DRF vs Graphene

@jonatasbaldin

ou

REST vs GraphQL

@jonatasbaldin

DRF = Django REST Framework

Graphene = GraphQL para Python

O que é REST?

O que é REST?

Um estilo arquitetural que consiste em um conjunto de restrições dentro de um sistema distribuído.

REST Restrições

Cliente-Servidor
Stateless
Cacheável
Interface Uniforme
Sistema em Camadas
Código em Demanda (opcional)

REST + HTTP = Web APIs

Recursos
Endpoints
Métodos
Códigos de Status
Cabeçalhos

http://localhost:8000/

No Environment

GET | http://localhost:8000/links/ | Params | Send | Save

Status: 200 OK Time: 17 ms Size: 463 B

Auth | Headers (1) | Body | Pre-req. | Tests | Cookies | Code

Body | Cookies (1) | Headers (7) | Tests

Key	Value	Description	Bulk Edit	Presets
Content-Type	application/json			
New key	Value	Description		

Pretty | Raw | Preview | JSON


```
[
    {
        "id": 1,
        "url": "http://twitter.com",
        "description": "My Twitter",
        "posted_by": 5
    },
    {
        "id": 2,
        "url": "http://twitter.com",
        "description": "My Twitter",
        "posted_by": 5
    },
    {
        "id": 3,
        "url": "http://twitter.com",
        "description": "My Twitter",
        "posted_by": 6
    }
]
```

HATEOAS: O Santo Graal do REST

http://localhost:8000/

No Environment

GET | http://localhost:8000/links/ | Params | Send | Save

Status: 200 OK Time: 39 ms Size: 556 B

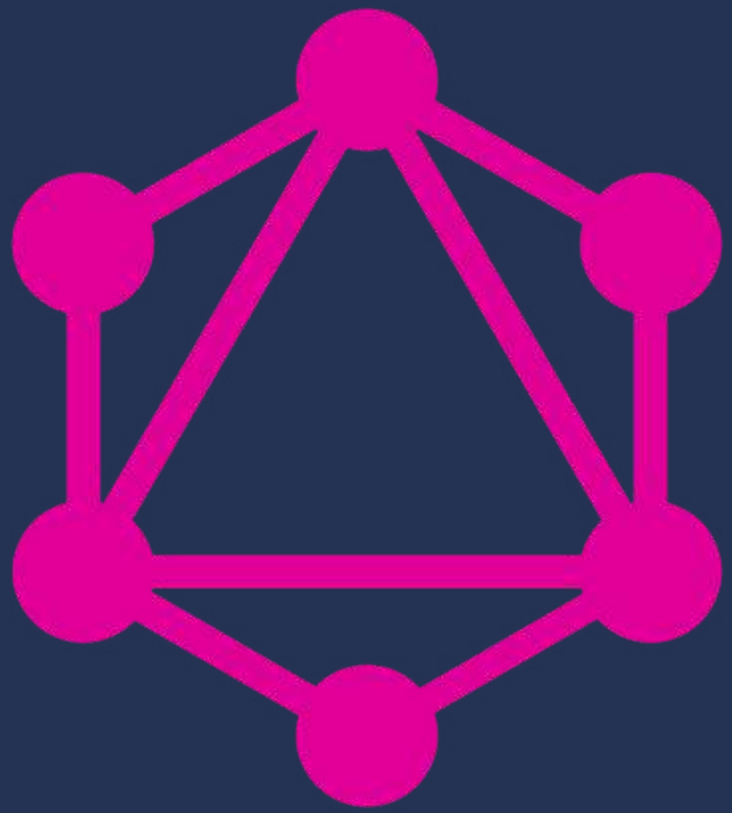
Auth | Headers (1) | Body | Pre-req. | Tests | Cookies | Code

	Key	Value	Description	•••	Bulk Edit	Presets
☑	Content-Type	application/json				
	New key	Value	Description			

Body | Cookies (1) | Headers (7) | Tests

Pretty | Raw | Preview | JSON


```
[
    {
        "id": 1,
        "url": "http://twitter.com",
        "description": "My Twitter",
        "posted_by": "http://localhost:8000/users/5/"
    },
    {
        "id": 2,
        "url": "http://twitter.com",
        "description": "My Twitter",
        "posted_by": "http://localhost:8000/users/5/"
    },
    {
        "id": 3,
        "url": "http://twitter.com",
        "description": "My Twitter",
        "posted_by": "http://localhost:8000/users/6/"
    }
]
```



GraphQL

O que é GraphQL?

O que é GraphQL?

