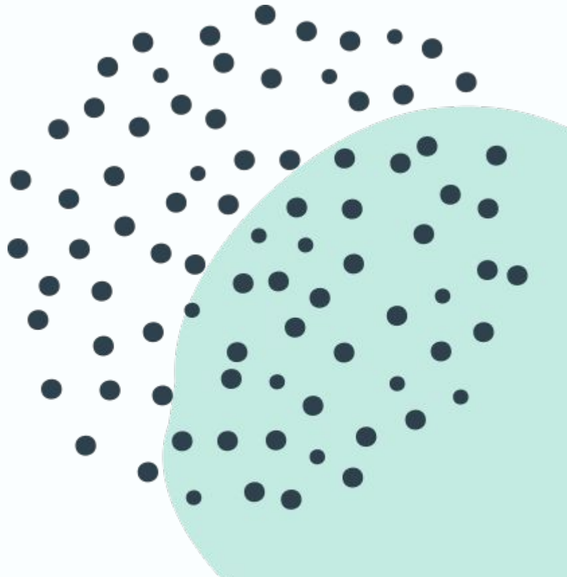
Developing GraphQL APIs in Django using Graphene

By Nisarg Shah



Nisarg Shah

Crazy Developer

HELLO!

Undergrad CS Student

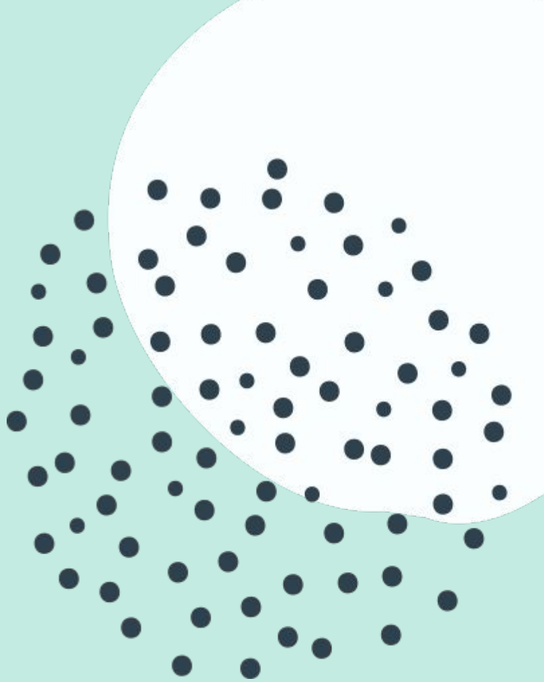
Software Developer at Tweetozy

Co-creator of CoursesAround

CONNECT WITH ME!

iamnisarg.in
nisarg1499
nisargshah14
nisshah1499@gmail.com

Today's Talk



MAIN POINTS

General View on API

REST APIs and GraphQL APIs

Understand GraphQL

Implementation using Graphene

Building Web Applications

- Most web applications use APIs in their backend and build their interface upon that.
- Complete business logic in one place

SOME POPULAR API PROTOCOLS

- SOAP
- REST
- and many more.....

RESTful APIs

- Endpoints

GET https /{id}/getProfile

PUT https /{id}/talkTitle

POST https /{id}/newProfile

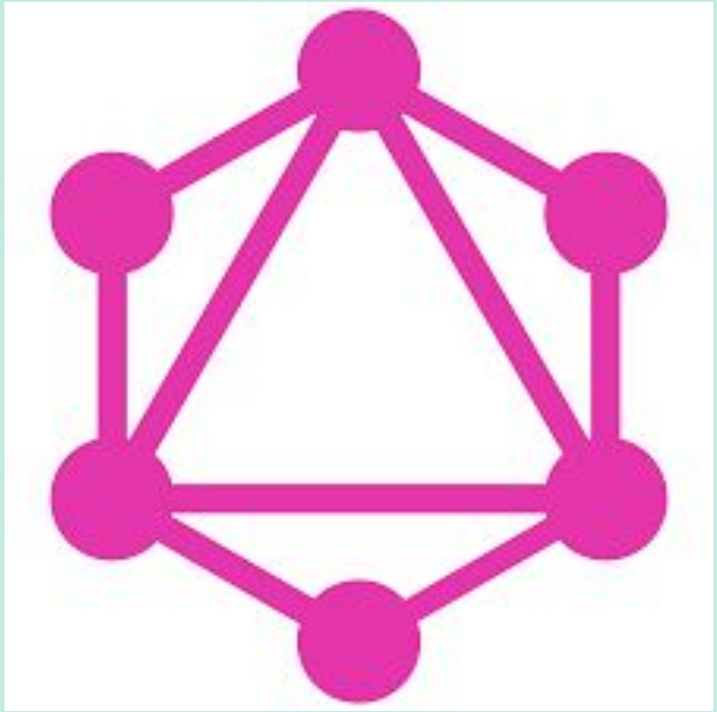
DELETE https /{profileId}

- A piece of code is executed when these APIs are called.
- Server returns the response to client

