Custom Serializer in DRF

What is Serializer?

A serializer is like a translator. It converts your model (Python object) into JSON (for API) and vice versa.

```
from rest_framework import serializers

class BookSerializer(serializers.Serializer):
    title = serializers.CharField(max_length=100)
    author = serializers.CharField(max_length=100)
    published = serializers.DateField()
```

What is ModelSerializer?

A ModelSerializer automatically: Reads your Django model, Converts fields into JSON and Validates and saves data.

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

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = '__all__'
```

This works for basic CRUD (Create, Read, Update, Delete).

What is a Custom Serializer?

It means you add extra logic or fields into your serializer, because the default serializer is not enough for your case.

For example

```
<!-- model.py -->
# Task model
class Task(models.Model):
    title = models.CharField(max_length=100)
    due_date = models.DateField()

<!-- Serializers.py -->
from rest_framework import serializers
from .models import Task
from datetime import date
```

```python
class TaskSerializer(serializers.ModelSerializer):
    # Extra field (not in model): days_remaining
    # This will calculate how many days are left until due_date
    days_remaining = serializers.SerializerMethodField()

    class Meta:
        model = Task  # Link this serializer to Task model
        fields = ['id', 'title', 'description', 'due_date', 'is_complete',
'days_remaining']
        # Include both model fields and our custom field

    # This method supports SerializerMethodField
    def get_days_remaining(self, obj):
        # obj is the Task object
        if obj.due_date:
            return (obj.due_date - date.today()).days
        return None

    # Custom validation for a specific field (title)
    def validate_title(self, value):
        if len(value) < 3:
            raise serializers.ValidationError("Title must be at least 3 characters
long.")
        return value

    # Custom validation for multiple fields (optional)
    def validate(self, data):
        if data.get('is_complete') and not data.get('due_date'):
            raise serializers.ValidationError("Completed tasks must have a due
date.")
        return data

    # Custom logic while creating an object
    def create(self, validated_data):
        print("Creating a new task:", validated_data)
        return super().create(validated_data)

    # Custom logic while updating an object (optional)
    def update(self, instance, validated_data):
        print(f"Updating Task [{instance.title}] with:", validated_data)
        return super().update(instance, validated_data)
```