



**LAB
CODES**
SOFTWARE STUDIO

GraphQL: nem sempre REST é a melhor alternativa para sua API

Nicolle Cysneiros

QUEM SOU EU?

- Fullstack Developer @ Labcodes
 - Django
 - AngularJS
- PUG-PE
- PyLadies Recife

LABCODES

- Estúdio de Software de Recife para o mundo
- Tecnologias trabalhadas
- Labcodes e a comunidade



MOTIVAÇÃO

GRAPHQL

- Novo padrão para APIs
- Linguagem de consulta
- Desenvolvido e mantido open-source pelo


facebook®

QUEM USA?



dailymotion






Jonas Helfer in Apollo GraphQL

Mar 3 · 7 min read

```

28     </ul>;
29   };
30
31   const ChannelsListWithData = graphql(gql`{
32     channels {
33       id
34       name
35     }
36   }`)(ChannelsList);
37


```



Jonas Helfer in Apollo GraphQL

Apr 22, 2016 · 9 min read

Full-stack





Scott Taylor in Times Open


Jun 29 · 5 min read

Part 1—the

Read more...

 1.3K






Read more...

In the world of architecture, REST has

decade or more...

Read more...

 1.8K


```

graph TD
    A(( )) --- B(( ))
    A --- C(( ))
    A --- D(( ))
    B --- C
    B --- E(( ))
    C --- D
    C --- E
    D --- E
    D --- F(( ))
    E --- F
    F --- G(( ))
  
```


Tutorial: How to build a GraphQL server

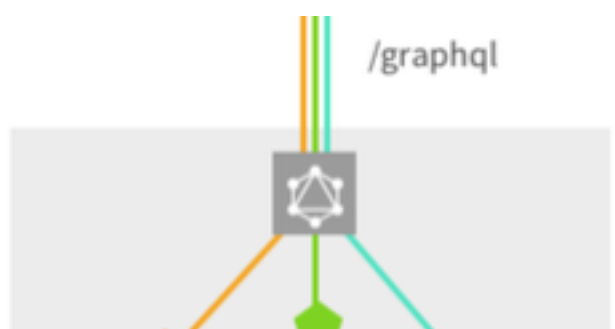
Talk to MongoDB, SQL and REST with these simple steps

Read more...

 799

31 responses







GraphQL vs. REST

Two ways to send data over HTTP: What's the difference?

Read more...

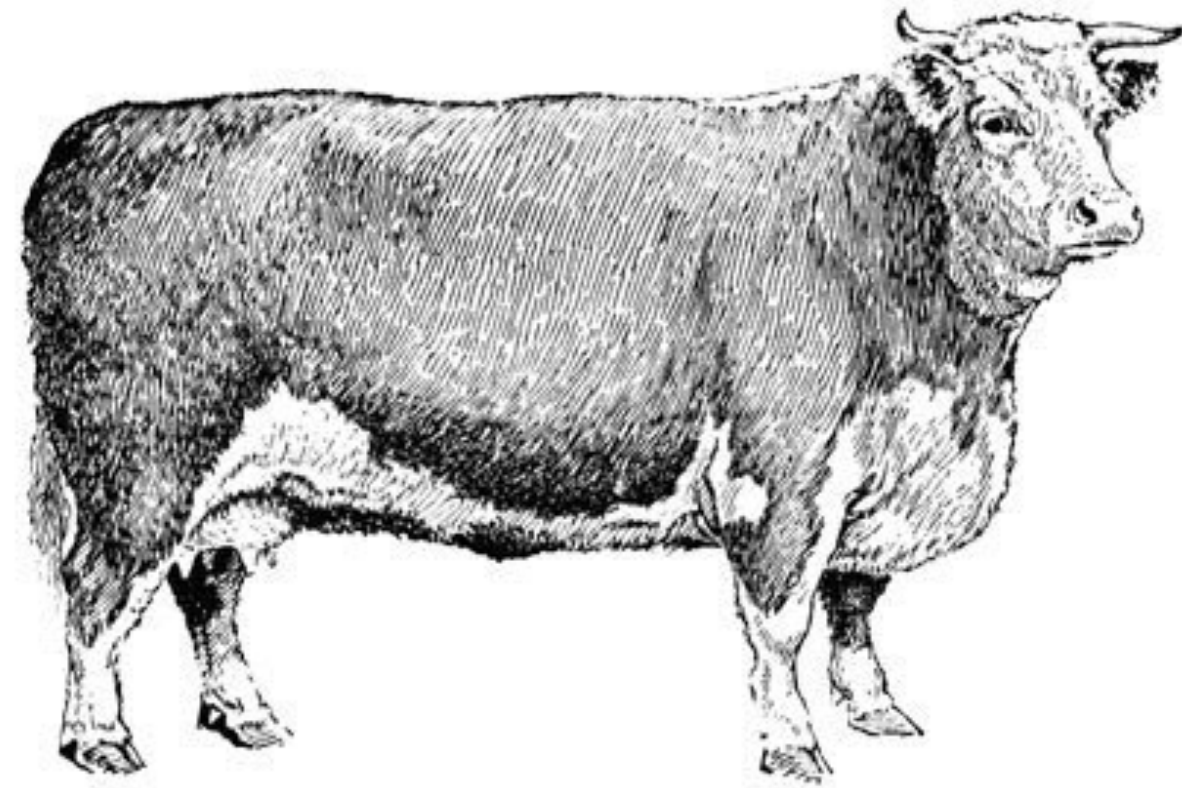
 1.5K

26 responses



HYPE?

