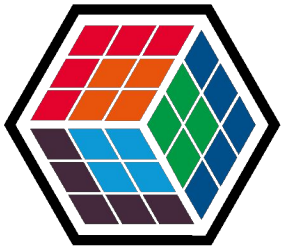


THE DEVELOPER'S CONFERENCE

Trilha – Python

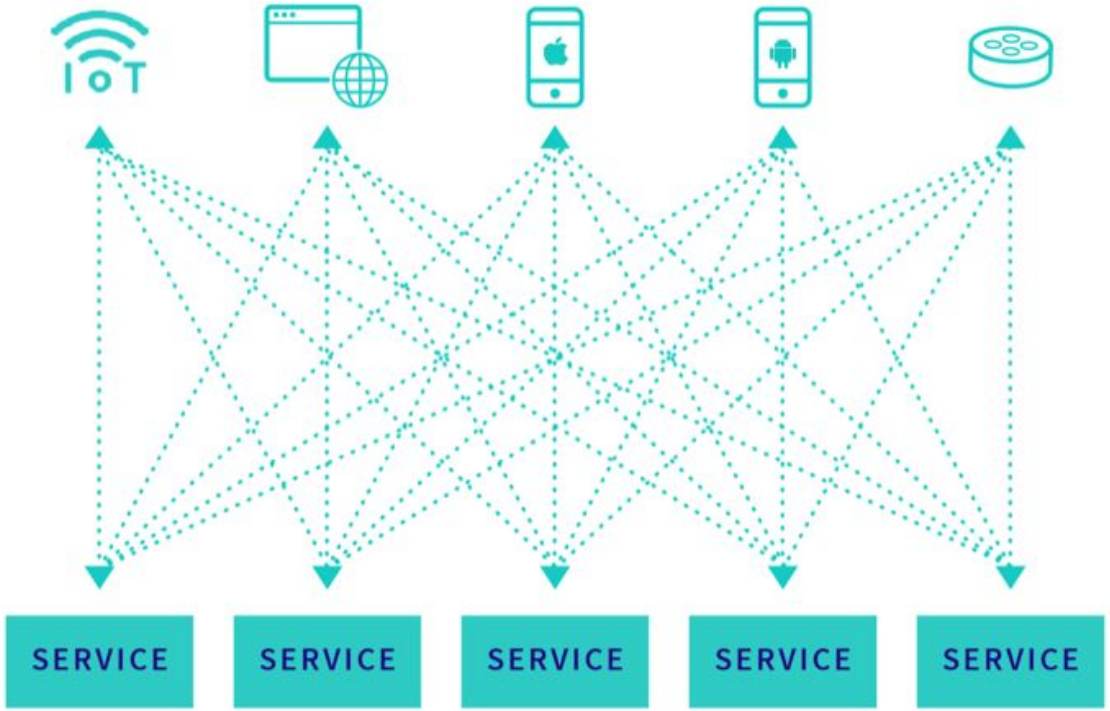
Lucas Costa

GraphQL no mundo real com Graphene

GraphQL

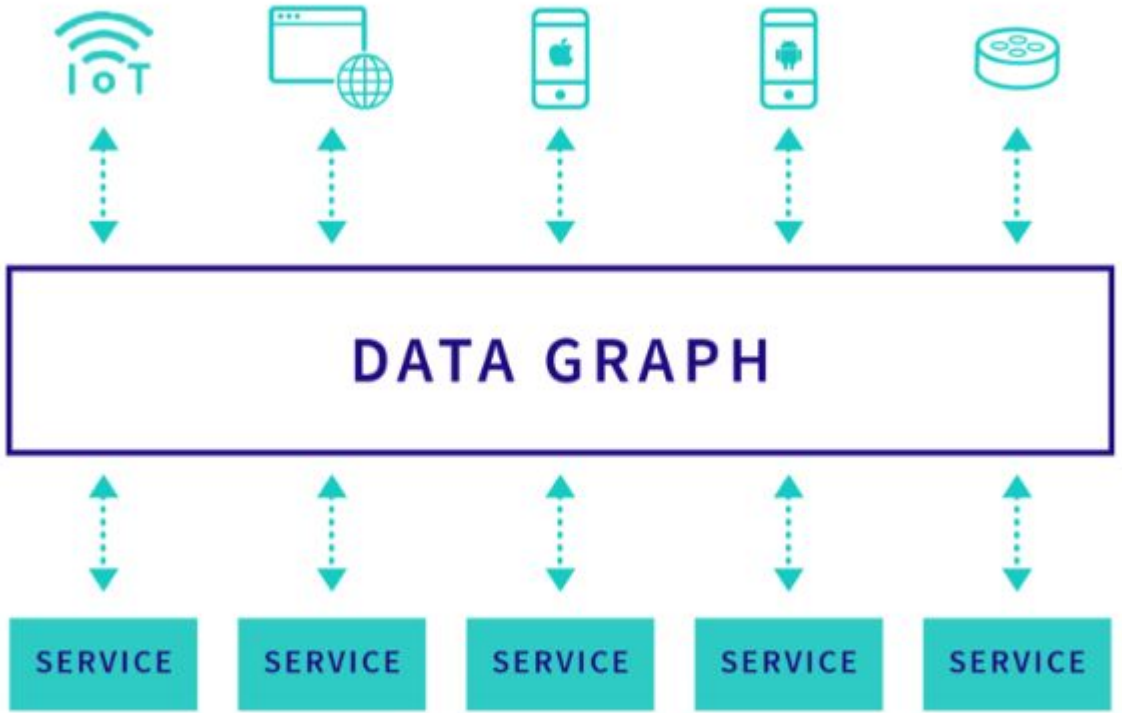
GraphQL is a **query language** for APIs and a **runtime** for fulfilling those queries with your existing data.

