



mercado  
livre

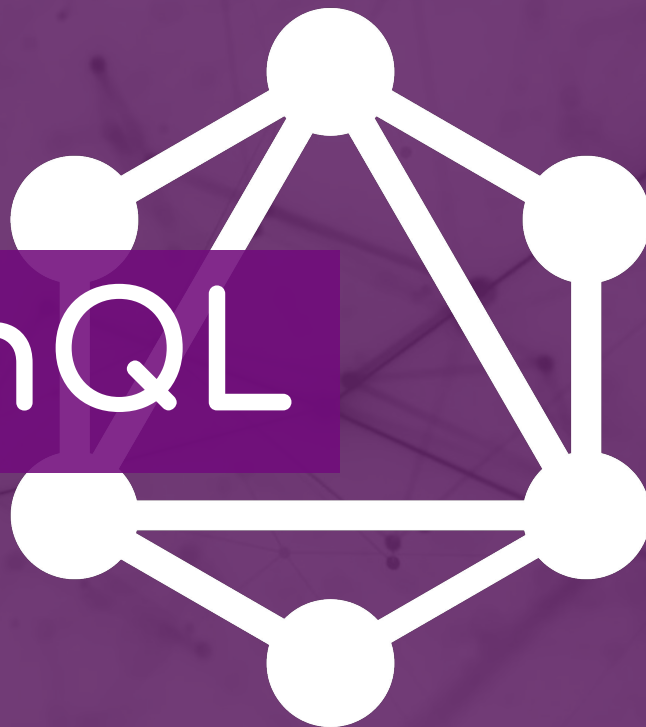
---

Por André Justi

Grupo de estudos

Intro à

GraphQL



# Agenda

- O que é GraphQL?
- O que não é GraphQL?
- Breve história
- Linguagens
- O que estão falando por aí
- Quem utiliza
- Vantagens e desvantagens
- Partes do GraphQL (SDL, Queries, Mutations, Subscriptions etc)
- Ferramentas
- Exemplos
- Referências



# O que é GraphQL?

GraphQL é uma linguagem de consulta e postagem de dados baseada em grafos para APIs que funciona em tempo de execução, fornecendo uma descrição completa e compreensível dos dados em sua API, oferecendo aos clientes dessa API o poder de perguntar exatamente o que precisam, o que torna a evolução das apis algo mais simples e sem grandes necessidades de versionamento.



# O que é não GraphQL?

## GraphQL não é um banco de dados

GraphQL não tem nenhuma relação com o banco de dados, e não é um ORM. Ele nem precisa de um banco de dados para funcionar.

## GraphQL não é um framework

Não se trata de mais um framework, em sua essência GraphQL é uma especificação que possui implementações em diversas linguagens.

## GraphQL não é exclusivo para HTTP/APIs

GraphQL não é simplesmente uma nova forma de se criar APIs com HTTP. Na verdade ele nem usa verbos HTTP, ele desconhece completamente esta camada. GraphQL não se limita ao contexto de aplicações HTTP.



# Breve história

Como e porque surgiu...

Em 2012, o Facebook começou um esforço para reconstruir suas aplicações mobile nativas, antes disso suas aplicações Android e iOS eram views da versão mobile do site. Quando essas aplicações começaram a se tornar cada vez mais complexas sua performance começou a degradar, resultando em várias quedas.



# Breve história

No processo de transição para views nativas eles perceberam pela primeira vez a necessidade de ter uma API com os dados do News Feed (que nesse ponto era entregue apenas como HTML). Algumas das opções que avaliaram foram servidores RESTful e FQL tables (que é uma API no estilo SQL do Facebook), mas que foram descartados por não representarem o modelo do Facebook e como os dados são consumido (que são nada mais que grafos de objetos com JSONs, por exemplo). Além disso seria necessário a escrita de muitas linhas de código no servidor para que o cliente conseguisse fazer a conversão.



# Breve história

Atualmente...

Atualmente GraphQL está sendo mantido pela Linux Foundation e para isso foi criado a GraphQL Foundation, o intuito disso é aumentar a colaboração para evolução da especificação e ferramentas

 THE **LINUX** FOUNDATION

+



**GraphQL**  
Foundation



# Oque estão falando por aí

Pesquisa no Google Trends de GraphQL comparado a Rest nos últimos 2 anos.