Problems faced in RESTful APIs

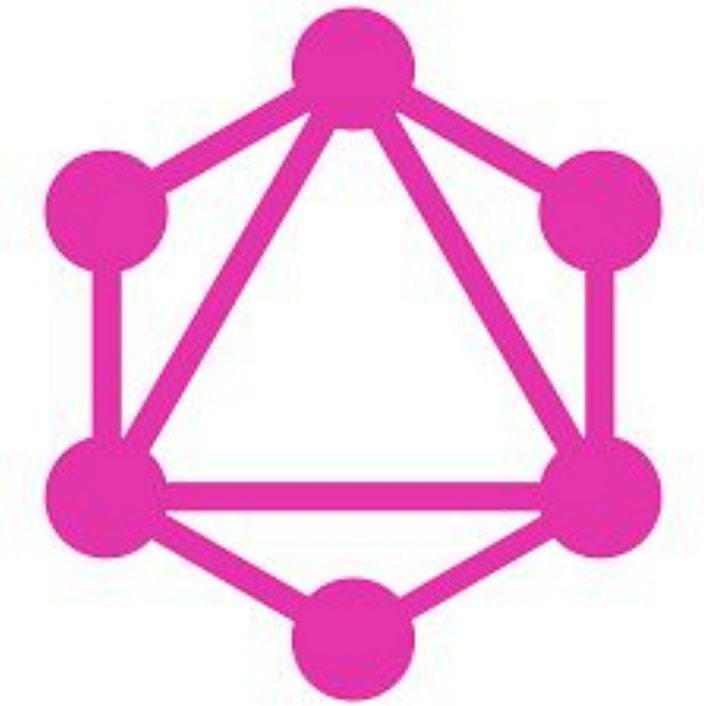
- Multiple Endpoints
- Over Fetching
- Under Fetching

What can be the alternative?



GraphQL

GraphQL



- Open Source
- GraphQL is a Query Language
- Uses Schema based system
- Easy and efficient to use

Why GraphQL?

- Client requests the needed data. Client decides the query and according to that data is fetched.

And you know what...

- Only one API Endpoint
- No over fetching or under fetching
- Auto-generation of API documentation

Let's learn about GraphQL

- Schema : structure
- Mutation : updating data on server
- Queries : fetching data
- Subscriptions : real time data exchange

Schema

```
type Product {
  productId :ID!,
  productName : String!,
  productCategory :String!,
  productPrice : Float!,
  productDiscountPrice : Float,
  productPreviewDesc : String,
  productFullDesc :String,
  orderproductSet : [OrderProductType]
}
```

Product Schema

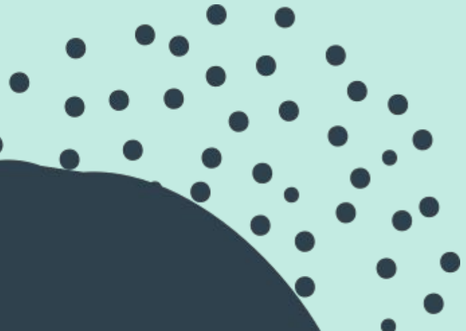
- GraphQL Object Type
  - Product
- Fields
  - productId
  - productName
  - and few listed in pic
- Scalar Types
  - Int
  - String
  - and many more...

Mutation

```
mutation{
  addProduct(productName: "Keychain", productPrice: 40,
    productFullDesc:"This is a Teddy keychain", productCategory:"Others",
    productPreviewDesc:"", productDiscountPrice: 0){

    addProduct{
      productId,
      productName,
      productPrice,
      productCategory,
      productFullDesc,
      productPreviewDesc,
      productDiscountPrice
    }
  }
}
```

- Used for changing data on server
- Return the response according to your needs
- Variables passed can be scalars or ObjectTypes

Response from server →



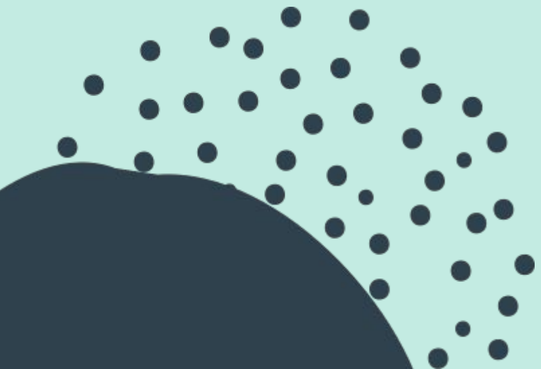
```
{
  "data": {
    "addProduct": {
      "addProduct": {
        "productId": "1",
        "productName": "Keychain",
        "productPrice": 40,
        "productCategory": "Others",
        "productFullDesc": "This is a Teddy keychain",
        "productPreviewDesc": null,
        "productDiscountPrice": null
      }
    }
  }
}
```

Using query variables for inserting data

```
mutation($productName: String!, $productPrice: Float!){
  addProduct(productName: $productName, productPrice: $productPrice,
    productFullDesc:"Cool shoes", productCategory:"Footwear",
    productPreviewDesc:"No warranty", productDiscountPrice: 0){

    addProduct{
      productId,
      productName,
      productPrice,
      productCategory,
      productFullDesc,
      productPreviewDesc,
      productDiscountPrice
    }
  }
}
```