GraphQL



datanami.com/2018/11/07/will-graphql-become-a-standard-for-the-new-data-economy

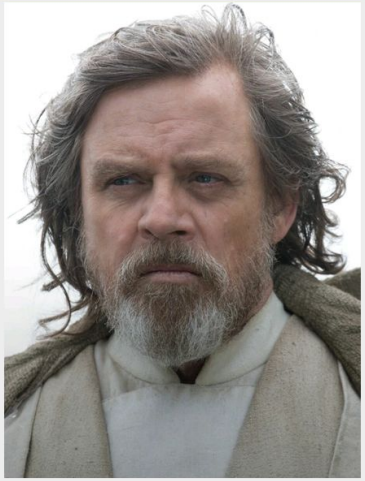
GraphQL



datanami.com/2018/11/07/will-graphql-become-a-standard-for-the-new-data-economy

Exemplo

Luke Skywalker



Informações biográficas	
Planeta natal	Tatooine[1] (nascido em Polis Massa)[2]
Data de nascimento	19 ABY[2][3]
Data de morte	34 DBY[3], Ahch-To[4]
Descrição física	
Espécie	Humano[1]
Gênero	Masculino[1]
Altura	1,72 metro
Cor do cabelo	Loiro[1]; mais tarde, grisalho[5]
Cor dos olhos	Azul[1]
Cor da pele	Clara[1]

Aparições

- *Star Wars: Galactic Defense*
- *Star Wars: The Clone Wars* – "Destiny" (Mencionado indiretamente)
- *Star Wars: The Clone Wars* – "Sacrifice" (Mencionado indiretamente)
- *Star Wars* Episódio III: *A Vingança dos Sith*
- *Star Wars: Kanan 2: The Last Padawan, Parte II: Flight* (Mencionado indiretamente)
- *Star Wars 7: Do diário do velho Ben Kenobi: "O último de sua raça"*
- *Star Wars Rebels: A Fagulha de Uma Rebelião* (Mencionado indiretamente)
- *Star Wars* Episódio IV: *Uma Nova Esperança* (Primeira aparição)
- *A New Hope Read-Along Storybook and CD*
- *Star Wars: Heroes Path*
- *Estrelas Perdidas* (Apenas mencionado)

starwars.fandom.com/pt/wiki/Luke_Skywalker

Exemplo

REST endpoints

```
swrestapi.com/person/[person_id]?
    fields=id,name,home_planet_id,species_id,film_ids,...

swrestapi.com/planets/[planet_id]?fields=name

swrestapi.com/species/[species_id]?fields=name

swrestapi.com/films/[film_id_1],[film_id_2],...
    ?fields=title,release_date,...
```

