`Django REST Framework (DRF) Serializer holo DRF er ekta core component, ja Django model theke data ke JSON, XML, ba onnano format e convert kore (serialization), ar tar reverse process (deserialization) o kore. Eita API banate onek important.

`Cholo, DRF Serializer er **A to Z** bistarito vabe bujhi:

`1. Serializer ki?

- `Serializer data ke **Python objects** theke **JSON/XML** er moto easily transferable format e convert kore (serialization).
- `Ar **client theke asha JSON/XML** ke Python object e convert kore (deserialization).
- `Ekhane validation o kora hoy.

---

2. `Serializer types

a) `Serializer` class (Manual)

- `serializers.Serializer diye manually field gula define korte hoy.
- `Validation, create, update sob kichu manually likhte hoy.

```
from rest_framework import serializers

class EmployeeSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    first_name = serializers.CharField(max_length=100)
    last_name = serializers.CharField(max_length=100)
    email = serializers.EmailField()
    is_active = serializers.BooleanField(default=True)

    def create(self, validated_data):
        return Employee.objects.create(**validated_data)

    def update(self, instance, validated_data):
        instance.first_name = validated_data.get('first_name',
instance.first_name)
        instance.last_name = validated_data.get('last_name',
instance.last_name)
        instance.email = validated_data.get('email', instance.email)
        instance.is_active = validated_data.get('is_active',
instance.is_active)
```

```
        instance.save()
        return instance
```

b) `ModelSerializer` (Recommended)

- Django model theke automatically serializer create kore.
- `fields`, `exclude`, `read_only_fields` er maddhome customize kora jay.
- Create & update method auto handle kore.

```
from rest_framework import serializers
from employees.models import Employee

class EmployeeSerializer(serializers.ModelSerializer):
    class Meta:
        model = Employee
        fields = ['id', 'first_name', 'last_name', 'email', 'is_active']
        read_only_fields = ['id']
```

---

3. Serializer Fields

DRF onek built-in fields dey, jemon:

Field Type	Description
`CharField`	String data
`IntegerField`	Integer data
`FloatField`	Float data
`BooleanField`	Boolean value
`EmailField`	Email address
`DateField`	Date
`DateTimeField`	Date and time
`SerializerMethodField`	Custom method diye field create
`ChoiceField`	Choices theke value select kore

---

4. `Serializer Validation

a) Field-level validation

```python
class EmployeeSerializer(serializers.ModelSerializer):
    class Meta:
        model = Employee
        fields = ['email']

    def validate_email(self, value):
        if not value.endswith('@company.com'):
            raise serializers.ValidationError("Email must be company
domain.")
        return value
```

b) Object-level validation

```python
def validate(self, data):
    if data['start_date'] > data['end_date']:
        raise serializers.ValidationError("End date must be after start
date.")
    return data
```

---

5. `Serializer Methods

- `create(self, validated_data)` — `data save korar jonno
- `update(self, instance, validated_data)` — existing data update `korar jonno
- `to_representation(self, instance)` — `data ke output format e convert korar jonno (response)
- `to_internal_value(self, data)` — input data ke Python format e `convert korar jonno

---

6. `Nested Serializers

- One-to-many or many-to-many related data `ke nested vabe serialize korte use hoy.

```python
class DepartmentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Department
        fields = ['id', 'name']

class EmployeeSerializer(serializers.ModelSerializer):
    department = DepartmentSerializer(read_only=True)

    class Meta:
```

```
        model = Employee
        fields = ['id', 'first_name', 'department']
```

---

7. SerializerMethodField

- Jodi kono field model e na thake kintu response e thakte chai, tokhon use hoy.

```
class EmployeeSerializer(serializers.ModelSerializer):
    full_name = serializers.SerializerMethodField()

    class Meta:
        model = Employee
        fields = ['id', 'first_name', 'last_name', 'full_name']

    def get_full_name(self, obj):
        return f"{obj.first_name} {obj.last_name}"
```

---

8. `Serializer Context

- `View theke extra data serializer e pathate` context` use hoy.

```
serializer = EmployeeSerializer(employee, context={'request': request})
```

Serializer er modhe `self.context` diye access kora jay.

---

9. Partial Updates

- PATCH request e kono field na dileo hoy.

```
serializer = EmployeeSerializer(instance, data=request.data, partial=True)
```

---

10. Custom Serializer Fields

- Nijer custom field o banaite paro.

```
class UpperCaseCharField(serializers.CharField):
    def to_representation(self, value):
```

```
        return value.upper()
```

---

11. Serializer Performance Tips

- `select_related` and `prefetch_related` diye queryset optimize koro.
- Nested serializer er moddhe `read_only=True` use koro jekhane data change korbe na.

Summary

Concept	Description
Serializer	Python objects ↔ JSON conversion
`Serializer`	Manual serializer
`ModelSerializer`	Auto serializer from model
Fields	Built-in field types
Validation	Field-level & object-level
create/update	Save & update data methods
Nested serializer	Related object serialization
SerializerMethodField	Custom field from method
Context	Extra data pass from view
Partial updates	PATCH support

`**Note:** DRF Serializer holo REST API er backbone. Bujhle API build kora khub easy hoye jay. Ki dhoroner API kintu banate chaccho?`

---

example:

Sure! Ami ekta **simple project er example** diye step-by-step DRF Serializer er **sob important part** bujhaye dichchi.

Project scenario:

Suppose ekta company ache — **Department** and **Employee** ache.

- Department e: `id`, `name`
- Employee e: `id`, `first_name`, `last_name`, `email`, `department` (ForeignKey to Department), `date_joined`

---

1. Models (for context)

```
from django.db import models

class Department(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Employee(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    department = models.ForeignKey(Department, related_name='employees',
on_delete=models.CASCADE)
    date_joined = models.DateField()

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

---

2. Serializer example with everything included

```
from rest_framework import serializers
from .models import Department, Employee

# Department Serializer (simple ModelSerializer)
class DepartmentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Department
        fields = ['id', 'name']

# Employee Serializer with nested Department and custom fields
class EmployeeSerializer(serializers.ModelSerializer):
    # Nested Department Serializer (read-only)
```

```
    department = DepartmentSerializer(read_only=True)

    # Accept department id for writing
    department_id = serializers.PrimaryKeyRelatedField(
        queryset=Department.objects.all(), source='department',
write_only=True
    )

    # Custom SerializerMethodField (full name)
    full_name = serializers.SerializerMethodField()

    class Meta:
        model = Employee
        fields = ['id', 'first_name', 'last_name', 'full_name', 'email',
'department', 'department_id', 'date_joined']

    # Method to get full_name dynamically
    def get_full_name(self, obj):
        return f"{obj.first_name} {obj.last_name}"

    # Field-level validation example
    def validate_email(self, value):
        if not value.endswith('@company.com'):
            raise serializers.ValidationError("Email must be from
company.com domain.")
        return value

    # Object-level validation example
    def validate(self, data):
        if 'date_joined' in data and data['date_joined'] >
timezone.now().date():
            raise serializers.ValidationError("Date joined cannot be in
the future.")
        return data

    # create method (for deserialization)
    def create(self, validated_data):
        return Employee.objects.create(**validated_data)

    # update method (for deserialization)
    def update(self, instance, validated_data):
        instance.first_name = validated_data.get('first_name',
instance.first_name)
        instance.last_name = validated_data.get('last_name',
instance.last_name)
        instance.email = validated_data.get('email', instance.email)
        instance.department = validated_data.get('department',
instance.department)
        instance.date_joined = validated_data.get('date_joined',
instance.date_joined)
```

```
        instance.save()
        return instance
```

---

3. Explanation of important points in this example

Feature	Explanation
`ModelSerializer`	Automatically generates fields from the model.
Nested Serializer (`department`)	Shows full Department data inside Employee data on GET (read-only).
`department_id` (PrimaryKeyRelatedField)	Allows setting Department by ID when creating/updating Employee (write-only).
`SerializerMethodField` (`full_name`)	Dynamically returns employee full name combining first & last name.
Field-level validation	`validate_email` ensures email ends with `@company.com` .
Object-level validation	`validate` checks date_joined is not in the future.
`create()` method	How new Employee instance is created after validation.
`update()` method	How existing Employee instance is updated.

---

4. Sample usage in view (for completeness)

```
from rest_framework import viewsets
from .models import Employee
from .serializers import EmployeeSerializer

class EmployeeViewSet(viewsets.ModelViewSet):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer
```

---

5. Sample serialized output (GET /employees/1/)

```
{
  "id": 1,
  "first_name": "John",
```

```
  "last_name": "Doe",
  "full_name": "John Doe",
  "email": "john.doe@company.com",
  "department": {
    "id": 3,
    "name": "HR"
  },
  "date_joined": "2023-01-15"
}
```

---

6. Sample input for creating an employee (POST /employees/)

```
{
  "first_name": "Alice",
  "last_name": "Smith",
  "email": "alice.smith@company.com",
  "department_id": 3,
  "date_joined": "2023-10-01"
}
```

---

7. Bonus: Partial update example (PATCH)

- You can send only the fields you want to update, e.g.