QUERY VARIABLES

```
{
  "productName": "XYZ Shoes",
  "productPrice": 10000
}
```



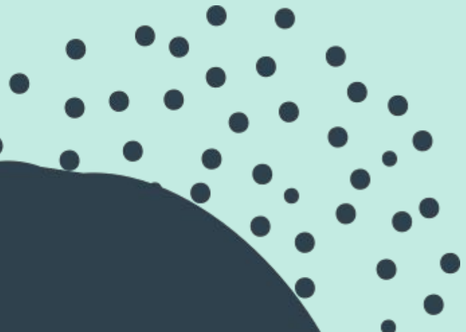
```
{
  "data": {
    "addProduct": {
      "addProduct": {
        "productId": "4",
        "productName": "XYZ Shoes",
        "productPrice": 10000,
        "productCategory": "Footwear",
        "productFullDesc": "Cool shoes",
        "productPreviewDesc": null,
        "productDiscountPrice": null
      }
    }
  }
}
```

Query

```
query{
  products(productName: "Keychain", first: 5, jump: 0){
    productId,
    productName,
    productPrice,
    productCategory,
    productFullDesc
  }
}
```

- Get data from server
- Ask for specific fields on objects
- Design query according to needs

Response from server



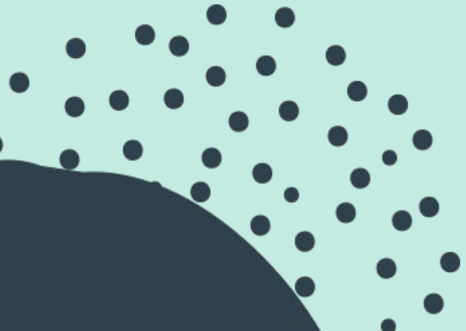

```
{
  "data": {
    "products": [
      {
        "productId": "1",
        "productName": "Keychain",
        "productPrice": 40,
        "productCategory": "Others",
        "productFullDesc": "This is a Teddy keychain"
      },
      {
        "productId": "2",
        "productName": "Keychain-Panda",
        "productPrice": 30,
        "productCategory": "Others",
        "productFullDesc": "This is a Panda keychain"
      }
    ]
  }
}
```

Subscriptions

```
subscription{
    addProduct{
        productName,
        productPrice,
        productDesc
    }
}
```

Response from server 

```
{
    "addProduct" : {
        "productName" : " Toy",
        "productPrice" : 500,
        "productDesc" : "New toy"
    }
}
```



- Realtime connection to server
- Client subscribes to an event
- Server pushes data to client when event occurs
- Same syntax as queries and mutations

