ঠিক আছে! 😎
DRF **serializers** হলো API-র **heart**। এগুলো শুধু data convert করে না, বরং **validation, nested creation, and business logic** handle করতেও ব্যবহার করা হয়।
আমি step-by-step **কি কি handle করা উচিত এবং best practices** দেখাচ্ছি।

1️⃣ Field Declaration / Mapping

- Serializer এ model fields declare করা হয়।
- Example:

```
class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = ['id', 'name', 'category', 'price', 'stock']
```

Best Practices:

- `read_only_fields` use করো যেটা user এডিট করতে পারবে না।
- `extra_kwargs` দিয়ে extra validation বা default settings দিতে পারো।

2️⃣ Validation

Serializer এ **field-level এবং object-level validation** করা হয়।

Field-level validation

```
def validate_price(self, value):
    if value <= 0:
        raise serializers.ValidationError("Price must be greater than 
zero.")
    return value
```

Object-level validation

```
def validate(self, data):
    if data['sale_price'] > data['price']:
        raise serializers.ValidationError("Sale price cannot exceed 
regular price.")
    return data
```

Best Practices:

- Small checks → field-level
- Cross-field rules → object-level

3️⃣ Nested Serialization

Parent-child relationship handle করতে nested serializer ব্যবহার করা হয়।

```
class SaleItemSerializer(serializers.ModelSerializer):
    product = ProductSerializer(read_only=True)

    class Meta:
        model = SaleItem
        fields = ['id', 'product', 'quantity', 'price']
```

Best Practices:

- Nested creation/updates handle করতে `create()` / `update()` override করো।
- `read_only=True` দিয়ে circular writes avoid করো।

4️⃣ Nested Creation Example

Sale + SaleItem creation:

```
class SaleSerializer(serializers.ModelSerializer):
    items = SaleItemSerializer(many=True)

    class Meta:
        model = Sale
        fields = ['id', 'customer', 'date', 'items']

    def create(self, validated_data):
        items_data = validated_data.pop('items')
        sale = Sale.objects.create(**validated_data)
        for item_data in items_data:
            SaleItem.objects.create(sale=sale, **item_data)
        return sale
```

Best Practices:

- Use `transaction.atomic()` if multiple table writes → ensure atomicity.

5️⃣ Business Logic

- Stock updates, discounts, taxes, and totals **usually handled in serializer create/update**.

```
from django.db import transaction

def create(self, validated_data):
    items_data = validated_data.pop('items')
    with transaction.atomic():
        sale = Sale.objects.create(**validated_data)
        for item_data in items_data:
            product = item_data['product']
            if product.stock < item_data['quantity']:
                raise serializers.ValidationError(f"{product.name} out of
stock")
            product.stock -= item_data['quantity']
            product.save()
            SaleItem.objects.create(sale=sale, **item_data)
    return sale
```

Best Practices:

- Serializer should **validate and enforce business rules**, not just convert data.

6️⃣ Read-only / Write-only Fields

- Sensitive info → `write_only=True`
- Computed fields → `read_only=True`

```
total_amount = serializers.FloatField(read_only=True)
password = serializers.CharField(write_only=True)
```

7️⃣ Computed / SerializerMethodField

- Dynamic data calculation:

```
total_amount = serializers.SerializerMethodField()
```

```
def get_total_amount(self, obj):
    return sum(item.quantity * item.price for item in obj.items.all())
```

Best Practices:

- Avoid heavy DB queries in SerializerMethodField → use `select_related` / `prefetch_related`.

8️⃣ Performance Tips

- Use `select_related` for FK fields
- Use `prefetch_related` for reverse M2M / FK relationships
- Avoid N+1 queries in nested serializers

9️⃣ Error Handling

- Always raise `serializers.ValidationError` for invalid data
- Return descriptive messages for API consumers

✅ Summary – What to Handle in Serializers

Responsibility	How / Example
Field mapping	`fields`, `read_only_fields`, `extra_kwargs`
Validation	`validate_field`, `validate`, business rules
Nested creation	override `create()` / `update()`
Computed fields	`SerializerMethodField`
Business logic	Stock updates, totals, discounts
Performance	`select_related`, `prefetch_related`
Security	`write_only=True` for sensitive info
Atomicity	`transaction.atomic()` for multi-table writes

💡 Rule of Thumb:

- Serializer = **validation + conversion + business logic enforcement**

- View = **handle request/response + permissions**

যদি চাও, আমি তোমার **Super Shop project এর Sale Serializer + Purchase Serializer** full **nested + stock handling + validation** ready করে দেখাতে পারি।
এটি দিয়ে তুমি practicalভাবে সব best practice শিখতে পারবে।

চাও আমি সেটা বানাই?