Django i GraphQL

https://przemyslawjanpietrzak.github.io/przemyslawjanpietrzak.github.io/django-graphql/

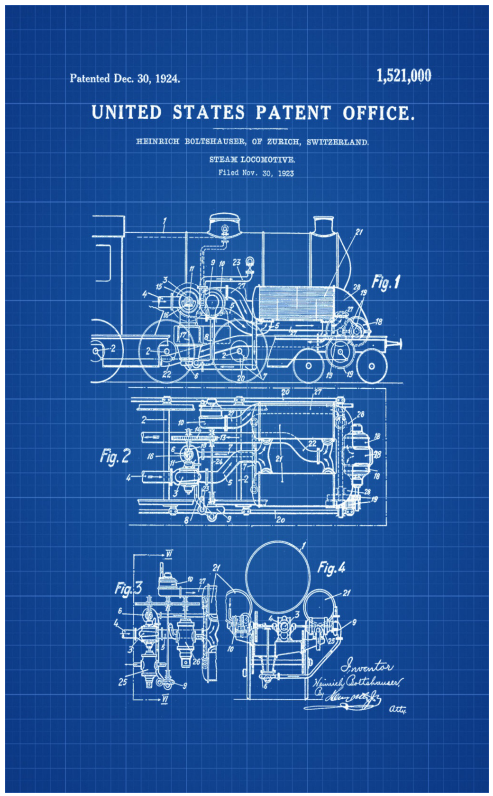
https://github.com/przemyslawjanpietrzak/pyMonet

https://twitter.com/przemyslawjanp

REST

```
async def request(url: URL, method: Methods, payload=None, **queryParams):
  db_side_effect(payload)
  return response
```

Backend & Frontend



Patented Dec. 30, 1924. 1,521,000

UNITED STATES PATENT OFFICE.

HEINRICH BOLTSHAUSER, OF ZURICH, SWITZERLAND.

STEAM LOCOMOTIVE.

Filed Nov. 30, 1923

Fig. 1

Fig. 2

Fig. 3

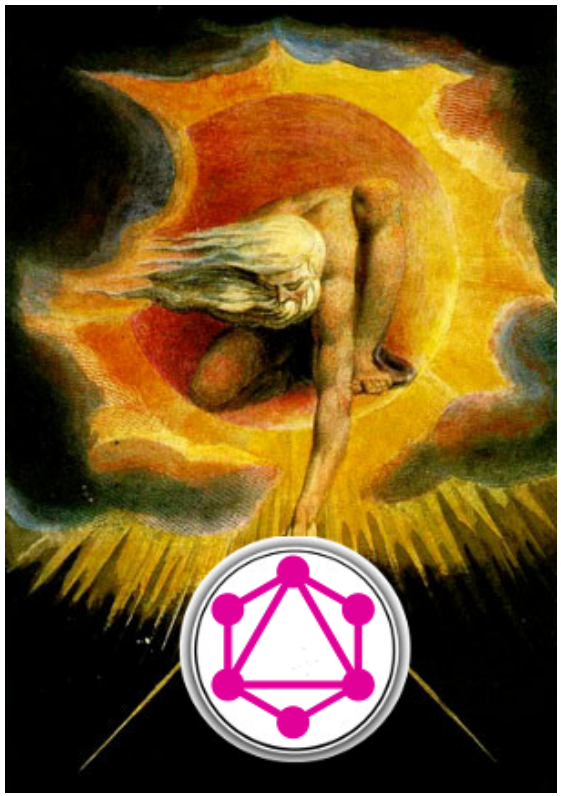
Fig. 4

Inventor

Heinrich Boltshauser

The Dev Estimate - Munch, Edvard (1887)

Graph Query Language



Query Syntax

```
{
  actor {
    firstName
    lastName
    birthDate
  }
}
```

```
{
  "actor": {
    "firstName": "Charles",
    "lastName": "Chaplin",
    "birthDate": "16.04.1889"
  }
}
```

