best for	Custom logic	CRUD APIs

🟢 WHEN SHOULD YOU USE APIView?

Use APIView for:

- Login

- Logout
- Custom actions
- Reports
- Analytics
- OTP verification

🟢 APIView EXAMPLE (USER CREATE + LIST)

📁 File: users/views.py

```
from rest_framework.views import APIView

from rest_framework.response import Response

from rest_framework import status

from .models import User

from .serializers import UserSerializer
```

📘 APIView Code

```
class UserAPIView(APIView):
    """
    Handles:
    - GET  → list users
    - POST → create user
    """


    def get(self, request):
        users = User.objects.all()
        serializer = UserSerializer(users, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)


    def post(self, request):
        serializer = UserSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(
                serializer.data,
                status=status.HTTP_201_CREATED
            )

        return Response(
            serializer.errors,
            status=status.HTTP_400_BAD_REQUEST
        )
```

🧠 **WHAT IS HAPPENING HERE (VERY SIMPLE)**

Request

↓

APIView

↓

Manual get/post

↓

Serializer

↓

Model

↓

Response

You write **everything yourself**.

🟢 **APIView PATH (VERY IMPORTANT)**

📁 **File: users/urls.py**

```
from django.urls import path

from .views import UserAPIView

urlpatterns = [

    path('users-api/', UserAPIView.as_view(), name='users-api'),

]
```

🌐 APIView Endpoint

```
GET  /api/users-api/

POST /api/users-api/
```

🔁 HOW APIView CONNECTS (MENTAL MODEL)

path()

 ↓

APIView.as_view()

 ↓

HTTP method

 ↓

get() / post()