Exemplo

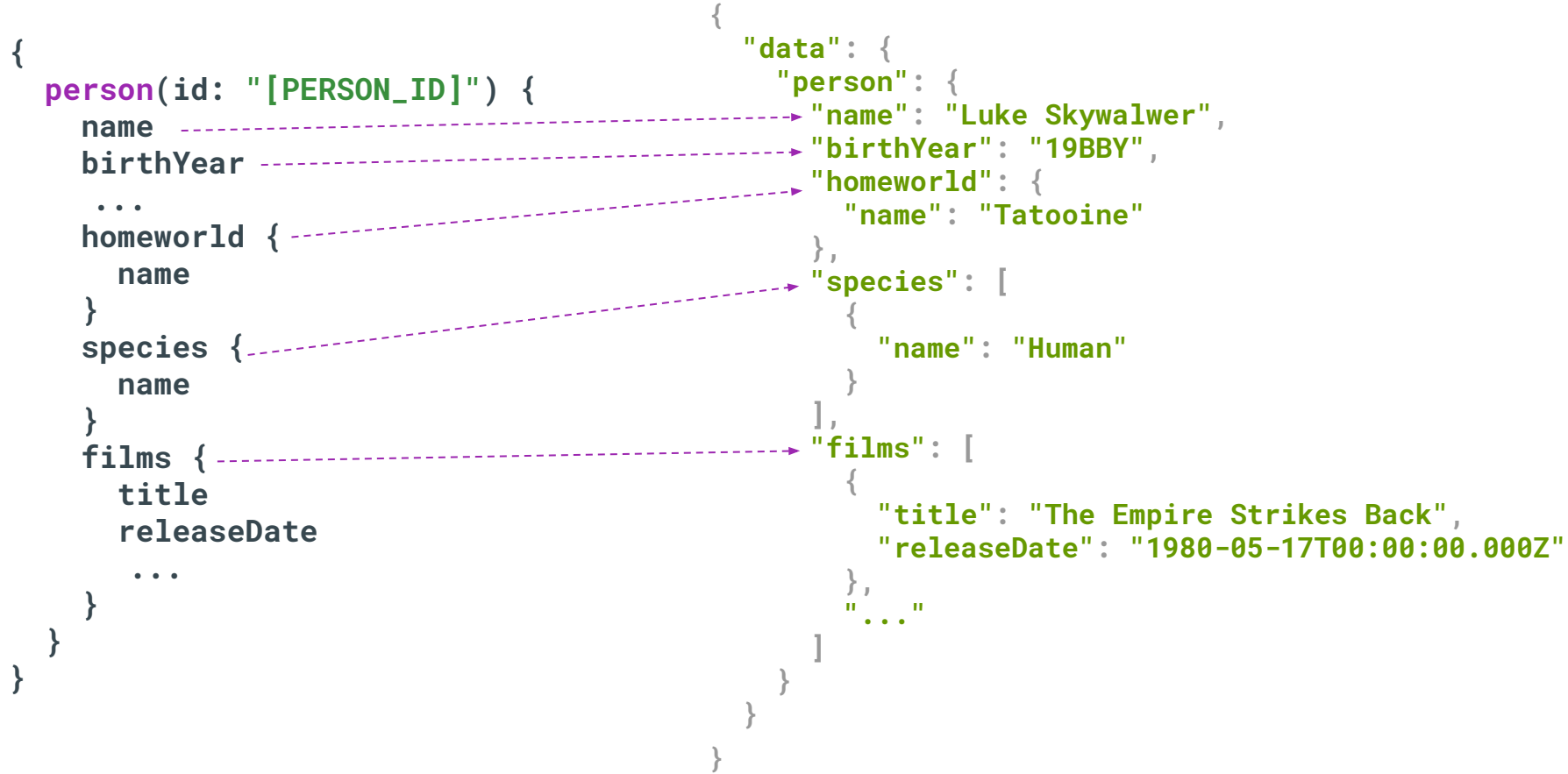
Custom endpoints

`swrestapi.com/profile/[person_id]`

`swrestapi.com/mobile/profile/[person_id]`

`swrestapi.com/.../profile/[person_id]`

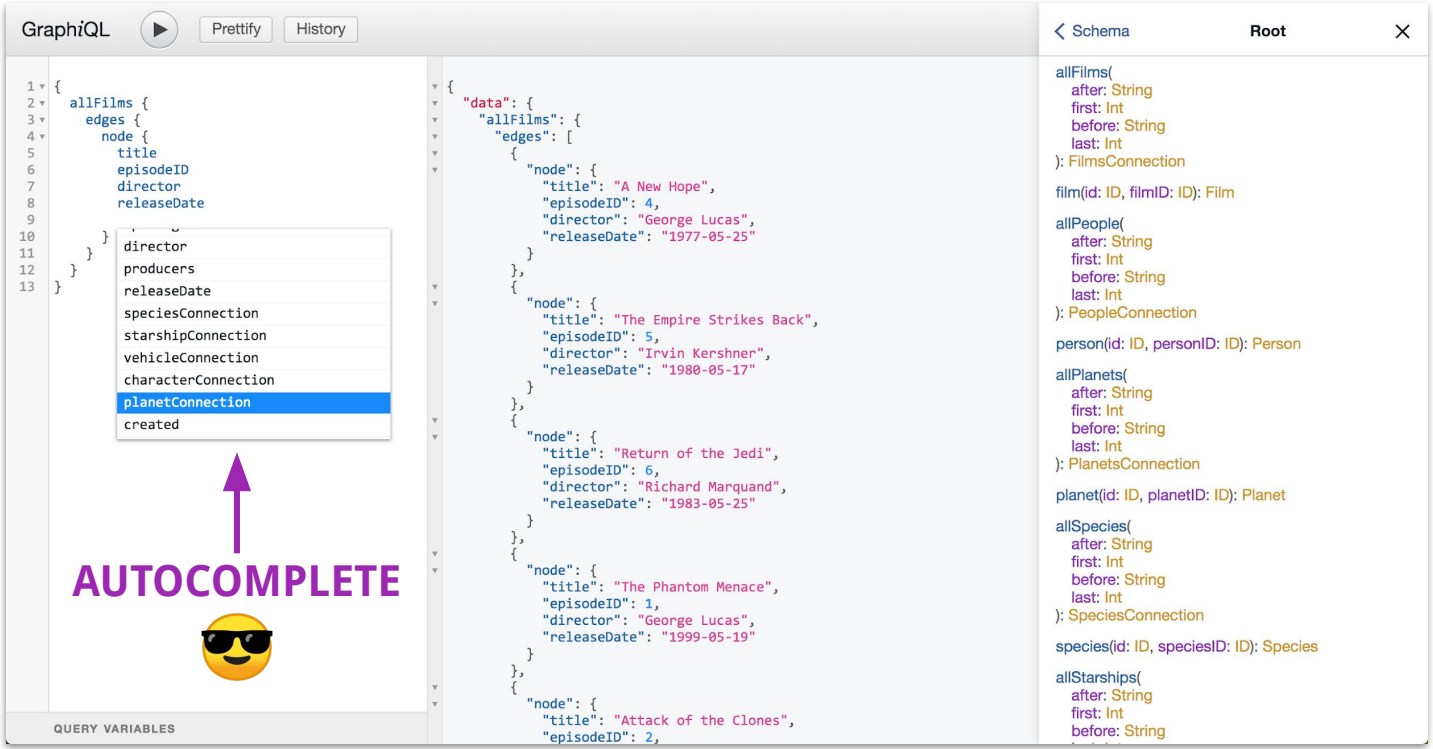
GraphQL


{
person(id: "[PERSON_ID]") {
name
birthYear
...
homeworld {
name
}
species {
name
}
films {
title
releaseDate
...
}
}
}
{
"data": {
"person": {
"name": "Luke Skywalwer",
"birthYear": "19BBY",
"homeworld": {
"name": "Tatooine"
},
"species": [
{
"name": "Human"
}
],
"films": [
{
"title": "The Empire Strikes Back",
"releaseDate": "1980-05-17T00:00:00.000Z"
},
"..."
]
}
}
}

Algumas vantagens

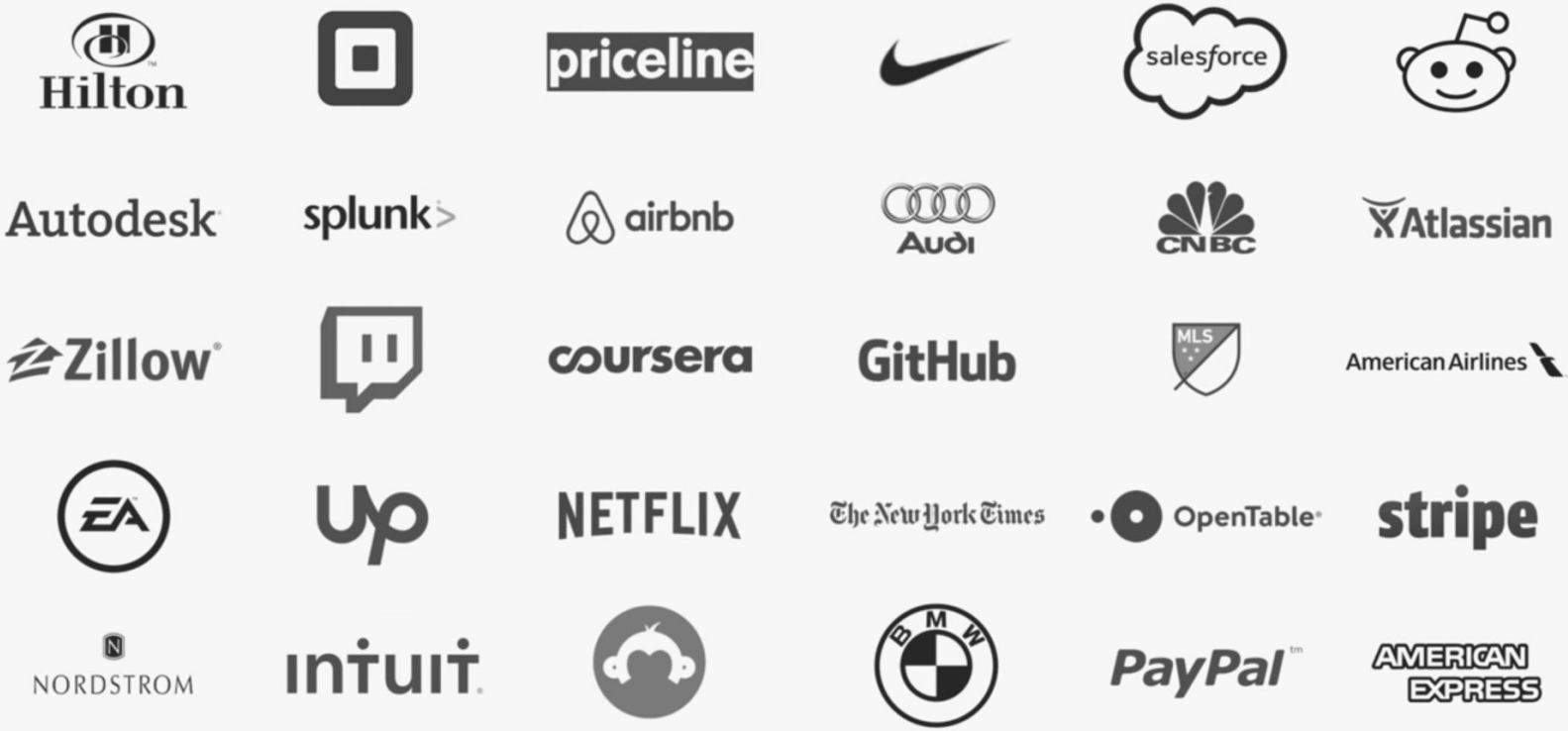
- Facilita desenvolvimento (contratos claros e estrutura coerente)
- Documentação embutida
- Expõe os recursos para os clientes (fim dos endpoints customizados)
- Otimiza eficiência de requisições
- Agiliza desenvolvimento *cross-platform*
- Centraliza lógica de negócios
- Auxilia transição para microsserviços
- Proporciona gerenciamento e segurança da API centralizados
- Facilita descoberta de recursos através da introspecção do Schema
- Possibilita modelagem da API baseada no domínio do negócio
- Democratiza acesso à API a outras áreas