Qual problema
GraphQL está
tentando resolver?



Hype-Driven
Development

Follow The Herd

REST

REST

- Representational State Transfer
- Estilo arquitetural baseado em **recursos**
 - Entidades do sistema
 - Identificados por URIs
 - Manipulados através de **representações** (HTML, XML, JSON)

REST - PRECEITOS

- Cliente-servidor
- Stateless
- Cache
- Sistema em camadas
- Código sob demanda
- Interface uniforme

```
GET /api/users/12345678
```

```
{  
  "id": "12345678",  
  "name": "John Snow",  
  "age": "15",  
  "gender": "Male",  
  "profile_pic": "http://bit.ly/2hLdTor",  
}
```


GET /api/users/12345678/friends

```
{
  "friends": [
    {
      "id": "12345679",
      "name": "Daenerys Targaryen",
      "age": "14",
      "gender": "Female",
      "profile_pic": "http://bit.ly/3gHeUps",
    },
    {
      "id": "12345680",
      "name": "Arya Stark",
      "age": "10",
      "gender": "Female",
      "profile_pic": "http://bit.ly/4hIfVot",
    },
    ...
  ]
}
```

MAIS REST?

“O que é esse tal de REST?”

[https://
www.youtube.com/
watch?v=ptzNmai1hCQ](https://www.youtube.com/watch?v=ptzNmai1hCQ)

[https://pt.slideshare.net/
filipeximenes/o-que-
esse-tal-de-rest-
pybr2016](https://pt.slideshare.net/filipeximenes/o-que-esse-tal-de-rest-pybr2016)



PROBLEMAS

> Overfetching

- Download de dados desnecessários

> Underfetching

- Acessar mais de um endpoint para obter a informação desejada

PROBLEMAS

- Estruturar endpoints de acordo com as views da aplicação
- Front-end e back-end acoplados
- Não permite iterações rápidas da aplicação

GRAPHQL

CONCEITOS FUNDAMENTAIS

SCHEMA

- Recursos disponíveis são definidos por um sistema de tipos
- Contrato entre cliente e servidor sobre como a aplicação pode acessar os dados
 - Front-end e back-end **desacoplados**
- Schema Definition Language (SDL)

```
type User {  
    id: ID!,  
    name: String!  
    age: Int!  
    gender: String  
    profile_pic: String  
    friends: [User]  
    communities: [Community]  
}
```

```
type Community {  
    id: ID!  
    name: String!  
    category: String!  
    users: [User]  
}
```

QUERY

- Em APIs REST, os dados são codificados nas URLs
 - Múltiplos endpoints retornam estruturas de dados fixas
- Em APIs GraphQL, existe um **único endpoint**
 - O cliente deve informar qual dado será necessário

```
{  
  allUsers {  
    name  
    profile_pic  
  }  
}
```



```
{
  "data": {
    "allUsers": [
      {
        "name": "John Snow",
        "profile_pic": "http://bit.ly/2hLdTor"
      },
      {
        "name": "Daenerys Targaryen",
        "profile_pic": "http://bit.ly/3gHeUps"
      }
    ]
  }
}
```

```
{  
  user(name: "John Snow") {  
    name  
    profile_pic  
    friends {  
      name  
      profile_pic  
    }  
  }  
}
```

```
{
  "data": {
    "user": {
      "name": "John Snow",
      "profile_pic": "http://bit.ly/2hLdTor",
      "friends": [
        {
          "name": "Daenerys Targaryen",
          "profile_pic": "http://bit.ly/3gHeUps"
        },
        {
          "name": "Arya Stark",
          "profile_pic": "http://bit.ly/4hIfVot"
        }
      ]
    }
  },
}
```

MUTATION

- Realiza mudanças nos dados armazenados no back-end
 - Criar
 - Alterar
 - Deletar

```
mutation {  
  createUser(  
    name: "Cersei Lannister",  
    age: 30,  
    gender: "Female" ) {  
    id  
  }  
}
```


SUBSCRIPTION

- Conexão real-time com o servidor
 - Cliente subscreve para um evento
- Quando o evento acontece, os dados são enviados do servidor para o cliente
 - Stream de dados

SCHEMA

- Os tipos **Query**, **Mutation** e **Subscription** também precisam ser definidos no schema
- São os pontos de entrada para as requisições enviadas pelo cliente

```
type Query {  
    allUsers: [User!]!  
    user(name: String): User!  
}
```

```
type Mutation {  
    createUser(  
        name: String!,  
        age: Int!,  
        gender: String,  
        profile_pic: String): User!  
}
```

```
type Subscription {  
    newUser: User!  
}
```

COMO USA?

