Here's how to create the same API using Django function-based views rather than class-based views.

---

1. Set Up the Django Project and App (Same as Before)

Follow the initial setup steps:

1. Create a Django project and app (`myproject` and `myapp`).
2. Define the `Book` model.
3. Run migrations.
4. Set up `INSTALLED_APPS` in `settings.py` as described previously.

---

2. Create the Serializer (Same as Before)

Define the serializer for the `Book` model.

```
python
Copy code
# myapp/serializers.py
from rest_framework import serializers
from .models import Book

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = ['id', 'title', 'author', 'published_date', 'isbn']
```

---

3. Create Function-Based Views

Define function-based views in `views.py`. In function-based views, you'll need to use decorators and explicitly handle HTTP methods like `GET`, `POST`, `PUT`, and `DELETE`.

```
python
Copy code
# myapp/views.py
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status
from .models import Book
from .serializers import BookSerializer

@api_view(['GET', 'POST'])
def book_list_create(request):
    # GET method: List all books
    if request.method == 'GET':
```

```
        books = Book.objects.all()
        serializer = BookSerializer(books, many=True)
        return Response(serializer.data)

    # POST method: Create a new book
    elif request.method == 'POST':
        serializer = BookSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)


@api_view(['GET', 'PUT', 'DELETE'])
def book_detail(request, pk):
    try:
        book = Book.objects.get(pk=pk)
    except Book.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)

    # GET method: Retrieve a single book
    if request.method == 'GET':
        serializer = BookSerializer(book)
        return Response(serializer.data)

    # PUT method: Update a book
    elif request.method == 'PUT':
        serializer = BookSerializer(book, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    # DELETE method: Delete a book
    elif request.method == 'DELETE':
        book.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

---

4. Set Up URLs

Define URLs to route to the function-based views.

```
python
Copy code
# myapp/urls.py
from django.urls import path
from .views import book_list_create, book_detail

urlpatterns = [
    path('books/', book_list_create, name='book-list-create'),
    path('books/<int:pk>/', book_detail, name='book-detail'),
]
```

Include these URLs in the main `urls.py` file of the project:

```
python
Copy code
# myproject/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('myapp.urls')),
]
```

---

5. Test the API

Now, start the Django server:

```
bash
Copy code
python manage.py runserver
```

Access the API at:

- `http://127.0.0.1:8000/api/books/` – for listing and creating books.
- `http://127.0.0.1:8000/api/books/<id>/` – for retrieving, updating, or deleting a book.

---

6. Adding Authentication (Optional)

As with the class-based example, you can restrict access by configuring authentication settings in `settings.py` as shown earlier.

This completes the API using function-based views. This approach gives you more control over each HTTP method while still leveraging Django REST Framework's serialization and response capabilities.