GraphiQL (IDE)



graphql.github.io/swapi-graphql

Adoção



youtube.com/watch?v=IjJkAL4RWyQ

Schema

Queries

Consulta de dados

- Fields
- Arguments

```
{
  human(id: "1000") {
    name
    height
  }
}
```

Mutations

Alteração de dados

- Name
- Arguments
- Response Fields

```
mutation CreateNewPerson {
  createPerson(name: "Yoda") {
    name
  }
}
```

Tipos

Scalars (dados concretos)

- Int
- Float
- Boolean
- String
- ID

Enums

Lists

Custom Scalars

- Date
- Time
- JSONString
- [Write your own]

Objects

Inputs

Execução

- Parse
- Validate
- Resolve/Mutate

Resolver

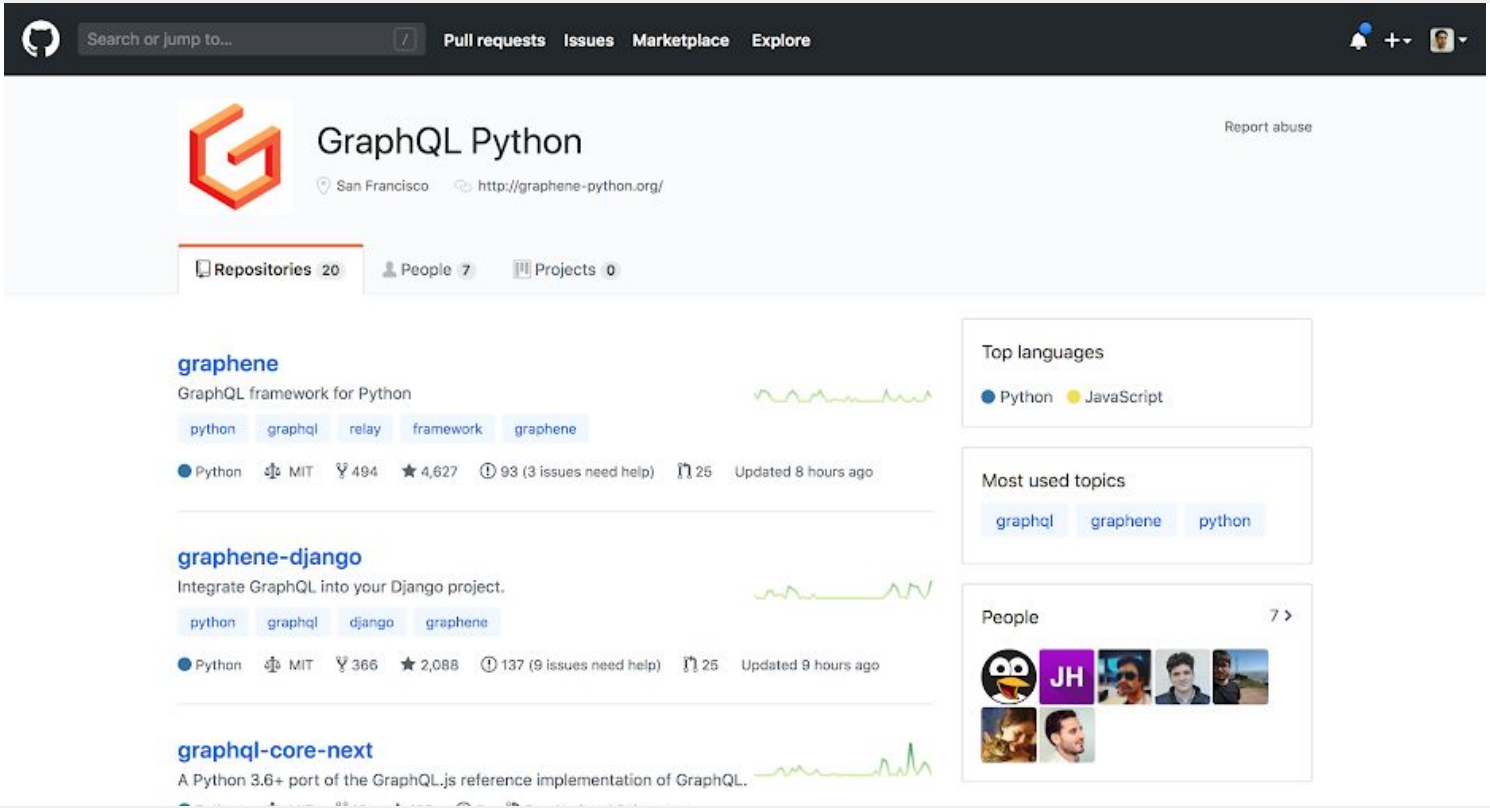
Função que produz o resultado de um campo

Graphene Python

graphene-python.org

Framework de GraphQL em Python

Ecossistema



github.com/graphql-python / swapi.graph.cool

Ecossistema

graphql-core (graphql-core-next)

graphene

graphql-server-core

flask-graphql

graphene-django

....