Libraries for building GraphQL APIs in Python

GRAPHENE

5.6k+ Stars

Code-First Approach

STRAWBERRY

456 Stars

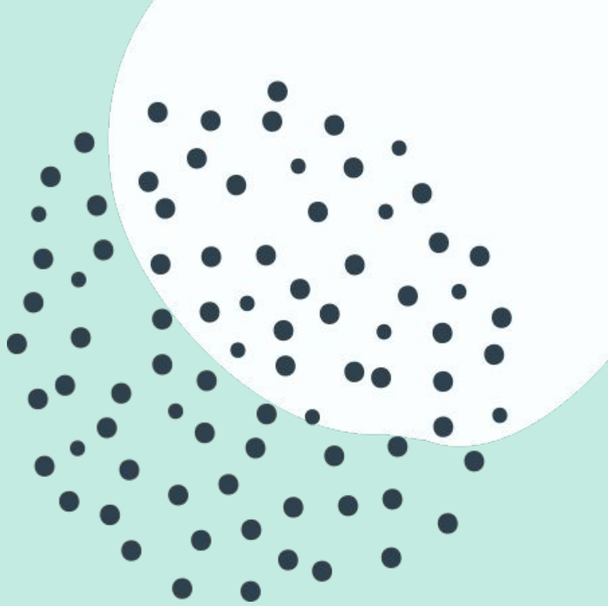
Code-First Approach

ARIADNE

896 Stars

Schema-First Approach

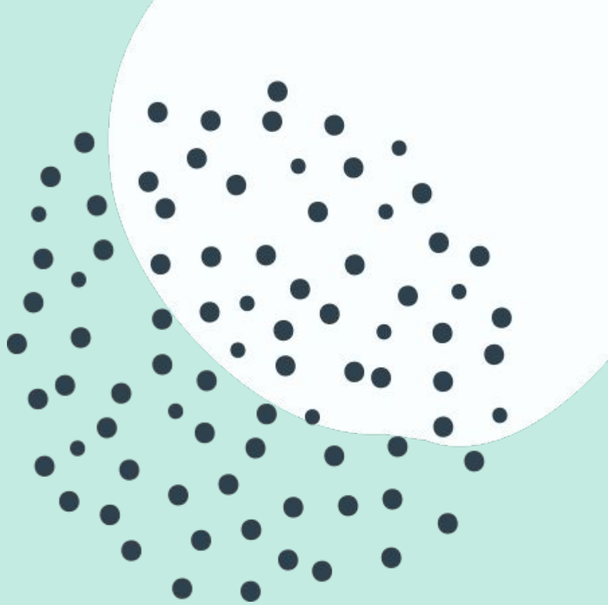
Let's Build GraphQL APIs



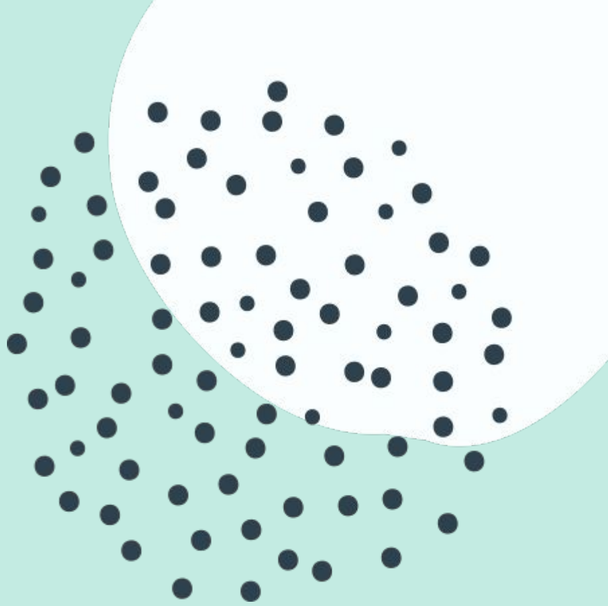
We will use

- Django
- Graphene
- PostgreSQL

Environment setup



- Create a virtual environment
  - python3 -m venv venv
- Setup a django project
  - pip3 install django
  - django-admin startproject project
  - cd project
  - python3 manage.py makemigrations
  - python3 manage.py migrate



- Install graphene
  - pip3 install graphene
- Install graphene-django(provides DjangoObjectTypes)
  - pip3 install graphene-django
- Change settings.py to setup postgresql

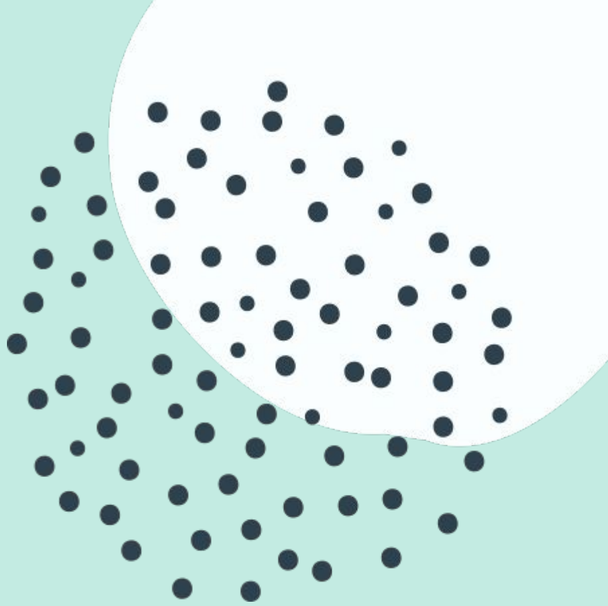
Necessary changes in settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ecommerce',
    'graphene_django'
]
```

Add the following in your settings.py file

```
GRAPHENE = {
    'SCHEMA' : 'project.schema.schema'
}
```

Few Concepts of Graphene



- ObjectType
- Schema
- Resolvers
- Scalars

```python
import graphene
from graphene_django import DjangoObjectType
from django.db.models import Q
from graphene import ObjectType

class ProductType(DjangoObjectType):
    class Meta:
        model = Product

class Query(graphene.ObjectType):
    products = graphene.List(ProductType, productName = graphene.String())

    def resolve_products(self, info, productName, **kwargs):
        filter = (Q(productName__icontains = productName))
        return Product.objects.filter(filter)
```

ObjectType

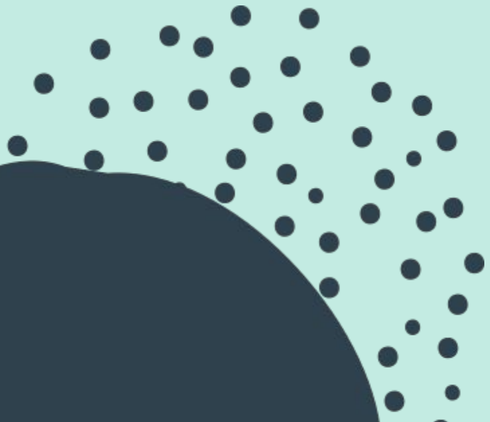
- A block which is used to define a relation between fields and schema

Schema

- Relationship between fields in API

```
class Product(graphene.ObjectType):
    productName = graphene.String()
    productPrice = graphene.Float()
    productCategory = graphene.String()
    productFullDesc = graphene.String(name='desc')
```