- GraphQL nada mais é que uma **especificação**
 - Documento que descreve como um servidor GraphQL deve se comportar
- Você tem que implementar o próprio servidor GraphQL
 - Implementar o stream de dados do Subscription

GRAPHENE

GRAPHENE

- É uma lib Python para construir schemas GraphQL
- Possui diferentes integrações
 - Django
 - SQL Alchemy
 - Google App Engine



GRAPHENE

- Enums e Scalars
- Lists
- Interfaces
- **Object Types**
- Mutation
- Schema

GRAPHENE

➤ Principais tipos escalares:

- `graphene.String`
- `graphene.Int` and `graphene.Float`
- `graphene.Boolean`
- `graphene.ID`
- `graphene.types.datetime.DateTime`
- `graphene.types.json.JSONString`

➤ Lists `graphene.List`

```
import graphene
```

```
class User(graphene.ObjectType):  
    id = graphene.ID()  
    name = graphene.String(required=True)  
    age = graphene.Int(required=True)  
    gender = graphene.String()  
    profile_pic = graphene.String()  
    friends = graphene.List(lambda: User)  
    communities = graphene.List(lambda: Community)
```

```
class Community(graphene.ObjectType):  
    id = graphene.ID()  
    name = graphene.String(required=True)  
    category = graphene.String(required=True)  
    users = graphene.List(User)
```

GRAPHENE

- Query Type
- Resolvers calculam um campo do tipo
 - O nome da função depende do nome do campo
 - Podem receber argumentos da query

```
import graphene

class Query(graphene.ObjectType):
    all_users = graphene.List(lambda: User)
    user = graphene.Field(
        lambda: User,
        name=graphene.String()
    )

    def resolve_all_users(self, info):
        return info.context.get('users')

    def resolve_user(self, info, name):
        users = info.context.get('users')

        for user in users:
            if user.name == name:
                return user
```

```
schema = graphene.Schema(query=Query)
context = {...}

result_all_users = schema.execute(
    '''{
        allUsers {
            name
            profilePic
        }
    }''',
    context_value=context
)

print(result_all_users.data)
```



```
OrderedDict([
  ('allUsers', [
    OrderedDict([
      ('name', 'Daenerys Targaryen'),
      ('profilePic', 'http://bit.ly/3gHeUps')]),
    OrderedDict([
      ('name', 'Arya Stark'),
      ('profilePic', 'http://bit.ly/4hIfVot')]),
    OrderedDict([
      ('name', 'John Snow'),
      ('profilePic', 'http://bit.ly/2hLdTor')])
  ])
])
```

```
result_get_user = schema.execute(  
    '''query getUser($name: String) {  
        user(name: $name) {  
            id  
            name  
            profilePic  
        }  
    }''',  
    context_value=context,  
    variable_values={'name': 'John Snow'}  
)
```

```
OrderedDict([
  ('user', OrderedDict([
    ('id', '<graphql.types.scalars.ID object...>'),
    ('name', 'John Snow'),
    ('profilePic', 'http://bit.ly/2hLdTor')]))
])
```

GRAPHENE

- Mutations
- Os argumentos da `Mutation` são definidos na classe `Arguments`
- É preciso implementar o método `mutate`

```
class CreateUser(graphene.Mutation):
    class Arguments:
        name = graphene.String(required=True)
        age = graphene.Int(required=True)
        gender = graphene.String()
        profile_pic = graphene.String()

    user = graphene.Field(lambda: User)

    def mutate(root, info, name, age, gender, profile_pic):
        user = User(
            name=name,
            age=age,
            gender=gender,
            profile_pic=profile_pic,
        )
        info.context.get('users').append(user)
        return CreateUser(user=user)

class Mutation(graphene.ObjectType):
    create_user = CreateUser.Field()
```

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

result_mutation = schema.execute(
    '''mutation {
        createUser(name:"Cersei Lannister", age:30,
gender:"Female", profilePic:"") {
            user {
                id
                name
            }
        }
    }''',
    context_value=context,
)
```



```
OrderedDict([
  ('createUser',
    OrderedDict([
      ('user',
        OrderedDict([
          ('id', '<graphql.types.scalars.ID object at
0x101bb8b70>'),
          ('name', 'Cersei Lannister')])
        ])
      )
    ])
  ])
])
```

GRAPHENE E DJANGO

Exemplo de aplicação

Demo: <https://horcut.herokuapp.com/graphql/>

Código: <https://github.com/nicysneiros/horcut>

GRAPHQL VS REST (?)

GraphQL

- › Linguagem de Consulta
- › Otimiza performance e flexibilidade
- › Não precisa de versionamento
- › Permite iterações rápidas de design do front-end

REST

- › Estilo Arquitetural
- › Utiliza protocolos existentes (HTTP)
- › Navegação entre recursos através de links (HATEOAS)
- › Melhor caching



OBRIGADA!

NICOLLE CYSNEIROS

@nicysneiros

www.labcodes.com.br

[GitHub.com/labcodes](https://github.com/labcodes)

medium.com/labcodes

speakerdeck.com/labcodes

