Validierung

von Daten im Django Restframework

Validierung in Django REST Framework

- Validierung ist der Prozess der Überprüfung, ob eingegebene Daten korrekt und sinnvoll sind.
- In DRF wird die Validierung verwendet, um die Integrität eingehender Daten zu sichern.

1. Feldspezifische Validierung mit extra_kwargs

- Überprüft einzelne Felder auf Gültigkeit.
- mit extra_kwargs auf Standard-Fehler prüfen: `max_length`, `min_value`, `required`.

2. Cross-Field Validierung

- Überprüft die Daten alle Felder
- Beispiel: Die Kombination von Start- und Enddatum ist gültig.

3. Validator Funktionen

- Benutzerdefinierte Funktionen zur Überprüfung der Daten.
- Beispiel: Eine E-Mail-Adresse ist einzigartig in der Datenbank.

Implementierung

- Validierungen werden im Serializer definiert.
- DRF bietet eine Vielzahl von eingebauten Validierern.
- Eigene Validierungslogik kann leicht hinzugefügt werden.

1. Feldspezifische Validierung

Auf Feldebene können Sie in Django Rest Framework (DRF) benutzerdefinierte Validierungen durchführen, indem Sie validate_<feldname> Methoden zu Ihrem Serializer hinzufügen. Diese Methoden sind den clean_<feldname> Methoden in Django-Formularen ähnlich.

Diese Methoden nehmen einen einzelnen Parameter entgegen, der den aktuellen Feldwert darstellt, der validiert werden muss.

Ihre validate_<feldname> Methoden sollten entweder den validierten Wert zurückgeben oder einen serializers.ValidationError Fehler werfen. Dieser wirft einen HTTP_400_BAD_REQUEST.

Beispiel

```
class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Person
        fields = ['name', 'age']

    def validate_name(self, value):
        """
        Stellen Sie sicher, dass der Name nicht leer ist und nicht nur aus Leerzeichen besteht.
        """
        if value.strip() == '':
            raise serializers.ValidationError("Der Name darf nicht leer sein.")
        return value
```

extra_kwargs

extra_kwargs ist ein Attribut, das im Serializer verwendet wird, um zusätzliche Argumente für die Felder des Serializers anzugeben. Es ermöglicht eine feinere Kontrolle und Anpassung des Verhaltens der einzelnen Felder. extra_kwargs wird typischerweise in der Meta-Klasse eines Serializers definiert und ist besonders nützlich, wenn Sie das Standardverhalten der automatisch generierten Felder anpassen möchten.

Verwendung von extra_kwargs:

Sie können extra_kwargs verwenden, um verschiedene Aspekte der Felder anzupassen, wie Validierung, Anzeige, Schreibbarkeit, usw. Einige gängige Anwendungen sind:

Festlegen von read_only=True, um ein Feld schreibgeschützt zu machen.

Definieren von write_only=True, um sicherzustellen, dass ein Feld nur für Schreiboperationen verwendet wird (beispielsweise für Passwörter).

Angeben von Validierungsoptionen wie max_length.

extra_kwargs Beispiel

Hier ist ein Beispiel, wie extra_kwargs in einem Serializer verwendet werden kann:

```
from rest_framework import serializers
from .models import Item
from .custom_validators import custom_name_validator

class ItemSerializer(serializers.ModelSerializer):
    class Meta:
        model = Item
        fields = ['id', 'name', 'description', 'password']
        extra_kwargs = {
            'password': {'write_only': True},
            'description': {'max_length': 100, 'error_messages': {'max_length': 'Description is too long.'}},
            'name': {'validators': [custom_name_validator]}
        }
```

2. Cross-Field Validierung

Die Methode validate(self, data) des Serializers bietet über data Zugriff auf alle aktullen Feldwerte. data ist ein dictionary, welches auch wieder zugrückgegeben werden muss.

```
class ItemSerializer(serializers.ModelSerializer):
    class Meta:
        model = Item
        fields = ['start_date', 'end_date']

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

3. Validator-Funktionen

Es lassen sich leicht eigene Validator-Funktionen anlegen und diese über extra_kwargs dem Serializier hinzufügen.

```
def validate_email(value):
    if 'example.com' not in value:
        raise serializers.ValidationError("Email must be from example.com domain")


extra_kwargs = {
            'email': {'validators': [validate_email]}
        }
```