Query Syntax #2

```
{
  actor {
    firstName
    birthDate
    growth
  }
}
```

```
{
  "actor": {
    "firstName": "Charles",
    "birthDate": "16.04.1889",
    "growth": 165
  }
}
```

Query Syntax #3

```
{
  actor {
    firstName
    lastName
    movies {
      name
      rating
    }
  }
}
```

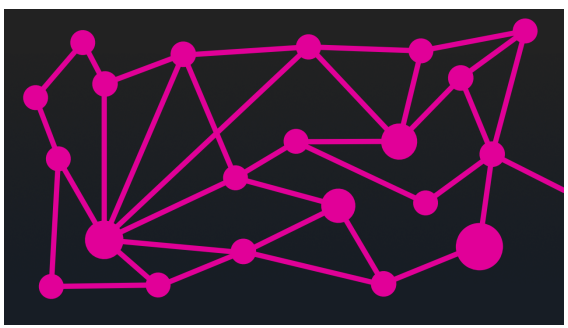
```
{
  "actor": {
    "firstName": "Charles",
    "lastName": "Chaplin",
    "movies": [
      { "name": "The Great Dictator", rating: 8 },
      { "name": "Monsieur Verdoux", rating: 7.7 }
    ]
  }
}
```

Query Syntax #4

```
{
  actor {
    firstName
    lastName
    movies(name: "Limelight") {
      name
      director {
        firstName
        lastName
      }
    }
  }
}
```

```
{
  "actor": {
    "firstName": "Charles",
    "lastName": "Chaplin",
    "movies": [
      { "name": "Limelight", "director": {
        "firstName": "Charles",
        "lastName": "Chaplin"
      } },
    ]
  }
}
```

Static typing



```
type Movie {
  name: String
  rating: Float
  director: Director
}

type Actor {
  firstName: String
  lastName: String
  birthDate: Date
  movies: [Movie]
}
```

Mutation

```
mutation UpdateActor($firstName: String!, $lastName: String!) {
  actor {
    firstName
    lastName
  }
}
```

Lets do it

```
pip install graphene graphene-django
```

```
INSTALLED_APPS = (
    # ...
    'graphene_django',
)
```

```
from django.conf.urls import url
from graphene_django.views import GraphQLView

urlpatterns = [
    # ...
    url(r'^graphql', GraphQLView.as_view(graphiql=True, schema=schema)),
]
```

Make graph great again

```
from graphene_django import DjangoObjectType
import graphene

class ActorNode(DjangoObjectType):
    class Meta:
        model = ActorModel

class Query(graphene.ObjectType):
    actors = graphene.List(ActorNode)
    actor = graphene.Field(ActorNode)

    def resolve_actors(self, info):
        return ActorModel.objects.all()

    def resolve_actor(self, info, **kwargs):
        id = kwargs.get('id')

        return ActorModel.objects.get(pk=id)

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

Access

```
from graphene_django import DjangoObjectType
import graphene

class Actor(DjangoObjectType):
    class Meta:
        model = ActorModel
        only_fields = ('firstName', 'lastName')
        # exclude_fields = ('pk',)
        interfaces = (relay.Node, )


      @classmethod
      def get_node(cls, id, info):
        if not info.context.user.is_authenticated():
          return None
        try:
            actor = cls._meta.model.objects.get(id=id)
        except cls._meta.model.DoesNotExist:
            return None
```

Custom nodes

```
import graphene

class Query(graphene.ObjectType):
    hello = graphene.String(name=graphene.String(default_value="stranger"))