./hello_world.py

```python
import graphene

class Query(graphene.ObjectType):
    hello = graphene.String()

    def resolve_hello(self, info):
        return 'World'

schema = graphene.Schema(query=Query)
response = schema.execute('''
  query {
    hello
  }
''')
print(response.to_dict())
```

```
~$ python hello_world.py
{'data': {'hello': 'World'}}
```

./schema.py

```python
import graphene

class Person(graphene.ObjectType):
    id = graphene.ID()
    name = graphene.String()
    birth_year = graphene.String()

class Query(graphene.ObjectType):
    person = graphene.Field(Person)

    def resolve_person(self, info):
        return Person(
            id="1", name="Luke Skywalker", birth_year="19BBY")

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

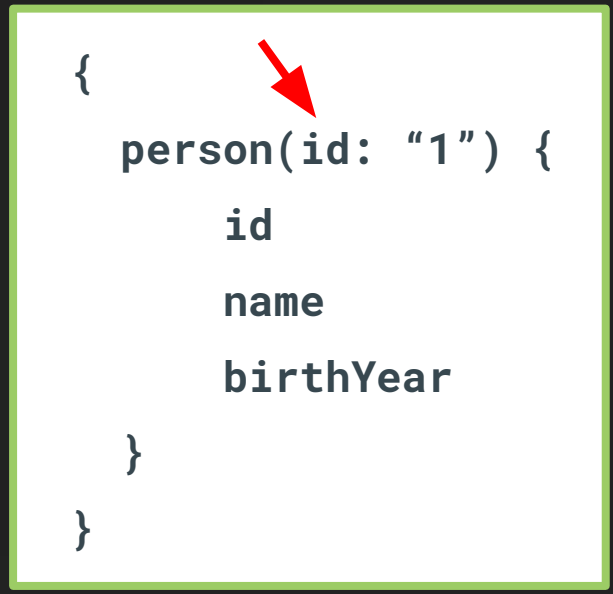
```
{
  person {
      id
      name
      birthYear
  }
}
```

./schema.py

```python
import graphene

class Person(graphene.ObjectType):
    id = graphene.ID()
    name = graphene.String()
    birth_year = graphene.String()

class Query(graphene.ObjectType):
    person = graphene.Field(Person)

    def resolve_person(self, info):
        person = {
            "id": "1",
            "name": "Luke Skywalker",
            "birth_year": "19BBY",
        }
        return Person(**person)

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

./schema.py

```python
import graphene
from repositories.people_repository import PeopleRepository

class Person(graphene.ObjectType):
    id = graphene.ID()
    name = graphene.String()
    birth_year = graphene.String()

class Query(graphene.ObjectType):
    person = graphene.Field(
        Person, id=graphene.String())

    def resolve_person(self, info, id):
        person = PeopleRepository.find_by_id(id)
        return Person(**person)

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



./schema.py

```python
import graphene

from repositories.people_repository import PeopleR
from repositories.planets_repository import Planet

class Planet(graphene.ObjectType):
    id = graphene.ID()
    name = graphene.String()

class Person(graphene.ObjectType):
    id = graphene.ID()
    name = graphene.String()
    birth_year = graphene.String()
    homeworld = graphene.Field(Planet)

    @staticmethod
    def resolve_homeworld(parent, info):
        person_id = parent.id
        planet = PlanetRepository.find_by_person_id(person_id)
        return Planet(**planet)
```

```
{
  person(id: "1") {
      id
      name
      birthYear
      homeworld {
          name
      }
  }
}
```

./schema.py

```python
from repositories.films_repository import FilmsRep

class Film(graphene.ObjectType):
    id = graphene.ID()
    title = graphene.String()
    release_date = graphene.types.datetime.Date()

class Person(graphene.ObjectType):
    id = graphene.ID()
    name = graphene.String()
    birth_year = graphene.String()
    homeworld = graphene.Field(Planet)
    films = graphene.List(Film)

    @staticmethod
    def resolve_films(parent, info):
        person_id = parent.id
        films = FilmsRepository.find_by_person_id(person_id)
        return [Film(**film) for film in films]
```

```
{
  person(id: "1") {
      id
      name
      birthYear
      homeworld {
          name
      }
      films {
          title
          releaseDate
      }
  }
}
```

./schema.py

```python
from graphene.types.generic import GenericScalar

class FilmsEdge(graphene.ObjectType)
    edges = graphene.List(Film)
    paging = GenericScalar()

class Person(graphene.ObjectType):
    [...]
    films = graphene.Field(
        FilmsEdge, limit=graphene.Int(), paging=GenericScalar())

    @staticmethod
    def resolve_films(parent, info, limit=None, paging=None):
        person_id = parent.id
        films, paging = FilmsRepository.find_by_person_id(
            person_id, limit, paging)
        edge = {edges": films, paging": paging}
        return FilmsEdge(**edge)
```

./schema.py

```python
from graphene.types.generic import

class FilmsEdge(graphene.ObjectType
    edges = graphene.List(Film)
    paging = GenericScalar()

class Person(graphene.ObjectType):
    [...]
    films = graphene.Field(
        FilmsEdge, limit=graphene.