Principais termos de pesquisa:

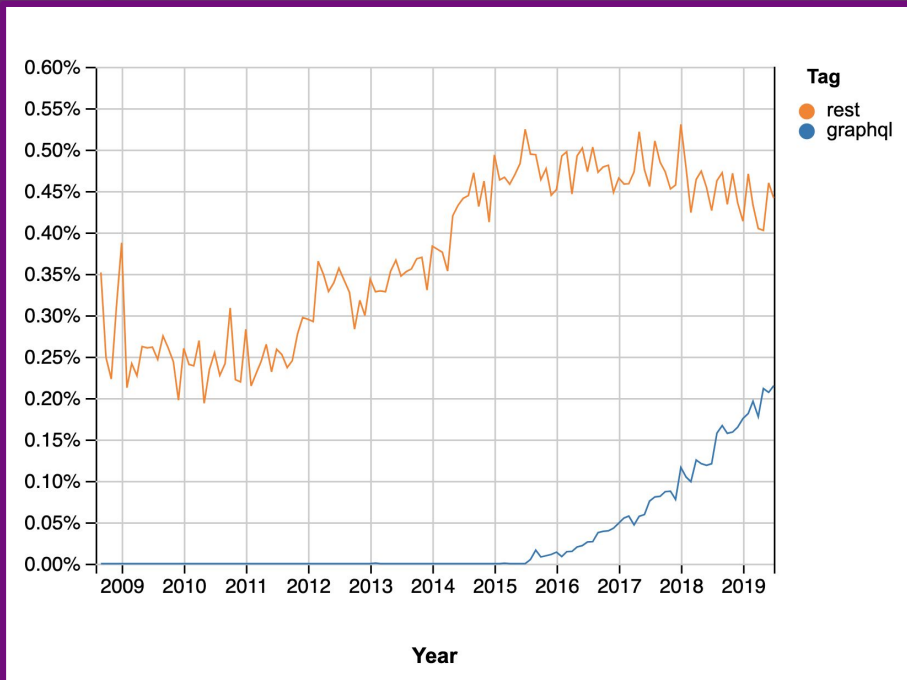
- apollo graphql
- react graphql
- query graphql
- graphql api
- graphql type
- graphql schema
- java graphql
- js graphql
- graphql vs rest
- graphql server
- graphql mutation
- graphql example
- graphql client
- github graphql
- node graphql





# Oque estão falando por aí

Pesquisa no Stack Overflow Trends de GraphQL comparado a Rest.



# Quem utiliza



...Entre outras



# Vantagens

## Flexibilidade e autonomia do cliente (frontend)

Com GraphQL o cliente solicita o que consumir, tendo em vista que as API são construídas baseadas em suas entidades de domínio, orientadas a negócio. Caso o cliente queira mais uma informação, ela provavelmente já estará lá e não será necessário alteração no backend.

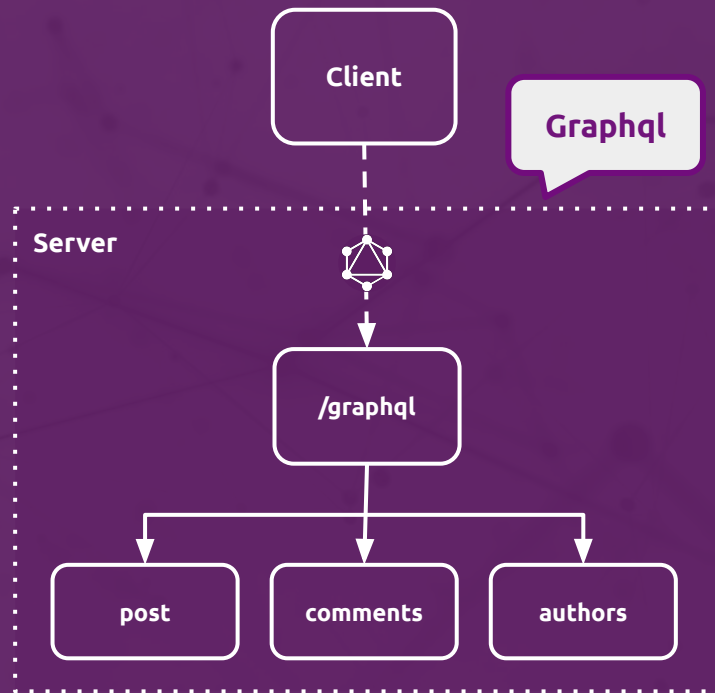
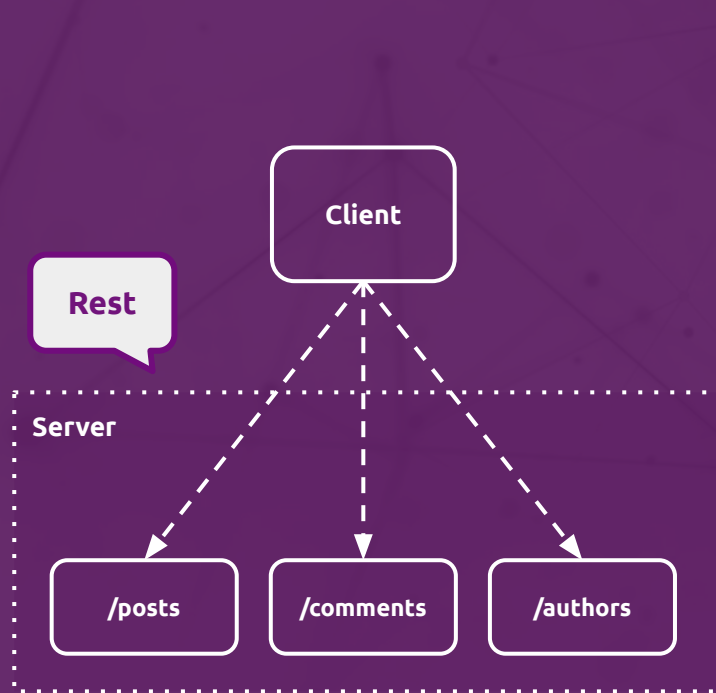
## Contratos e documentação bem definida

