GraphQL is a query language for APIs, while Django Rest Framework (DRF) is a toolkit for building RESTful APIs. While they serve similar purposes, they use different paradigms and data structures, so it's not immediately obvious how to use GraphQL with DRF.

Here's one approach for integrating GraphQL with DRF:

1. Install the necessary packages:

Copy code
```
pip install graphene-django
```
2. Create a schema for your GraphQL API using Graphene-Django. This involves defining the types, queries, and mutations that your API will support. Here's an example:

python

Copy code
```python
import graphene
from graphene_django import DjangoObjectType
from .models import Book

class BookType(DjangoObjectType):
    class Meta:
        model = Book

class Query(graphene.ObjectType):
    all_books = graphene.List(BookType)

    def resolve_all_books(self, info):
        return Book.objects.all()

schema = graphene.Schema(query=Query)
```

This schema defines a `Book` type and a `Query` object that returns a list of all books. Note that you'll need to import your Django models and use them to define the types in your schema.

3. Create a view in DRF that accepts GraphQL queries and returns JSON responses. Here's an example:

python

Copy code

```python
from rest_framework.views import APIView
from rest_framework.response import Response
from graphene_django.views import GraphQLView

class GraphQLAPIView(GraphQLView):
    def get_context(self, request):
        context = super().get_context(request)
        context["request"] = request
        return context

    def post(self, request, *args, **kwargs):
        response = super().post(request, *args, **kwargs)
        response.data["data"] = {"result": response.data["data"]}
        return response
```

This view subclasses `GraphQLView` from Graphene-Django and overrides the `get_context` method to include the DRF request object in the GraphQL context. It also overrides the `post` method to wrap the GraphQL response in a top-level "result" key, which is a common convention for GraphQL APIs.

4. Add a URL pattern in your DRF router that points to your GraphQL view. Here's an example:

python

Copy code

```
from django.urls import path, include
from rest_framework import routers

from .views import GraphQLAPIView

router = routers.DefaultRouter()

urlpatterns = [
    path("graphql", GraphQLAPIView.as_view(graphiql=True)),
    path("", include(router.urls)),
]
```

This sets up a URL pattern for the `/graphql` endpoint that points to your GraphQL view.

With these steps, you should now be able to use GraphQL to query your DRF API. You can send GraphQL queries as POST requests to `/graphql`, and you should receive JSON responses in the format:

json

Copy code

```
{
  "data": {
    "result": {
      "allBooks": [
        {
          "id": 1,
          "title": "Example book",
          "author": "Jane Doe",
          "published": "2022-01-01"
        },
        ...
      ]
```

```
        }
    }
}
```

Note that the top-level `data` key is added by Graphene-Django, and the `result` key is added by the custom view. You can customize the GraphQL schema and view to suit your specific needs.