GraphQL é uma *query language*, uma especificação e um conjunto de ferramentas utilizada em um único *endpoint*, otimizada para performance e flexibilidade.

Conceitos Gerais

Schema Definition Language (DSL)
Fields e *Resolvers*
Único *endpoint*
Queries e *Mutations*
Subscriptions – não tem no Python :(
Introspecção

Schema Definition Language (SDL)

```
type Link {
  id: Int
  url: String!
  description: String
}
```

Linguagem com forte tipagem

Possui campos – *fields* – de um tipo específico

Podemos criar um *type* com uma coleção de *fields*

Cada *field* é processado por um *resolver*

Executando *queries*

```
query {
  links {
    id
    url
    description
  }
}
```

```
{
  "data": {
    "links": [
      {
        "id": "1",
        "url": "http://twitter.com",
        "description": "My Twitter"
      },
      {
        "id": "2",
        "url": "http://github.com/jonatasbaldin",
        "description": "My Github"
      }
    ]
  }
}
```

Podemos escolher os *fields* que queremos

```
query {
  links {
    id
  }
}
```

```
{
  "data": {
    "links": [
      {
        "id": "1"
      },
      {
        "id": "2"
      }
    ]
  }
}
```

Executando *queries* com argumentos

```
query {
  links(id: 1) {
    id
    url
    description
  }
}
```

```
{
  "data": {
    "links": [
      {
        "id": "1",
        "url": "http://twitter.com",
        "description": "My Twitter"
      }
    ]
  }
}
```

Enviando dados para o *backend*

```
mutation {
  createLink(url: "http://deployeveryday.com", description: "My blog") {
    id
    url
    description
  }
}
```

```
{
  "data": {
    "createLink": {
      "id": 6,
      "url": "http://deployeveryday.com",
      "description": "My blog"
    }
  }
}
```

Construindo o *Schema* completo

```
type Link {
  id: Int
  url: String!
  description: String
}

type Query {
  links(id: Int): [Link!]
}

type Mutation {
  createLink(url: String!, description: String): Link!
}

schema {
  query: Query
  mutation: Mutation
}
```

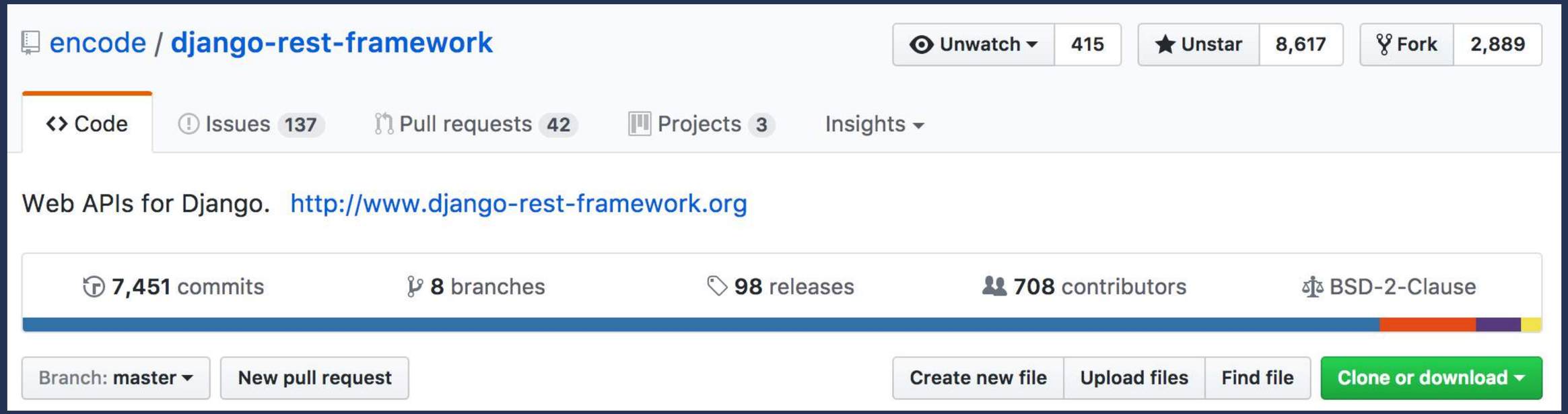
Falar é barato
Mostre me o código


django
REST
framework 3

Django REST Framework

Fortemente integrado com o ORM do Django
Altamente customizável
Documentação extensa
Web browsable API
Usado por uma galera

