Django REST Framework (DRF) Serializers - Beginner Friendly Guide

1. What is a Serializer?

A **serializer** in Django REST Framework is a tool used to:

- Convert **Django model instances (Python objects)** to **JSON** (for API responses)
- Convert **JSON data from API requests** to **Python objects** (for validation and saving)

It's like a **translator** between your backend (Python/Django) and frontend (JSON/API).

---

2. Why Do We Need Serializers?

Without serializers, you must manually:

- Convert model objects to JSON
- Parse and validate incoming JSON data
- Handle field validation and error responses

Serializers automate all this and make your code cleaner, safer, and faster to develop.

---

3. Types of Serializers

a. `ModelSerializer`

- Automatically maps Django model fields
- Easiest and most commonly used

b. `Serializer`

- Manually define each field
- Used when you need more control or aren't using a model

---

4. Example: Using ModelSerializer

models.py

```
from django.db import models
```

```
class EMPLOYEE(models.Model):
    emp_name = models.CharField(max_length=100)
    emp_email = models.EmailField()
    emp_mob = models.CharField(max_length=15)
```

serializers.py

```
from rest_framework import serializers
from .models import EMPLOYEE

class EmployeeSerializer(serializers.ModelSerializer):
    class Meta:
        model = EMPLOYEE
        fields = ['id', 'emp_name', 'emp_email', 'emp_mob']
```

- `EmployeeSerializer`: defines how the `EMPLOYEE` model should be converted to/from JSON.
- `Meta`: a nested class where you define the model to use and which fields to include.

---

5. GET Example: Convert Model to JSON

views.py

```
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .models import EMPLOYEE
from .serializers import EmployeeSerializer

@api_view(['GET'])
def get_employee(request, id):
    try:
        employee = EMPLOYEE.objects.get(id=id)
        serializer = EmployeeSerializer(employee)  # Converts model instance to
JSON
        return Response({"msg": "user found", "data": serializer.data})  #
serializer.data gives JSON
    except EMPLOYEE.DoesNotExist:
        return Response({"msg": "user not found"}, status=404)
```

- `api_view`: DRF decorator to define allowed HTTP methods.
- `serializer = EmployeeSerializer(employee)`: serialize the object.
- `serializer.data`: get serialized JSON.

---

6. POST Example: Accept JSON and Save

views.py

```
@api_view(['POST'])
def create_employee(request):
    serializer = EmployeeSerializer(data=request.data)  # Deserialize JSON to
Python
    if serializer.is_valid():  # Validate incoming data
        serializer.save()  # Save to database
        return Response({"msg": "employee created", "data": serializer.data},
status=201)
    return Response(serializer.errors, status=400)  # Return validation errors
```

- `data=request.data`: input data from request.
- `is_valid()`: checks required fields and types.
- `save()`: creates new model instance.
- `serializer.errors`: shows validation errors if any.

---

7. Benefits of Using Serializers

Feature	Without Serializer	With Serializer
Convert Model to JSON	Manually	`.data` built-in
Convert JSON to Model	Manually with `json.loads()`	`.is_valid()` and `.save()`
Validation	Manual and error-prone	Auto validation
Error Handling	Manual messages	Built-in error responses
Clean, Reusable Code	No	Yes

---

8. Manual Serializer (Without Model)

```
class ManualEmployeeSerializer(serializers.Serializer):
    emp_name = serializers.CharField(max_length=100)
    emp_email = serializers.EmailField()
    emp_mob = serializers.CharField(max_length=15)

    def create(self, validated_data):
        return EMPLOYEE.objects.create(**validated_data)  # Called in
serializer.save() for POST
```

```
    def update(self, instance, validated_data):
        instance.emp_name = validated_data.get('emp_name', instance.emp_name)
        instance.emp_email = validated_data.get('emp_email', instance.emp_email)
        instance.emp_mob = validated_data.get('emp_mob', instance.emp_mob)
        instance.save()
        return instance  # Called in serializer.save() for PUT/PATCH
```

- `create()`: defines how to save new objects to the database.
- `update()`: defines how to update an existing object.

---

9. Summary

- Use `ModelSerializer` for Django models (quick and easy)
- Use `Serializer` for custom validation or non-model data
- Always use serializers in DRF to keep your code clean, safe, and maintainable