    @staticmethod
    def resolve_films(parent, info
        person_id = parent.id
        films, paging = FilmsRepository.find_by_person_id(
            person_id, limit, paging)
        edge = {edges": films, paging": paging}
        return FilmsEdge(**edge)
```

```
{
  person(id: "1") {
      id
      name
      birthYear
      films(limit: 3, paging: [CURSOR])
{
          edges {
              title
              releaseDate
          }
          paging
      }
  }
}
```

./schema.py

```python
class PersonInput(graphene.InputObjectTyp
    name = graphene.String(required=True
    birth_year = graphene.String(require
    homeworld_id = graphene.ID(required=
    film_ids = graphene.List(graphene.ID

class CreatePerson(graphene.Mutation):
    class Arguments:
        person = PersonInput(required=Tru

    person = graphene.Field(Person)

    def mutate(self, info, person):
        new_person = PeopleRepository.cr
        return CreatePerson(person=Perso

class Mutations(graphene.ObjectType):
    create_person = CreatePerson.Field()

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

```
mutation CreatePerson  {
    createPerson(
        person: {
            name: "Darth Vader"
            birthYear: "41.9BBY"
            homeworldId: "568"
            filmIds: ["123", "234"]
        }
    ) {
        person {
            id
            name
            birthYear
            homeworld {
                name
            }
        }
    }
}
```

./schema.py

```python
class UpdatePersonInput(graphene.InputObjectType):
    name = graphene.String()
    birth_year = graphene.String()
    homeworld_id = graphene.ID()
    film_ids = graphene.List(graphene.ID, )

class UpdatePerson(graphene.Mutation):
    class Arguments:
        id = graphene.ID(required=True)
        person = UpdatePersonInput(required=True)

    person = graphene.Field(Person)

    def mutate(self, info, id, person):
        updated_person = PeopleRepository.update_person(id, person)
        return UpdatePerson(person=Person(**updated_person))

class Mutations(graphene.ObjectType):
    create_person = CreatePerson.Field()
    update_person = UpdatePerson.Field()

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

./schema.py

```python
class UpdatePersonInput(graphene.InputObj
    name = graphene.String()
    birth_year = graphene.String()
    homeworld_id = graphene.ID()
    film_ids = graphene.List(graphene.ID,

class UpdatePerson(graphene.Mutation):
    class Arguments:
        id = graphene.ID(required=True)
        person = UpdatePersonInput(requir

    person = graphene.Field(Person)

    def mutate(self, info, id, person):
        updated_person = PeopleRepository
        return UpdatePerson(person=Person

class Mutations(graphene.ObjectType):
    create_person = CreatePerson.Field()
    update_person = UpdatePerson.Field()

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

```
mutation UpdatePerson  {
    updatePerson(
        id: "2"
        person: {
            homeworldId: "568"
        }
    ) {
        person {
            id
            name
            birthYear
            homeworld {
                name
            }
        }
    }
}
```

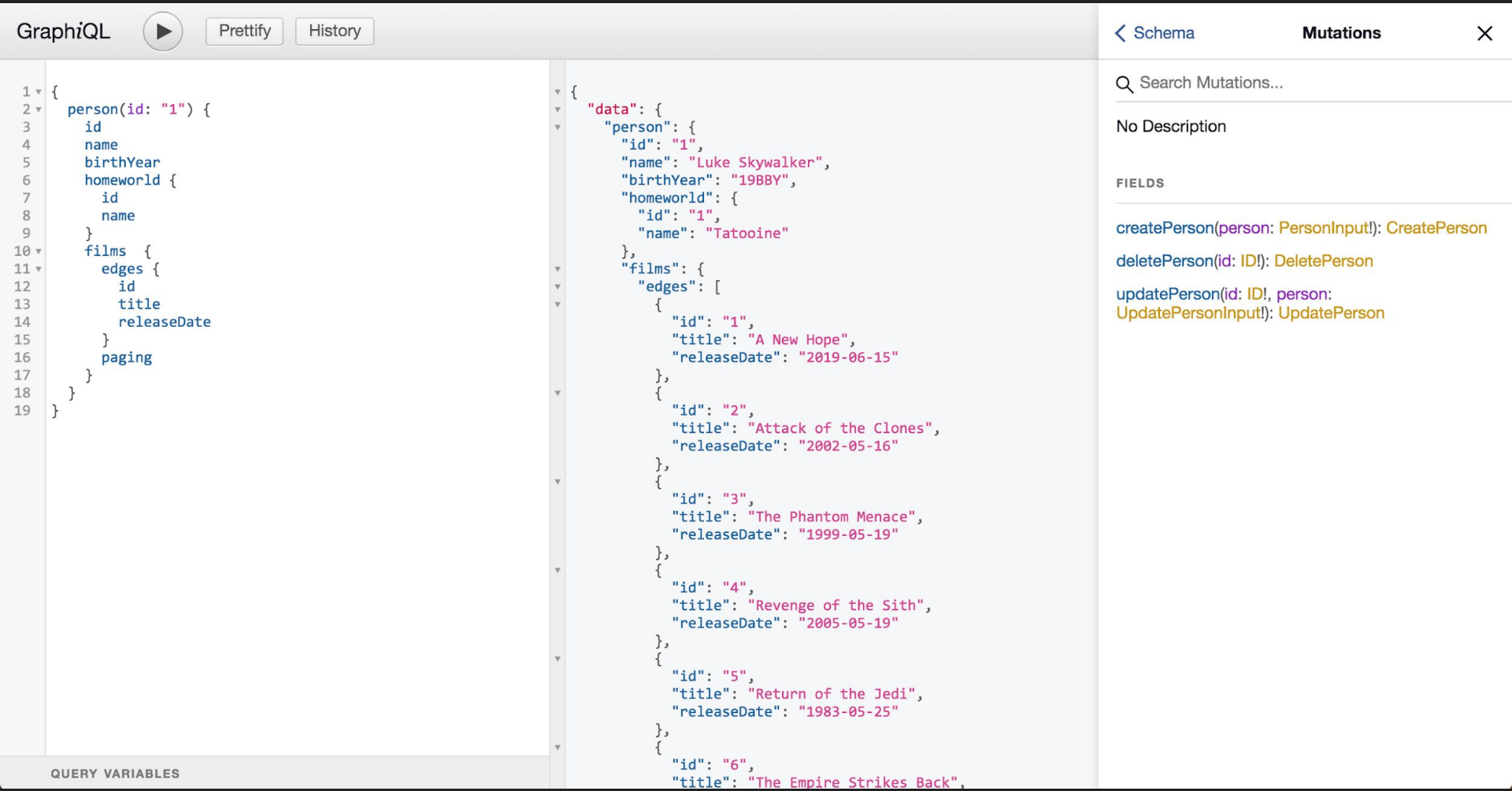
./schema.py

```python
class DeletePerson(graphene.Mutation):

    class Arguments:
        id = graphene.ID(required=True)

    success = graphene.Boolean()

    def mutate(self, info, id):
        PeopleRepository.delete_person(id)
        return DeletePerson(success=True)

class Mutations(graphene.ObjectType):
    create_person = CreatePerson.Field()
    delete_person = DeletePerson.Field()
    update_person = UpdatePerson.Field()

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