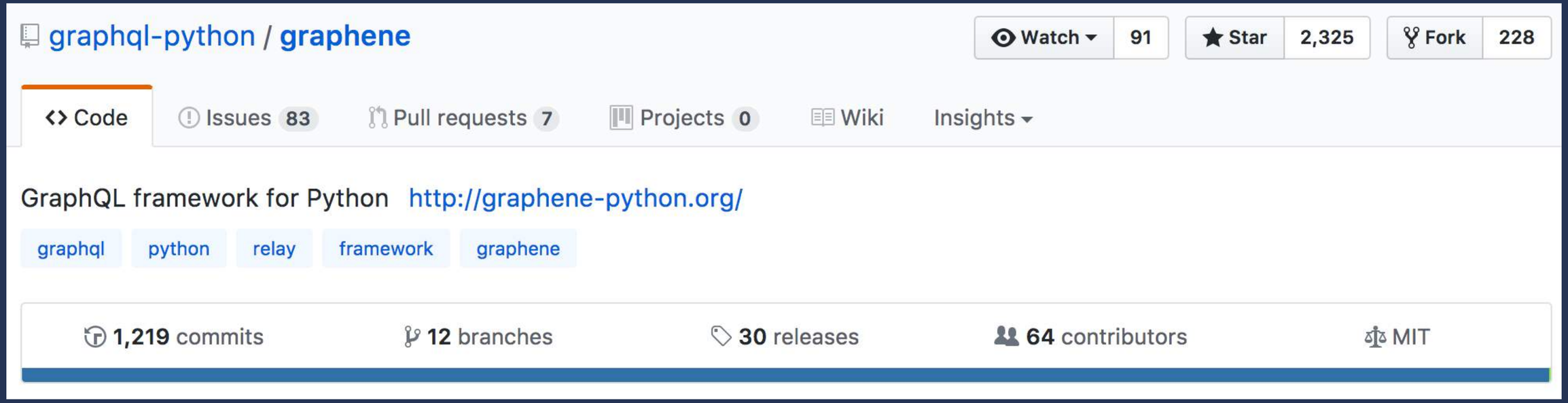
Django REST Framework



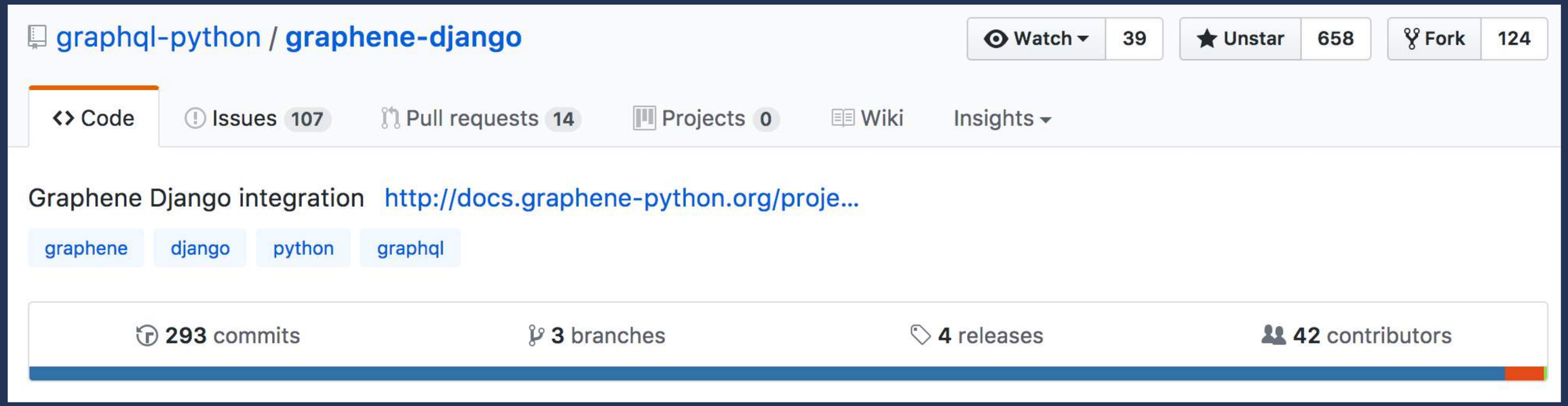
Graphene e Graphene-Django

Graphene é a implementação de GraphQL para Python
Graphene-Django integra com os padrões Django
Possui integração com outros projetos
Possui integração com o *Relay*
Projeto novo

Graphene


graphql-python / graphene
Watch 91
Star 2,325
Fork 228
Code
Issues 83
Pull requests 7
Projects 0
Wiki
Insights
GraphQL framework for Python http://graphene-python.org/
graphql
python
relay
framework
graphene
1,219 commits
12 branches
30 releases
64 contributors
MIT