```
class ProductType(DjangoObjectType):
    class Meta:
        model = Product

class Query(graphene.ObjectType):
    products = graphene.List(ProductType)
```

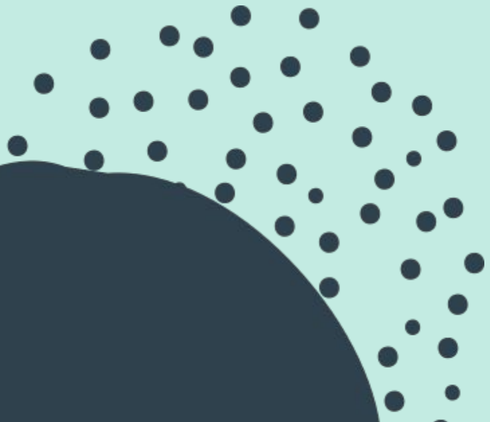


Resolvers

- A method that helps to answer queries

```python
class Query(graphene.ObjectType):
    login = graphene.Field(UserType, username = graphene.String(), password = graphene.String())
    users = graphene.List(UserType)

    def resolve_users(self, info):
        return get_user_model().objects.all()

    def resolve_login(self, info, username, password, **kwargs):
        auth_user = authenticate(username = username, password = password)

        if auth_user == None:
            raise Exception('Invalid Credentials')

        return auth_user
```

- login field is resolved by resolve_login method. Name should match
- The query string is executed and data is sent in query response
- Return any response to the frontend

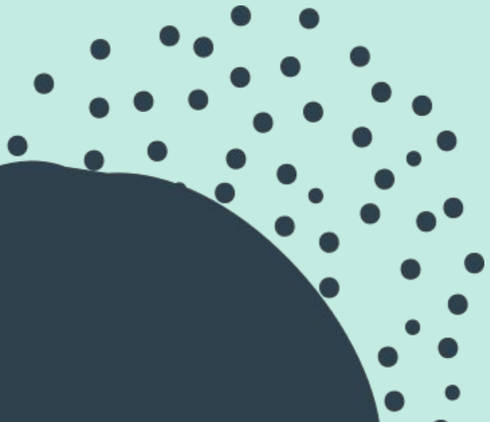


Resolvers

- Parameters in resolver methods : parent, info, **kwargs

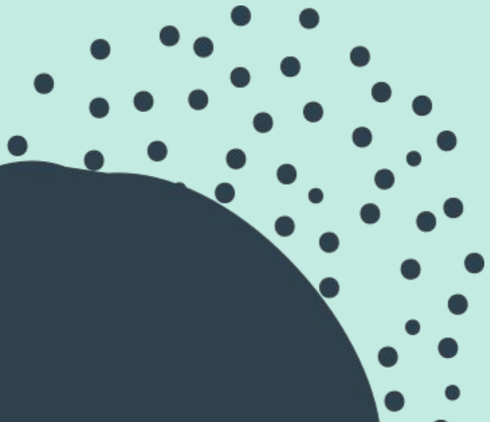
- parent : return the value of resolver for current parent's field (i.e. parent of current root)
- info : some meta information and context information
- **kwargs : graphql arguments (i.e. variables in query)

```
query($author: String!, $repoName: String!){
    repo(login: $author){
        repository(name: $repoName){
            forkCount,
            updatedAt
        }
    }
}
```



Scalars

- a.k.a : Data types



There are several scalar types which are in built in graphene. Some of them are :

- graphene.String
- graphene.Int
- graphene.Float
- graphene.ID
- graphene.DateTime
- and many more.....

You can also create your own customised scalar types according to requirements

QUERIES

Fetch the data from server

MUTATIONS

Update the data on server

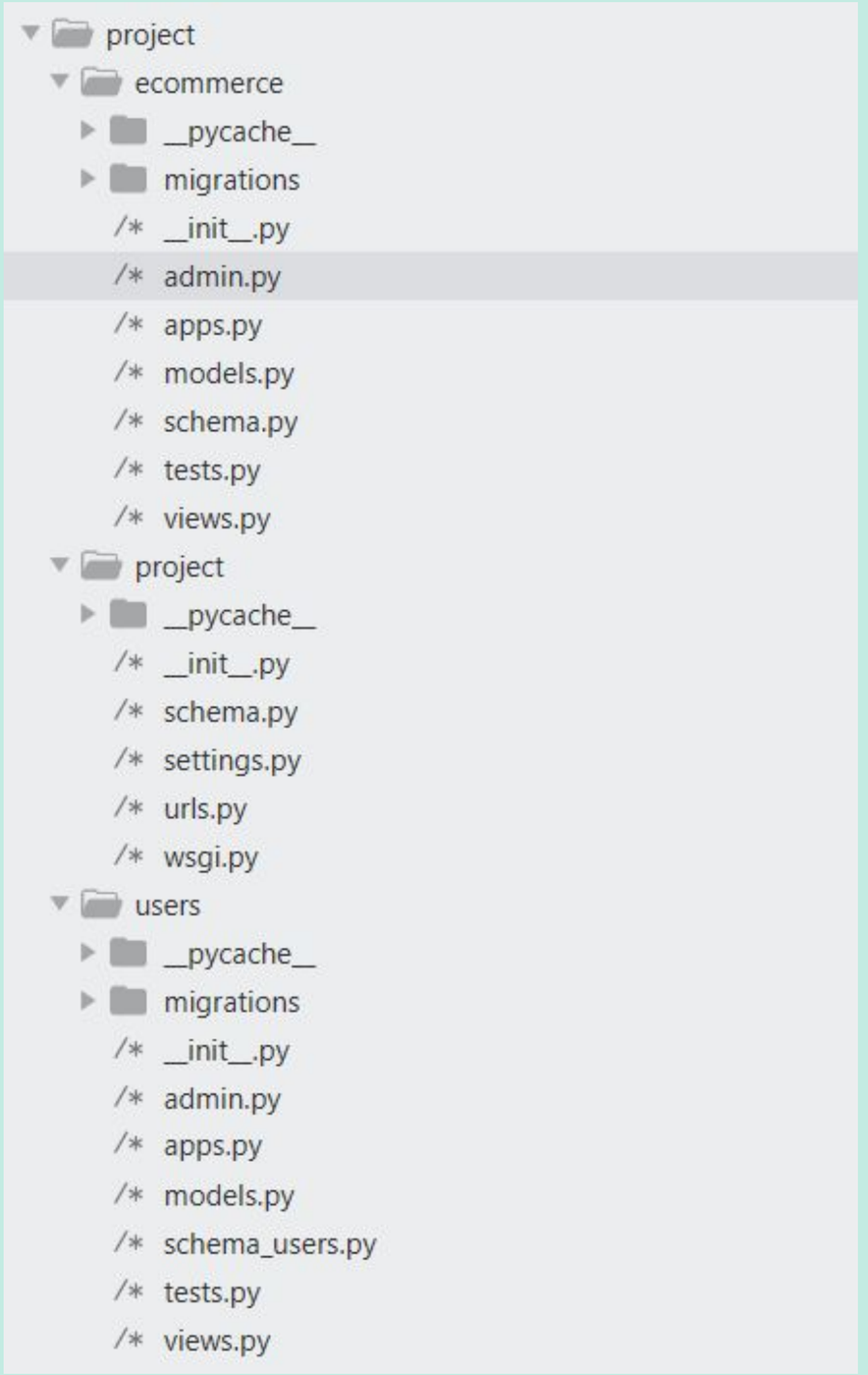
PAGINATION

Send a particular bunch of data instead of complete data

AUTHENTICATION

Secure your backend API's using JWT Tokens

File Structure



Writing Queries

```python
import graphene
from graphene_django import DjangoObjectType
from django.db.models import Q
from graphene import ObjectType

from .models import Product

class ProductType(DjangoObjectType):
    class Meta:
        model = Product


class Query(graphene.ObjectType):
    products = graphene.List(ProductType, product_name = graphene.String())

    def resolve_products(self, info, product_name, **kwargs):

        filter = (Q(product_name__icontains = product_name))
        return Product.objects.filter(filter)
```

Used concepts : ObjectType, DjangoObjectType, resolvers, Schema

Writing Mutations

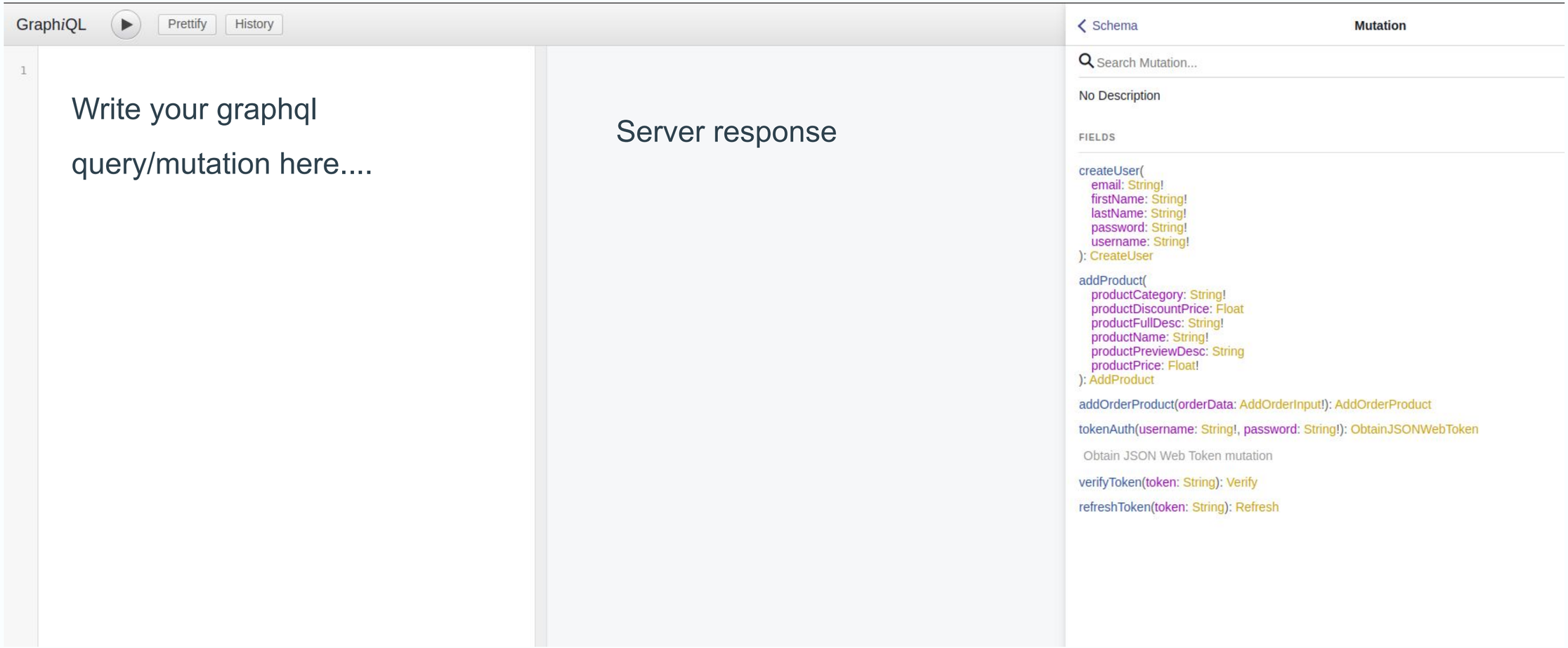
Used concepts : ObjectType, DjangoObjectType, arguments, mutate