Como é necessário mapear os contratos/schemas para todo ponto de entrada e saída do GraphQL, ele se torna automaticamente documentado, isso pode ser visto através de playground que existem para GraphQL similares ao Swagger etc só que muito mais ricos e interativos.



# Vantagens

Evita Overfetching e Underfetching



# Vantagens

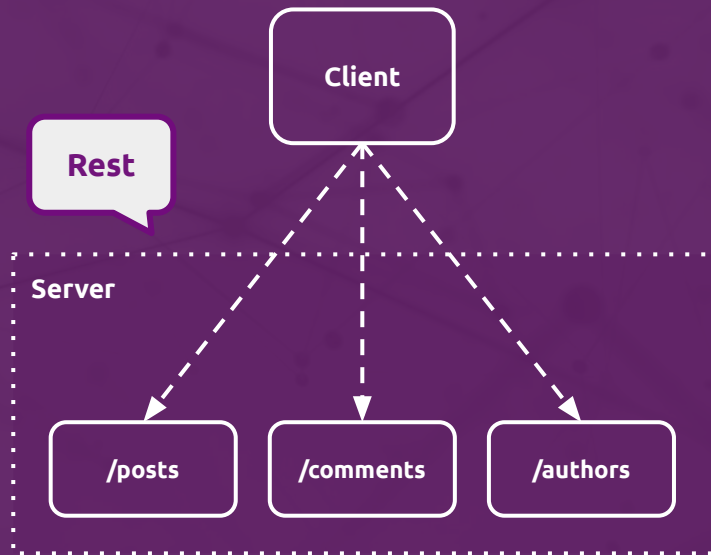
## Evita Overfetching e Underfetching

Imagine um requisito que é necessário mostrar um post (*title, desc, author*), de blog, seus comentários (*title, desc, date\_created e author*)

```
GET /posts/$id
{
  id
  title
  desc
  dateCreated
  authorId
}
```

```
GET /posts/$id/comments
{
  id
  title
  desc
  dateCreated
  authorId
}
```

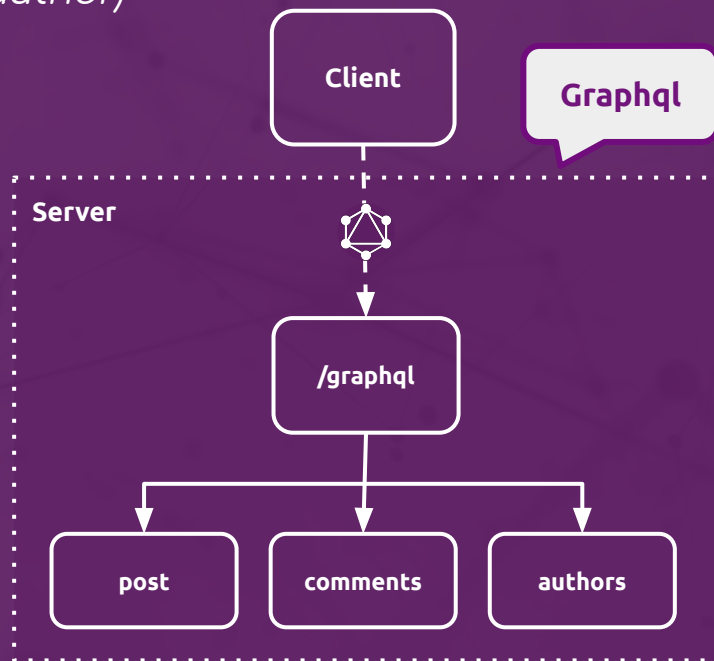
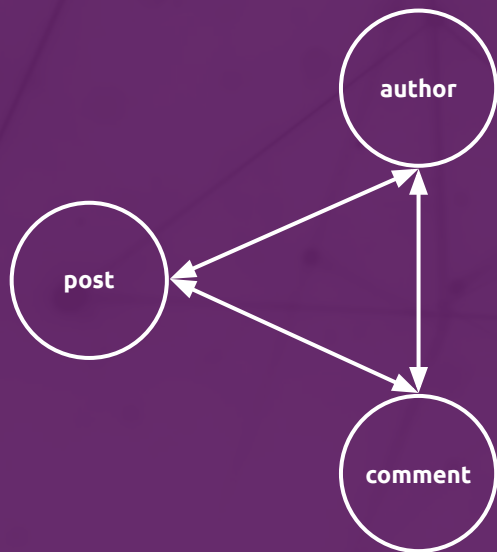
```
GET /authors/$id
{
  name
  email
}
```