Graphene-Django



Finalmente, DRF vs Graphene

Uso Básico

DRF

```
# views.py
class LinkViewSet(viewsets.ModelViewSet):
    queryset = Link.objects.all()
    serializer_class = LinkSerializer


# serializers.py
class LinkSerializer(serializers.ModelSerializer):
    class Meta:
        model = Link
        fields = ('id', 'url', 'description')


# urls.py
router = routers.DefaultRouter()
router.register(r'links', LinkViewSet)
urlpatterns = [
    url(r'^', include(router.urls)),
]
```

DRF

```
# views.py
class LinkViewSet(viewsets.ModelViewSet):
    queryset = Link.objects.all()
    serializer_class = LinkSerializer


# serializers.py
class LinkSerializer(serializers.ModelSerializer):
    class Meta:
        model = Link
        fields = ('id', 'url', 'description')


# urls.py
router = routers.DefaultRouter()
router.register(r'links', LinkViewSet)
urlpatterns = [
    url(r'^', include(router.urls)),
]
```

DRF

```
# views.py
class LinkViewSet(viewsets.ModelViewSet):
    queryset = Link.objects.all()
    serializer_class = LinkSerializer


# serializers.py
class LinkSerializer(serializers.ModelSerializer):
    class Meta:
        model = Link
        fields = ('id', 'url', 'description')


# urls.py
router = routers.DefaultRouter()
router.register(r'links', LinkViewSet)
urlpatterns = [
    url(r'^', include(router.urls)),
]
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    class Meta:
        model = Link

class Query(graphene.AbstractType):
    links = graphene.List(LinkType)
    link = graphene.List(LinkType, id=graphene.Int())

    def resolve_links(self, args, context, info):
        return Link.objects.all()

    def resolve_link(self, args, context, info):
        return Link.objects.get(id=id)

class SchemaQuery(Query, graphene.ObjectType):
    pass

schema = graphene.Schema(query=SchemaQuery)

# settings.py
GRAPHENE = {'SCHEMA': ‘project.schema.schema'}
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    class Meta:
        model = Link

class Query(graphene.AbstractType):
    links = graphene.List(LinkType)
    link = graphene.List(LinkType, id=graphene.Int())

    def resolve_links(self, args, context, info):
        return Link.objects.all()

    def resolve_link(self, args, context, info):
        return Link.objects.get(id=id)

class SchemaQuery(Query, graphene.ObjectType):
    pass

schema = graphene.Schema(query=SchemaQuery)

# settings.py
GRAPHENE = {'SCHEMA': ‘project.schema.schema'}
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    class Meta:
        model = Link

class Query(graphene.AbstractType):
    links = graphene.List(LinkType)
    link = graphene.List(LinkType, id=graphene.Int())

    def resolve_links(self, args, context, info):
        return Link.objects.all()

    def resolve_link(self, args, context, info):
        return Link.objects.get(id=id)

class SchemaQuery(Query, graphene.ObjectType):
    pass

schema = graphene.Schema(query=SchemaQuery)

# settings.py
GRAPHENE = {'SCHEMA': ‘project.schema.schema'}
```

Graphene

```python
# schema.py
class CreateLink(graphene.Mutation):
    id = graphene.Int()
    url = graphene.String()
    description = graphene.String()

    class Input:
        url = graphene.String()
        description = graphene.String()

    @staticmethod
    def mutate(root, input, context, info):
        link = Link(
            url=input.get('url'),
            description=input.get('description'),
        )
        link.save()

        return CreateLink(
            id=link.id,
            url=link.url,
            description=link.description,
        )


class Mutation(graphene.AbstractType):
    create_link = CreateLink.Field()

schema = graphene.Schema(mutation=Mutation)
```

Graphene

```
# schema.py
class CreateLink(graphene.Mutation):
    id = graphene.Int()
    url = graphene.String()
    description = graphene.String()

    class Input:
        url = graphene.String()
        description = graphene.String()

    @staticmethod
    def mutate(root, input, context, info):
        link = Link(
            url=input.get('url'),
            description=input.get('description'),
        )
        link.save()

        return CreateLink(
            id=link.id,
            url=link.url,
            description=link.description,
        )


class Mutation(graphene.AbstractType):
    create_link = CreateLink.Field()

schema = graphene.Schema(mutation=Mutation)
```

Filtrando

DRF

```python
# views.py
class LinkViewSet(viewsets.ModelViewSet):
    queryset = Link.objects.all()
    serializer_class = LinkSerializer

    def get_queryset(self):
        queryset = self.queryset
        url = self.request.query_params.get('url', None)
        if url:
            return queryset.filter(url__icontains=url)

        return queryset
```