```python
class ProductType(DjangoObjectType):
    class Meta:
        model = Product


class AddProduct(graphene.Mutation):
    addProduct = graphene.Field(ProductType)

    class Arguments:
        product_name = graphene.String(required=True)
        product_category = graphene.String(required=True)
        product_price = graphene.Float(required=True)
        product_discount_price = graphene.Float()
        product_preview_desc = graphene.String()
        product_full_desc = graphene.String(required=True)

    def mutate(self, info, product_name, product_category, product_price,
               product_discount_price, product_preview_desc, product_full_desc, **kwargs):
        product_discount_price = kwargs.get('product_discount_price', None)
        product_preview_desc = kwargs.get('product_preview_desc', None)

        product = Product(product_name=product_name,
                          product_category=product_category,
                          product_price=product_price,
                          product_discount_price=product_discount_price,
                          product_preview_desc=product_preview_desc,
                          product_full_desc=product_full_desc)

        product.save()

        return AddProduct(addProduct=product)
```

GraphiQL View

Add this in your urls.py file

path('graphql/', csrf_exempt(GraphQLView.as_view(graphiql=True)))



Pagination

Used concepts :

ObjectType,

DjangoObjectType,

Python slicing

```python
class ProductType(DjangoObjectType):
    class Meta:
        model = Product


class Query(graphene.ObjectType):
    products = graphene.List(ProductType, product_name = graphene.String(),
        first = graphene.Int(), jump = graphene.Int())

    def resolve_products(self, info, product_name, first=None, jump=None, **kwargs):

        all_products = Product.objects.all()
        if product_name:
            filter = (Q(product_name__icontains = product_name))
            filtered = all_products.filter(filter)

            if jump:
                filtered = filtered[jump:]
            if first:
                filtered = filtered[:first]

        return filtered
```

SENDING A BUNCH OF DATA

Authentication

```
GRAPHENE = {
    'SCHEMA': 'project.schema.schema',
    'MIDDLEWARE': [
        'graphql_jwt.middleware.JSONWebTokenMiddleware',
    ],
}

AUTHENTICATION_BACKENDS = [
    'graphql_jwt.backends.JSONWebTokenBackend',
    'django.contrib.auth.backends.ModelBackend',
]
```

settings.py

.project/schema.py

```
import graphene
import graphql_jwt

import ecommerce.schema
import users.schema_users

class Query(ecommerce.schema.Query,
    users.schema_users.Query,
    graphene.ObjectType):
    pass

class Mutation(ecommerce.schema.Mutation,
    users.schema_users.Mutation,
    graphene.ObjectType):
    token_auth = graphql_jwt.ObtainJSONWebToken.Field()
    verify_token = graphql_jwt.Verify.Field()
    refresh_token = graphql_jwt.Refresh.Field()

schema = graphene.Schema(query=Query, mutation=Mutation)
```

- Created a mutation for creating a user
- Create a user

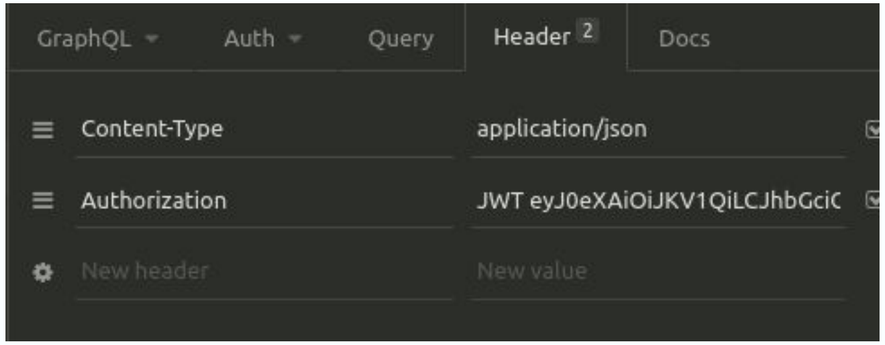
```
mutation{
  tokenAuth(username: "nisarg", password: "nisarg"){
    token,
    refreshExpiresIn
  }
}
```


