Nestest Serializer in Django Framework

Last Updated : 18 Mar, 2024

In web development, creating and consuming APIs (Application Programming Interfaces) is commonplace. Django Rest Framework (DRF) serves as a robust toolkit for building APIs in Django-based web applications. Within DRF, a pivotal concept is serializers. In this article, we will delve into the concept of nested serializers in Django and how they facilitate the handling of complex data relationships.

Significance of Nested Serializers

In many real-world scenarios, data models exhibit relationships with one another. For instance, you may have a Book model associated with a Category model through a foreign key. In such cases, merely serializing the Book object may not be sufficient; you might want to include the related Category information within the serialized output. This is precisely where nested serializers come into play.

Setting up the Project

Let's walk through a step-by-step implementation of nested serializers in Django to understand how they work in practice. Consider a scenario where we have two models, Category and Book, with each book belonging to a category. Our objective is to serialize a Book object along with its associated Category information.

- To install Django follow these steps.
- To install Django Rest Framework follow these steps.

Starting the Project

To start the project use this command

```
django-admin startproject core
cd core
```

To start the app use this command

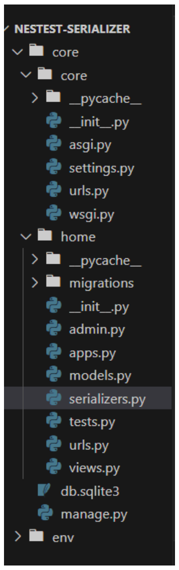
```
python manage.py startapp home
```

Now add this app and **'rest_framework'** to the **'settings.py'**

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "rest_framework",
    "home",
    "rest_framework.authtoken",

]
```

File Structure



Setting up the files

serializer.py: This Python code uses the Django REST framework to create two serializers: **CategorySerializer** and **BookSerializer**.

- CategorySerializer serializes the Category model, including only the category_name field.
- BookSerializer serializes the Book model with all fields. It also includes serialization of the related Category model.
- The commented-out line # depth = 1 suggests the ability to control the depth of related model serialization, but it's not currently active.
- These serializers allow converting Django model instances into JSON data for use in API responses or requests

Python

```python
from rest_framework import serializers
from .models import *
from django.contrib.auth.models import User
class GFGSerializer(serializers.ModelSerializer):
    class Meta:
        model = GFG
        fields = ['name',]

class ItemSerializer(serializers.ModelSerializer):

    gfg = GFGSerializer()

    class Meta:
        model = Item
        fields = '__all__'
        depth =1
```

models.py: This Python code defines two Django models:

- **Category Model:** Represents a category with a category_name field for its name.
- **Book Model:**
    - Represents a book with a book_title field for its title.

- Includes a foreign key relationship (category) to the Category model, connecting each book to a category.
- **Uses on_delete=models.CASCADE** to ensure that when a category is deleted, its associated books are also deleted.

Python

```python
from django.db import models

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

class Item(models.Model):
    gfg = models.ForeignKey(GFG, on_delete=models.CASCADE)
    item_title = models.CharField(max_length=100)
```

admin.py: First we register Category and then we register Book model.

Python

```python
from django.contrib import admin
from .models import *

admin.site.register(GFG)
admin.site.register(Item)
```

views.py: Various Django and Django REST framework modules and classes are imported.A view function **get_book** is defined using the **@api_view** decorator, specifying that it handles HTTP GET requests.All Book objects are retrieved from the database usin**g Book.objects.all().**The retrieved Book objects are serialized using the **BookSerializer.**A JSON response is returned with a status code of 200 and the serialized book data as the payload.

Python

```python
from django.shortcuts import render
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .models import *
from .serializers import *
from rest_framework.views import APIView
from rest_framework.authtoken.models import Token


@api_view(['GET'])
def get_item(request):
    item_objs = Item.objects.all()
    serializer = ItemSerializer(item_objs, many=True)
    return Response({'status': 200, 'payload': 
serializer.data})
```

core/urls.py: This is the urls.py file of our project folder in this urls.py file we import first all important library and after that we map the app with project using include function.

Python

```python
from django.contrib import admin
from django.urls import path, include
from rest_framework.authtoken import views

urlpatterns = [
    path('', include('home.urls')),
    path("admin/", admin.site.urls),

]
```

app/urls.py: In the urls.py file of the app we import the admin and also import the path and include method and also import the all functionality of views.py file . and then we create the path for performing the **nestest serailizer** operations

Python

```python
from django.contrib import admin
from django.urls import path, include
from .views import *


urlpatterns = [
    path('get-item/', get_item),

]
```

Deployement of the Project

Run these commands to apply the migrations:

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Run the server with the help of following command:

```
python3 manage.py runserver
```

Output

Conclusion :

In conclusion, nested serializers in Django provide a powerful mechanism for handling complex data relationships in API development. By incorporating related model serializers within primary serializers, developers can efficiently serialize and deserialize data hierarchies, delivering structured and informative API responses. Whether dealing with parent-child relationships, deep dependencies, or multi-level data structures, nested serializers enhance the flexibility and usability of Django-powered APIs, making them a valuable tool for creating rich and efficient web applications.

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our Mastering Django Framework - Beginner to Advanced Course on GeeksforGeeks, designed for aspiring developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

prath...

1

Previous Article

Next Article

File upload Fields in Serializers - Django REST Framework

Similar Reads

Core arguments in serializer fields - Django REST Framework

Serializer fields in Django are same as Django Form fields and Django model fields and thus require certain arguments to manipulate the behaviour of those...

6 min read

Serializer Fields - Django REST Framework

Serializer comes with some fields (entries) that process data in and out of the serializer in Django REST Framework. The very motive of Serializing is to conver...

13 min read

Serializer Relations - Django REST Framework

Serialization is one of the most important concepts in RESTful Webservices. It facilitates the conversion of complex data (such as model instances) to native...

15+ min read

Get Request.User in Django-Rest-Framework Serializer

In Django Rest Framework (DRF), serializers are used to transform complex data types, such as queryset and model instances, into native Python data types. One...

5 min read

How to Change Field Name in Django REST Framework Serializer

When working with Django REST Framework (DRF), we may encounter situations where we need to change the name of a field in a serializer. This can ...

3 min read

Pass Extra Arguments to Serializer Class in Django Rest Framework

Django Rest Framework (DRF) is a powerful toolkit for building web APIs. It provides serializers that translate complex data types such as Django querysets...

4 min read

Pass Request Context to Serializer from ViewSet in Django Rest Framework

Django Rest Framework (DRF) is a powerful toolkit for building Web APIs in Django. One of the features that make DRF so flexible is its ability to pass conte...

4 min read

How To Filter A Nested Serializer In Django Rest Framework?

When building APIs with Django Rest Framework (DRF), nested serializers are commonly used to represent relationships between models. A nested serializer...

6 min read

Integrating Django with Reactjs using Django REST Framework

In this article, we will learn the process of communicating between the Django Backend and React js frontend using the Django REST Framework. For the sake...

15+ min read

Implement Token Authentication using Django REST Framework

Token authentication refers to exchanging username and password for a token that will be used in all subsequent requests so to identify the user on the server...

2 min read

Article Tags : Python Geeks Premier League Python Django

Practice Tags : python

System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects