Certainly! Let's consider a scenario where we have models representing a `Library` that contains multiple `Book` objects, and each `Book` has multiple `Author` objects associated with it. We'll demonstrate how to serialize this complex data structure using nested serializers in Django Rest Framework (DRF.

python Copy code

```python
# models.py
from django.db import models

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

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    authors = models.ManyToManyField(Author)
    published_date = models.DateField()

    def __str__(self):
        return self.title

class Library(models.Model):
    name = models.CharField(max_length=100)
    books = models.ManyToManyField(Book)

    def __str__(self):
        return self.name
```

Now, let's create serializers for these models:

python Copy code

```python
# serializers.py
from rest_framework import serializers
from .models import Author, Book, Library
```

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

class BookSerializer(serializers.ModelSerializer):
    authors = AuthorSerializer(many=True)  # Nesting AuthorSerializer

    class Meta:
        model = Book
        fields = ['id', 'title', 'authors', 'published_date']

class LibrarySerializer(serializers.ModelSerializer):
    books = BookSerializer(many=True)  # Nesting BookSerializer

    class Meta:
        model = Library
        fields = ['id', 'name', 'books']
```

In this example:

- `AuthorSerializer` and `BookSerializer` are used to serialize `Author` and `Book` models, respectively.
- `BookSerializer` includes `AuthorSerializer` to handle the serialization of author objects associated with each book.
- `LibrarySerializer` includes `BookSerializer` to handle the serialization of book objects associated with each library.

Now, when you serialize a `Library` object using `LibrarySerializer`, it will automatically include details about all the books in the library, and each book will contain details about its authors.

python Copy code

```python
# views.py
from rest_framework.generics import ListAPIView
from .models import Library
from .serializers import LibrarySerializer
```

```python
class LibraryListView(ListAPIView):
    queryset = Library.objects.all()
    serializer_class = LibrarySerializer
```

With this setup, when you access an endpoint that returns a list of libraries, the response will contain nested JSON objects representing the library, its books, and the authors of each book, providing a comprehensive representation of the complex data structure.