```
{
  "email": "alice.new@company.com"
}
```

and in view:

```
serializer = EmployeeSerializer(instance, data=request.data, partial=True)
if serializer.is_valid():
    serializer.save()
```

Summary for this example

Concept	Usage in example
`ModelSerializer`	`EmployeeSerializer` extends it
Nested Serializer	`department` field
Related field write access	`department_id` field
Custom field	`full_name` SerializerMethodField
Field validation	`validate_email` method
Object validation	`validate` method
Create & Update methods	`create()` and `update()` overridden
Partial update	`partial=True` in serializer instantiation

---

another example:

`Sure! Another complex example diye dichhi — ei bar **e-commerce system** with:

- `Products, Categories, and Reviews
- `Product belongs to multiple Categories (ManyToMany)
- `Product has multiple Reviews (ForeignKey)
- `Review linked to User
- `Nested writable serializers for Categories and Reviews
- `Custom validations and custom fields

Models

```
from django.db import models
from django.contrib.auth.models import User

class Category(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField(blank=True)

class Product(models.Model):
    name = models.CharField(max_length=200)
    description = models.TextField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    categories = models.ManyToManyField(Category, related_name='products')
    created_at = models.DateTimeField(auto_now_add=True)

class Review(models.Model):
```

```
    product = models.ForeignKey(Product, related_name='reviews',
on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    rating = models.IntegerField()  # 1 to 5
    comment = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
```

Serializers

```
from rest_framework import serializers
from django.contrib.auth.models import User
from .models import Category, Product, Review

class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = ['id', 'name', 'description']

class ReviewSerializer(serializers.ModelSerializer):
    user = serializers.StringRelatedField(read_only=True)  # show username

    class Meta:
        model = Review
        fields = ['id', 'user', 'rating', 'comment', 'created_at']

    def validate_rating(self, value):
        if value < 1 or value > 5:
            raise serializers.ValidationError("Rating must be between 1
and 5.")
        return value

class ProductSerializer(serializers.ModelSerializer):
    categories = CategorySerializer(many=True)  # nested writable
    reviews = ReviewSerializer(many=True, read_only=True)  # nested read-
only reviews
    average_rating = serializers.SerializerMethodField()

    class Meta:
        model = Product
        fields = ['id', 'name', 'description', 'price', 'categories',
'reviews', 'average_rating', 'created_at']

    def get_average_rating(self, obj):
        reviews = obj.reviews.all()
        if not reviews.exists():
            return None
```

```
        return round(sum([r.rating for r in reviews]) / reviews.count(),
2)

    def create(self, validated_data):
        categories_data = validated_data.pop('categories')
        product = Product.objects.create(**validated_data)

        # create or get categories
        for cat_data in categories_data:
            category, created =
Category.objects.get_or_create(name=cat_data['name'], defaults=
{'description': cat_data.get('description', '')})
            product.categories.add(category)
        return product

    def update(self, instance, validated_data):
        categories_data = validated_data.pop('categories', None)

        instance.name = validated_data.get('name', instance.name)
        instance.description = validated_data.get('description',
instance.description)
        instance.price = validated_data.get('price', instance.price)
        instance.save()

        if categories_data is not None:
            instance.categories.clear()
            for cat_data in categories_data:
                category, created =
Category.objects.get_or_create(name=cat_data['name'], defaults=
{'description': cat_data.get('description', '')})
                instance.categories.add(category)

        return instance
```

Explanation

Feature	Description
Nested writable categories	ProductSerializer handles nested create/update of categories
Nested read-only reviews	Reviews are shown but not created/updated here
Custom field	`average_rating` dynamically calculates average rating for product
Validation	Rating validated in ReviewSerializer
`StringRelatedField`	Shows username as string in ReviewSerializer

Feature	Description
ManyToMany relationship	Handled via `categories` field in ProductSerializer

---

Sample Input (POST Product)

```
{
  "name": "Gaming Laptop",
  "description": "High performance laptop for gaming.",
  "price": "1299.99",
  "categories": [
    {"name": "Electronics", "description": "Electronic devices"},
    {"name": "Computers"}
  ]
}
```

Sample Output (GET Product)

```
{
  "id": 1,
  "name": "Gaming Laptop",
  "description": "High performance laptop for gaming.",
  "price": "1299.99",
  "categories": [
    {"id": 1, "name": "Electronics", "description": "Electronic devices"},
    {"id": 2, "name": "Computers", "description": ""}
  ],
  "reviews": [
    {
      "id": 5,
      "user": "john_doe",
      "rating": 5,
      "comment": "Awesome laptop!",
      "created_at": "2025-11-30T10:00:00Z"
    }
  ],
  "average_rating": 4.75,
  "created_at": "2025-11-01T15:00:00Z"
}
```

Notes

- For **review creation**, you can create a separate API or serializer that uses the user from request context and links to product.
- You can add more validations like price > 0, category name unique, etc.
- This example shows how to handle **nested writable ManyToMany** with categories and **nested read-only reverse relation** with reviews.
