**Using graphql with django**

If you don’t know what GraphQL is, it's a query language that gives clients the power to ask for exactly what they need from API’s, nothing more,nothing less. This makes it possible to get predictable results. Apps using GraphQL are fast and stable because they control the data they get, not the server.

**Setup graphene in django:**  
**Follow this link and make initial setup:**

<https://medium.com/swlh/introduction-to-graphql-using-django-ca7058006574>

<https://medium.com/@alhajee2009/graphql-with-django-a-tutorial-that-works-2812be163a26>

<https://stackabuse.com/building-a-graphql-api-with-django/>

**Queries** - Used for get, filtering data from data:

class Query(graphene.ObjectType):

links = graphene.List(LinkType)

def **resolve\_links**(self, info, \*\*kwargs):

return Link.objects.all()

It can be hitted by:  
 **query {**

**links {**

**id**

**description**

**url**

**}**

**}**

This query is a just used for listing all the rows from link table  
Make a variable links which is used as a query variable  
Then resolve it as resolve\_variable\_name - resolve\_links

**Mutation** - Used for creating, updating, deleting data from table

class CreateLink(graphene.Mutation):

id = graphene.Int()

url = graphene.String()

description = graphene.String()

*#2*

class Arguments:

url = graphene.String()

description = graphene.String()

*#3*

def **mutate**(self, info, url, description):

link = Link(url=url, description=description)

link.save()

return CreateLink(

id=link.id,

url=link.url,

description=link.description,

)

*#4*

class Mutation(graphene.ObjectType):

create\_link = CreateLink.Field()

1: Defines a mutation class. Right after, you define the output of the mutation, the data the server can send back to the client. The output is defined field by field for learning purposes. In the next mutation you’ll define them as just one.

2: Defines the data you can send to the server, in this case, the links’ url and description.

3: The mutation method: it creates a link in the database using the data sent by the user, through the url and description parameters. After, the server returns the CreateLink class with the data just created. See how this matches the parameters set on #1.

4: Creates a mutation class with a field to be resolved, which points to our mutation defined before.

**All this should be done in a app schema level**

**Pagination**:

The simple way defined in the GraphQL [pagination](http://graphql.org/learn/pagination/) documentation is to slice the results using two parameters: first, which returns the first *n* items and skip, which skips the first *n* items.

Ex:

*# Use them to slice the Django queryset*

def **resolve\_links**(self, info, search=None, first=None, skip=None, \*\*kwargs):

qs = Link.objects.all()

if search:

filter = (

Q(url\_\_icontains=search) |

Q(description\_\_icontains=search)

)

qs = qs.filter(filter)

if skip:

qs = qs[skip:]

if first:

qs = qs[:first]

return qs

def **resolve\_votes**(self, info, \*\*kwargs):

return Vote.objects.all()

You just need to specify the first and last named arg to perform pagination

**Filtering:**

You already can list all links, but another feature is to search them, by URL or description. In GraphQL, this concept is the same as mutations: you pass an argument to the links field, used by the resolver to filter the results

Ex:

from django.db.models import Q

class Query(graphene.ObjectType):

links = graphene.List(LinkType, search=graphene.String())

votes = graphene.List(VoteType)

def **resolve\_links**(self, info, search=None, \*\*kwargs):

if search:

filter = (

Q(url\_\_icontains=search) |

Q(description\_\_icontains=search)

)

return Link.objects.filter(filter)

return Link.objects.all()

def **resolve\_votes**(self, info, \*\*kwargs):

return Vote.objects.all()

**djongo library we will use to connect mongodb with django.**

pip install djongo

Graphene(class level) methods -:

types of variables -:

-: instance variable

it should be declared inside constructor,and will varry objects to objects.

-: static variable

it is a class level variable which is same for all objects.

-: local variable

Variables which is declared inside a constructor or specific method is call local variables.

types of method by using decorater -:

-: instance method

It is also known as object related method.In this method self must be required.We are using self to access data from constructors.

-: class method

For class level method @classmethod decorater is used for identifying it is a class level method.Ist argument must be cls else it will make difference in exact value or treat variable as local varibale if not foun will returns an error.

-: static method

@staticmethod decorator is used for identifying weather it is staticmethod or any other but this is not an required decorator without this function will automaticall treat as staticmethod.

Graphql -:

graphql is new api standered that provides an more efficient,powerful and flexible alternative to rest.

It was developed by open source facebook.

In this, client can specify that exactly what data it needs from an api.Instead of multiple endpoints that return fixed datastruture.

With GraphQL, you can send a query to your API and get exactly what you need, nothing more and nothing less.

Modern applications are now built in comprehensive ways where a single backend application supplies the data that is needed to run multiple clients.

Field types -:

Int: A signed 32‐bit integer.

Float:

String: A UTF‐8 character sequence.

Boolean: true or false.

ID: The ID scalar type represents a unique identifier.