```
{
  "data": {
    "tokenAuth": {
      "token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Im5pc2FyZyIsImV4cCI6MTU5NTE1NzA4
MCwib3JpZ0lhdCI6MTU5NTE1Njc4MH0.Kl67-wTewyWo7gD1RyTGg4xFuLej5-1QAnzLPljI0Ek",
      "refreshExpiresIn": 1595761580
    }
  }
}
```

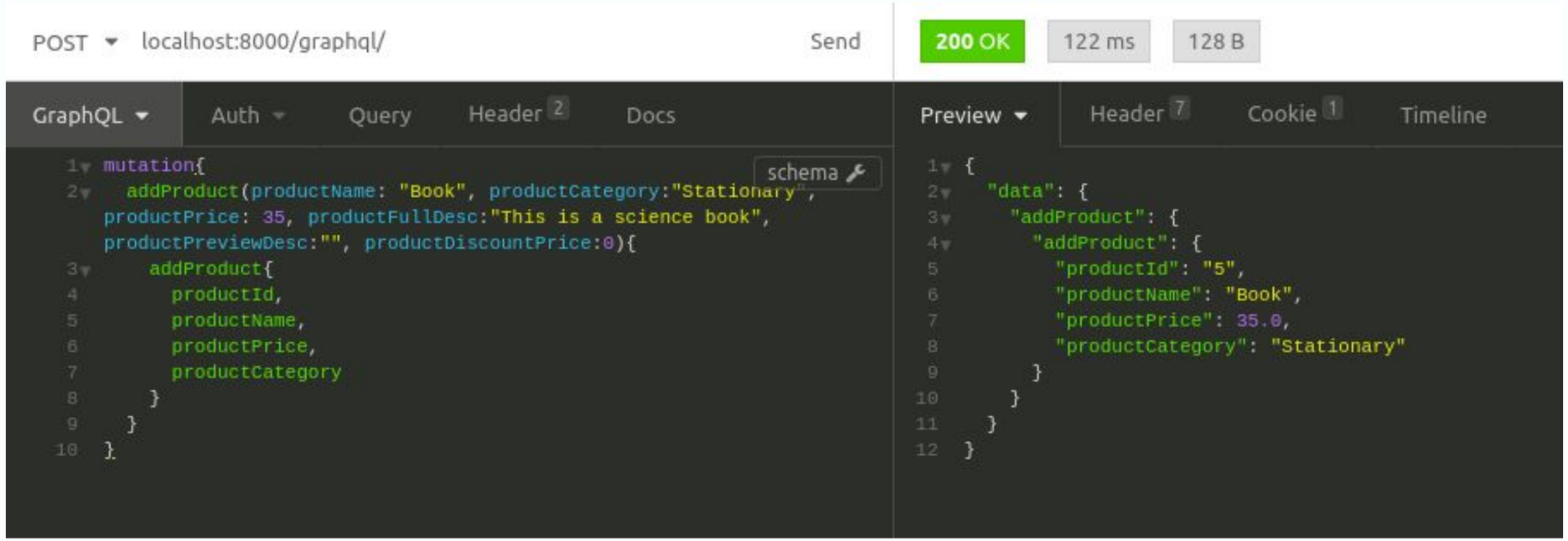

- Use this mutation while login
- Store this token and use for further queries.

```
user = info.context.user
if user.is_anonymous:
    raise Exception("Not logged in!!")
```

Add this is your queries and mutations



Add JWT Token in Headers prefixed by "JWT"



POST localhost:8000/graphql/ Send

200 OK 3.6 ms 127 B

GraphQL Auth Query Header 1 Docs

```
mutation{
  addProduct(productName: "Book", productCategory:"Stationary",
productPrice: 35, productFullDesc:"This is a science book",
productPreviewDesc:"", productDiscountPrice:0){
    addProduct{
      productId,
      productName,
      productPrice,
      productCategory
    }
  }
}
```

schema

Preview Header 7 Cookie 1 Timeline

```
{
  "errors": [
    {
      "message": "Not logged in!!",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "addProduct"
      ]
    }
  ],
  "data": {
    "addProduct": null
  }
}
```

Mutation without JWT Token

Helpful Resources

- GraphQL Website : https://graphql.org/
- GraphQL Blogs : Medium
- HowToGraphQL : https://www.howtographql.com/
- Graphene Documentation

Open Source Project

Github Repository :

https://github.com/nisarg1499/django-ecommerce-graphql

Currently 3 active contributors

Building boiler plate of ecommerce by implementing GraphQL APIs in django

FINAL WORDS

THANK YOU

CONNECT WITH ME!

iamnisarg.in
nisarg1499
nisargshah14
nisshah1499@gmail.com

- NISARG SHAH