Validações

No Django Rest Framework (DRF), as validações são fundamentais para garantir que os dados enviados para a API sejam válidos antes de serem processados ou armazenados. O DRF oferece diversas formas de validar dados, desde validações automáticas em **serializers** até validações personalizadas.

**Como as Validações Funcionam no DRF?**

1. **Entrada de Dados**:
   * O cliente envia dados para a API (geralmente via POST, PUT ou PATCH).
2. **Serializers**:
   * O DRF utiliza **serializers** para converter os dados JSON enviados para objetos Python e verificar sua validade.
3. **Validações**:
   * Validações são aplicadas automaticamente com base nos campos do serializer.
   * Regras customizadas podem ser implementadas para requisitos específicos.
4. **Resposta**:
   * Se os dados forem válidos, eles são processados.
   * Se forem inválidos, o DRF retorna um erro com detalhes das falhas de validação.

**Tipos de Validações no DRF**

1. **Validações Baseadas nos Campos do Serializer**
   * Cada campo no serializer pode ter regras de validação específicas.
   * Exemplo:
     + **Obrigatoriedade**: required=True.
     + **Tamanho máximo/mínimo**: max\_length e min\_length.
     + **Tipos de dados**: Inteiros, strings, datas, etc.

**Exemplo:**

from rest\_framework import serializers

class EbookSerializer(serializers.Serializer):

title = serializers.CharField(max\_length=100, required=True)

author = serializers.CharField(max\_length=50, required=True)

published\_date = serializers.DateField(required=False)

Se o cliente enviar dados inválidos, como um título com mais de 100 caracteres, a API retornará:

{

"title": ["Ensure this field has no more than 100 characters."]

}

**Validações Globais no Serializer**

* Para regras que dependem de múltiplos campos, você pode usar o método validate() no serializer.

**Exemplo:** Verificar se a data de publicação não é no futuro.

from datetime import date

class EbookSerializer(serializers.Serializer):

title = serializers.CharField(max\_length=100, required=True)

published\_date = serializers.DateField(required=False)

def validate(self, data):

published\_date = data.get('published\_date')

if published\_date and published\_date > date.today():

raise serializers.ValidationError("A data de publicação não pode ser no futuro.")

return data

**Validações Personalizadas para Campos Específicos**

* Para regras específicas de um único campo, use o método validate\_<nome\_do\_campo>.

**Exemplo:** Verificar se o título começa com "Ebook".

class EbookSerializer(serializers.Serializer):

title = serializers.CharField(max\_length=100, required=True)

def validate\_title(self, value):

if not value.startswith("Ebook"):

raise serializers.ValidationError("O título deve começar com 'Ebook'.")

return value

**Validações em ModelSerializer**

* Quando você usa um ModelSerializer, as validações automáticas são baseadas no modelo associado.

**Exemplo:**

from django.db import models

from rest\_framework import serializers

class Ebook(models.Model):

title = models.CharField(max\_length=100)

author = models.CharField(max\_length=50)

class EbookSerializer(serializers.ModelSerializer):

class Meta:

model = Ebook

fields = '\_\_all\_\_'

Regras como max\_length e null definidas no modelo são automaticamente aplicadas no serializer.

**Validações Customizadas (Funções Externas)**

* Você pode criar funções separadas para validações complexas e aplicá-las a campos do serializer.

**Exemplo:**

def validate\_title\_length(value):

if len(value) < 10:

raise serializers.ValidationError("O título deve ter pelo menos 10 caracteres.")

return value

class EbookSerializer(serializers.Serializer):

title = serializers.CharField(validators=[validate\_title\_length])

**Validações Baseadas em Objetos (Método is\_valid)**

* Quando você chama serializer.is\_valid(), ele executa todas as validações configuradas.

**Exemplo:**

data = {"title": "Python Basics", "published\_date": "2030-01-01"}

serializer = EbookSerializer(data=data)

if serializer.is\_valid():

# Dados válidos

print(serializer.validated\_data)

else:

# Dados inválidos

print(serializer.errors)

**Erros de Validação**

* Quando os dados não passam nas validações, o DRF retorna uma resposta com código HTTP 400 Bad Request e o corpo detalha os erros de validação.

**Exemplo de resposta de erro:**

{

"title": ["Ensure this field has no more than 100 characters."],

"published\_date": ["A data de publicação não pode ser no futuro."]

}

**Resumo**

| **Método de Validação** | **Propósito** | **Onde Usar** |
| --- | --- | --- |
| Campos do Serializer | Validações simples por campo (ex.: max\_length, required). | Validações padrão por tipo de dado. |
| validate\_<nome\_do\_campo>() | Validações customizadas para um único campo. | Regras específicas para um campo. |
| validate() | Validações globais envolvendo múltiplos campos. | Dependência entre campos. |
| Validações em ModelSerializer | Validações automáticas baseadas no modelo. | Quando se usa um modelo Django diretamente. |
| Validações com funções externas | Reutilizar validações complexas entre serializers. | Lógica de validação modular. |

Com essas ferramentas, o DRF permite criar APIs robustas com validações confiáveis e personalizadas!