DRF

```python
# views.py
class LinkViewSet(viewsets.ModelViewSet):
    queryset = Link.objects.all()
    serializer_class = LinkSerializer

    def get_queryset(self):
        queryset = self.queryset
        url = self.request.query_params.get('url', None)
        if url:
            return queryset.filter(url__icontains=url)

        return queryset
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    search = graphene.String()

    class Meta:
        model = Link


class Query(graphene.AbstractType):
    links = graphene.List(LinkType)

    def resolve_links(self, args, context, info):
        qs = Link.objects.all()

        search = args.get('search')
        if search:
            qs = qs.filter(url__icontains=search)

        return qs
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    search = graphene.String()

    class Meta:
        model = Link


class Query(graphene.AbstractType):
    links = graphene.List(LinkType)

    def resolve_links(self, args, context, info):
        qs = Link.objects.all()

        search = args.get('search')
        if search:
            qs = qs.filter(url__icontains=search)

        return qs
```

Permissões

DRF

```
# views.py
class LinkViewSet(viewsets.ModelViewSet):
    queryset = Link.objects.all()
    serializer_class = LinkSerializer
    permission_classes = (IsAuthenticated,)
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    class Meta:
        model = Link


class Query(graphene.AbstractType):
    links = graphene.List(LinkType)

    def resolve_links(self, args, context, info):
        if not context.user:
            raise GraphQLError('You must be logged!')

        return Link.objects.all()
```

Lidando com Erros

DRF

```python
# serializers.py
class LinkSerializer(serializers.ModelSerializer):
    class Meta:
        model = Link
        fields = ('id', 'url', 'description')

    def validate_description(self, value):
        if len(value) < 3:
            raise serializers.ValidationError(
                'Description must have more than 3 characters'
            )
        return value
```

DRF

```python
# serializers.py
class LinkSerializer(serializers.ModelSerializer):
    class Meta:
        model = Link
        fields = ('id', 'url', 'description')

    def validate_description(self, value):
        if len(value) < 3:
            raise serializers.ValidationError(
                'Description must have more than 3 characters'
            )
        return value
```

Graphene

```python
# schema.py
class LinkType(DjangoObjectType):
    class Meta:
        model = Link


class Query(graphene.AbstractType):
    links = graphene.List(LinkType)

    def resolve_links(self, args, context, info):
        if not context.user:
            raise GraphQLError('You must be logged!')

        return Link.objects.all()
```

Paginando

DRF

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 1
}
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    class Meta:
        model = Link


class Query(graphene.AbstractType):
    links = graphene.List(LinkType)

    def resolve_links(self, args, context, info):
        first = args.get('first')
        skip = args.get('skip')

        qs = Link.objects.all()

        if skip:
            qs = qs[skip::]

        if first:
            qs = qs[:first]

        return qs
```

Graphene

```
# schema.py
class LinkType(DjangoObjectType):
    class Meta:
        model = Link


class Query(graphene.AbstractType):
    links = graphene.List(LinkType)

    def resolve_links(self, args, context, info):
        first = args.get('first')
        skip = args.get('skip')

        qs = Link.objects.all()

        if skip:
            qs = qs[skip::]

        if first:
            qs = qs[:first]

        return qs
```

Bora testar!

Por que GraphQL?

Flexível
Versionamento mais dinâmico
Forte tipagem e *Schema*
Simplicidade e opinião
Poder ao cliente
Melhor trabalho em equipe

Por que REST?

Bem definido, vastamente usado e muito maduro
HTTP
Tratamento de Erros
Caching
Ferramentas (ecosistema)

REST não está morto
GraphQL não é a solução pra tudo

O futuro da Web API

HOW TO GRAPHQL

The Fullstack Tutorial for GraphQL

The free and open-source tutorial to learn all around GraphQL to go from zero to production.

WATCH OVERVIEW

Start with Introduction

HOW TO GRAPHQL

The Fullstack Tutorial for

WRITTEN BY

Jonatas Baldin

Developer @ Cheesecake Labs

Software developer @ Cheesecake Labs. Speaker, contributor and conferences organizer. Loves open source communities.

graphql-python Tutorial - Introduction

Start with Introduction

Mais Informações

HowToGraphQL
GraphQL.org
GraphQL Weekly
REST API Tutorial
Phil Sturgeon
RESTful Web APIs by O'Reilly

DRF vs Graphene

@jonatasbaldin