    def resolve_hello(self, info, name):
        return 'Hello ' + name
```

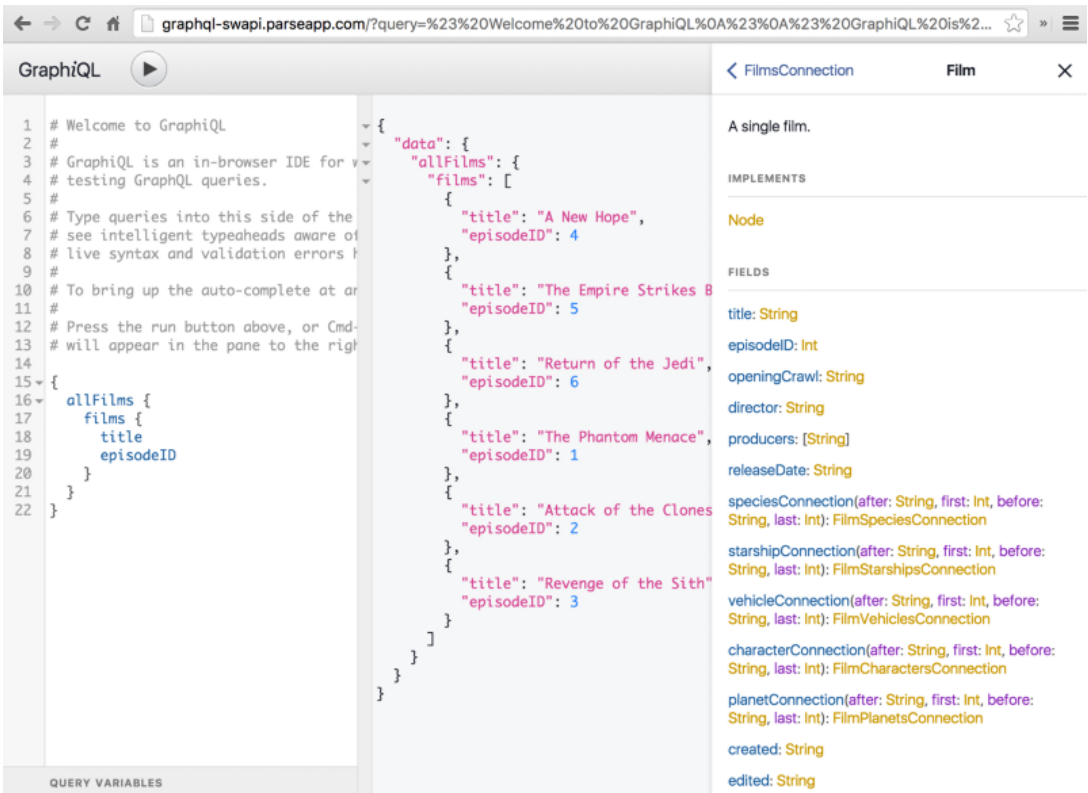
Filter

```
class ActorNode(DjangoObjectType):
    class Meta:
        # Assume you have an Actor model defined with the following fields
        model = Actor
        filter_fields = ['firstName', 'lastName', 'birthDate']
        interfaces = (relay.Node, )


class ActorFilter(django_filters.FilterSet):
    name = django_filters.CharFilter(lookup_expr=['iexact'])

    class Meta:
        model = Actor
        fields = ['firstName', 'lastName', 'birthDate']


class Query(ObjectType):
    actor = relay.Node.Field(ActorNode)
    all_actors = DjangoFilterConnectionField(ActorNode, filterset_class=ActorFilter)
```

Mutation node

```
from graphene_django.rest_framework.mutation import SerializerMutation

class ActorAwesomeMutation(SerializerMutation):
    class Meta:
        serializer_class = ActorSerializer
```

GraphiQL - docs + postman



graphQL schema + Client type system

```
./manage.py graphql_schema # generate graphQL schema in schema.json

 npx gql2ts schema.json -o data.d.ts # via gql2ts package generate TypeScript interfaces from schema.json
```

Try quicktype to generate interfaces in other languages (e.g. swift, kotlin)

Why?

- Many changes
- Elastic API
- len(frontend_power) > len(backend_power)
- Squash similar endpoint
- Reduce http calls
- CVDD

Why not?

- GraphQL for GraphQL
- len(frontend_power) < len(backend_power)
- Native cache
- Deadlines

Links

- GraphQL
- Graphene
- Graphene-Django
- pyMonet
- quicktype
- graphiql

Thank you :*