Validation

Field Level Validation

We can specify custom field-level validation by adding validate fieldName methods to your Serializer subclass.
These are similar to the clean _fieldName methods on Django forms.
Validate _fieldName methods should return the validated value or raise a serializer.
ValidationsError
Syntax:- def validate_fieldname(self, value)

```
from rest_framework import serializers
from api.models import Student

class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)
    roll = serializers.IntegerField()
    city = serializers.CharField(max_length=100)

    # Field-level validations
    def validate_name(self, value):
        if len(value) < 3:
            raise serializers.ValidationError('Name must be at least 3 characters.')
        return value

    def validate_roll(self, value):
        if value >= 200:
            raise serializers.ValidationError('Seat full.')
        if value <= 0:
            raise serializers.ValidationError('Roll must be positive.')
        return value

    def validate_city(self, value):
        if len(value) < 3:
            raise serializers.ValidationError('City name must be at least 3 characters.')
        return value
```

Object Level Validation

When we need to do validation that requires access to multiple fields we do object validation by adding a method called validate() to the Serializer subclass.
It raises a serializer. ValidationError if necessary, or just return the validated values.
Syntax:- def validate(self, data)
Where, data is data is a dictionary of field values.

```
from rest_framework import serializers
from api.models import Student

class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)
```

```
    roll = serializers.IntegerField()
    city = serializers.CharField(max_length=100)

    # Object Level Validation
    def validate(self, data):
        name = data.get('name')
        roll = data.get('roll')
        city = data.get('city')

        if name and city and name.lower() == city.lower():
            raise serializers.ValidationError("Name and city cannot be the same.")

        if roll and (roll <= 0 or roll > 1000):
            raise serializers.ValidationError("Roll number must be between 1 and 1000.")

        if name and len(name) < 2:
            raise serializers.ValidationError("Name must be at least 2 characters long.")

        return data
```

Validators

Most of the time you're dealing with validation in REST framework you'll simply be relying on the default field validation, or writing explicit validation methods on serializer or field classes

.

However, sometimes you'll want to place your validation logic into reusable components, so that it can easily be reused throughout your codebase. This can be achieved by using functions and validator classes.

```
from rest_framework import serializers
from api.models import Student

# Custom validator for name
def validate_name(value):
    if len(value) < 2:
        raise serializers.ValidationError("Name must be at least 2 characters long.")
    return value

# Custom validator for roll
def validate_roll(value):
    if value <= 0 or value > 1000:
        raise serializers.ValidationError("Roll number must be between 1 and 1000.")
    return value

# Custom validator for city (example: name and city should not match)
def validate_city(value, serializer_field=None):
    # Access the other fields if serializer_field is provided
    if serializer_field:
        name = serializer_field.context.get('name')
```

```
        if name and name.lower() == value.lower():
            raise serializers.ValidationError("City cannot be the same as name.")
    return value

class StudentSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100, validators=[validate_name])
    roll = serializers.IntegerField(validators=[validate_roll])
    city = serializers.CharField(max_length=100)

    def validate_city(self, value):
        # field-level validation with access to other fields if needed
        name = self.initial_data.get('name')
        if name and name.lower() == value.lower():
            raise serializers.ValidationError("City cannot be the same as name.")
        return value

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

    def update(self, instance, validated_data):
        instance.name = validated_data.get('name', instance.name)
        instance.roll = validated_data.get('roll', instance.roll)
        instance.city = validated_data.get('city', instance.city)
        instance.save()
        return instance
```