```
mutation DeletePerson  {
    deletePerson(
        id: "2"
    ) {
        success
    }
}
```

./app.py

```
import flask
import flask_cors
from flask_graphql import GraphQLView

from schema import schema

app = flask.Flask(__name__)
app.add_url_rule(
    '/graphql',
    view_func=GraphQLView.as_view('graphql', schema=schema, graphiql=True)
)

flask_cors.CORS(app)

if __name__ == '__main__':
    app.run()
```


GraphiQL
Prettify
History
{
person(id: "1") {
id
name
birthYear
homeworld {
id
name
}
films {
edges {
id
title
releaseDate
}
paging
}
}
}
QUERY VARIABLES
{
"data": {
"person": {
"id": "1",
"name": "Luke Skywalker",
"birthYear": "19BBY",
"homeworld": {
"id": "1",
"name": "Tatooine"
},
"films": {
"edges": [
{
"id": "1",
"title": "A New Hope",
"releaseDate": "2019-06-15"
},
{
"id": "2",
"title": "Attack of the Clones",
"releaseDate": "2002-05-16"
},
{
"id": "3",
"title": "The Phantom Menace",
"releaseDate": "1999-05-19"
},
{
"id": "4",
"title": "Revenge of the Sith",
"releaseDate": "2005-05-19"
},
{
"id": "5",
"title": "Return of the Jedi",
"releaseDate": "1983-05-25"
},
{
"id": "6",
"title": "The Empire Strikes Back",
Schema
Mutations
Search Mutations...
No Description
FIELDS
createPerson(person: PersonInput!): CreatePerson
deletePerson(id: ID!): DeletePerson
updatePerson(id: ID!, person: UpdatePersonInput!): UpdatePerson

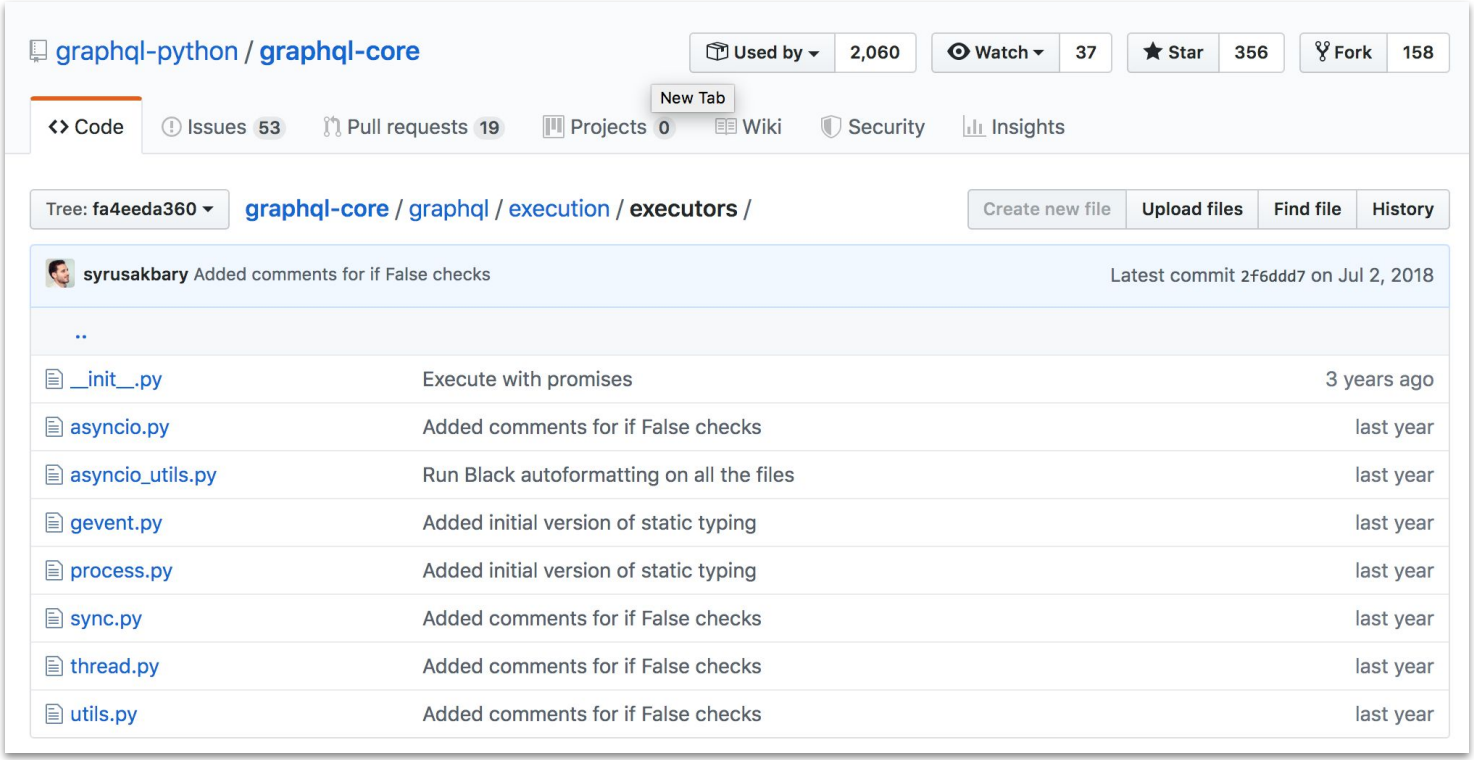
Tips & Tricks

Estrutura de Arquivos

- schema.py
- domain/domain_enums.py
- domain/domain_inputs.py
- domain/domain_mutations.py
- domain/domain_query.py
- domain/domain_resolvers.py

Tips & Tricks

Executors



graphql-python / graphql-core

Used by 2,060 | Watch 37 | Star 356 | Fork 158

New Tab

Code | Issues 53 | Pull requests 19 | Projects 0 | Wiki | Security | Insights

Tree: fa4eeda360 graphql-core / graphql / execution / executors / Create new file | Upload files | Find file | History

syrusakbary Added comments for if False checks — Latest commit 2f6ddd7 on Jul 2, 2018

..		
__init__.py	Execute with promises	3 years ago
asyncio.py	Added comments for if False checks	last year
asyncio_utils.py	Run Black autoformatting on all the files	last year
gevent.py	Added initial version of static typing	last year
process.py	Added initial version of static typing	last year
sync.py	Added comments for if False checks	last year
thread.py	Added comments for if False checks	last year
utils.py	Added comments for if False checks	last year

./app.py

```
import os
import flask
import flask_cors
from flask_graphql import GraphQLView
from graphql.execution.executors import asyncio

from schema import schema

executor = asyncio.AsyncioExecutor() if not os.getenv('IS_LOCAL') else None

app = flask.Flask(__name__)
app.add_url_rule(
    '/graphql',
    view_func=GraphQLView.as_view(
        'graphql',
        schema=schema,
        graphiql=True,
        executor=executor,
    )
)
```

Tips & Tricks

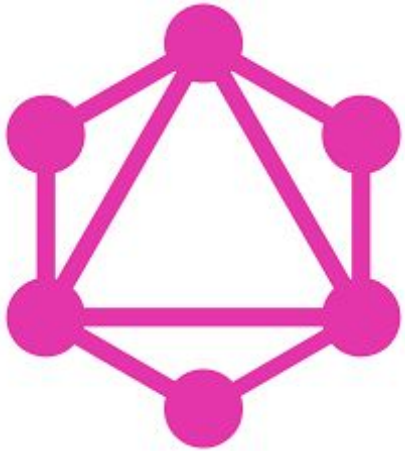
Batching

```
{
  allPersons(limit: 100) {
    id
    name
    films {
      id
      title
    }
  }
}
```

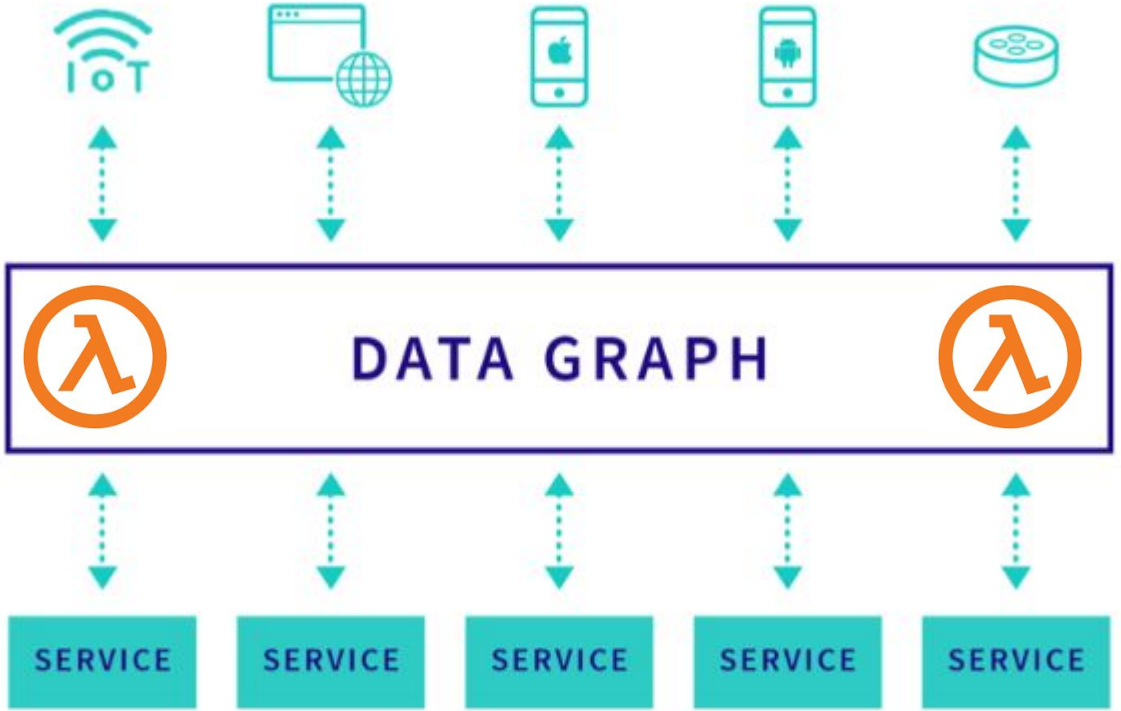
./resolvers.py

```python
from promise import Promise
from promise.dataloader import DataLoader


class PersonFilmsLoader(DataLoader):
    def batch_load_fn(self, people_ids):
        if len(people_ids) == 1:
            person_id = people_ids[0]
            films = FilmsRepository.find_by_person_id(person_id)
        else:
            films = FilmsRepository.find_by_people_ids(people_ids)

        return Promise.resolve([Film(**film) for film in films])

person_films_loader = PersonFilmsLoader()

def resolve_films(parent):
    person_id = parent.id
    return person_films_loader.load(person_id)
```

Tips & Tricks



Tips & Tricks



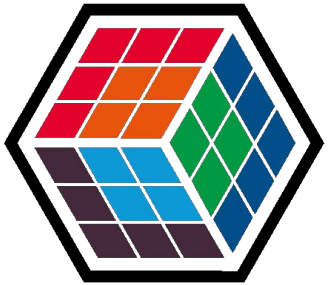
./app.py

```python
import flask
import flask_cors
import serverless_wsgi
from flask_graphql import GraphQLView

from schema import schema

app = flask.Flask(__name__)
app.add_url_rule(
    '/graphql',
    view_func=GraphQLView.as_view('graphql', sc
)

flask_cors.CORS(app)

def handler(event, context):
    return serverless_wsgi.handle_request(app, event, context)

if __name__ == '__main__':
    app.run()
```

/serverless.yml

```yaml
[...]
graph:
  handler: graph/app.handler
  events:
    - http:
        path: /
        method: get
    - http:
        path: /
        method: post
[...]
```

Obrigado



THE DEVELOPER'S CONFERENCE

@lucasrcosta