# Vantagens

## Evita Overfetching e Underfetching

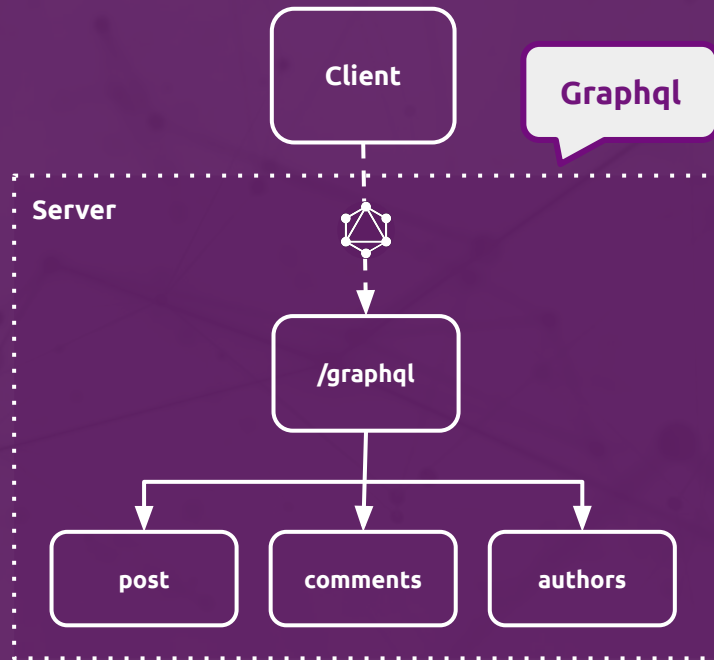
Imagine um requisito que é necessário mostrar um post (*title, desc, author*), de blog, seus comentários (*title, desc, dateCreated e author*)



# Vantagens

## Evita Overfetching e Underfetching

```
POST /graphql
{
  Post(id: "$id") {
    id
    title
    desc
    dateCreated
    author {
      name
      email
    }
    comments {
      id
      title
      desc
      dateCreated
      authorId
      author {
        name
        email
      }
    }
  }
}
```



# Desvantagens

- Complexidade
- Performance
- Caching
- Complexidade no gerenciamento de autorizações
- Difícil Limitar recursos





# Partes do GraphQL

- SDL - Schema Definition Language
- Type e Input
- Query
- Mutation
- Subscription
- Fragment
- Resolver



# SDL

## Schema Definition Language

O GraphQL tem seu próprio tipo de linguagem usado para escrever esquemas. Essa é uma sintaxe de esquema legível por humanos. O SDL será o mesmo, não importa qual tecnologia você esteja usando – você pode usá-lo com qualquer linguagem ou estrutura que desejar.

Essa linguagem de esquema é muito útil porque é simples entender quais tipos e operações sua API terá é algo análogo ao arquivo de schema do OPEN API / Swagger etc.

Em resumo SDL é a forma de definir o que a aplicação irá fazer e como será formatos de entrada e saída.



# Type / Input

Os types e inputs são um dos recursos mais importantes do GraphQL. Types e inputs são objetos personalizados, fortemente tipados que representam os pontos de entrada e saída da sua API. Por exemplo, se você está criando um aplicativo de mídia social, sua API deve ter type ou input como Posts , Users , Likes , Groups .

Eles possuem campos e esses campos retornam um type específico de dados. Por exemplo, vamos criar um tipo de usuário, devemos ter alguns campos de name , email e age . Os campos de tipo podem ser qualquer coisa e sempre retornam um tipo de dado como Int, Float, String, Boolean, ID, uma lista de tipos de objetos ou tipos de objetos personalizados é algo como o WSDL do SOAP.



# Type / Input

```
type / input Author {  
  id: ID!  
  name: String!  
  email: String!  
}
```

- Definição de tipo

```
type / input Comment {  
  id: ID!  
  text: String!  
}
```

- “!” Significa que o campo não pode ser nulo

```
type / input Post {  
  id: ID!  
  title: String!  
  desc: String  
  comments: [Comment]  
}
```

- “[ ]” Representa um campo array



# Queries

Para explicar de uma maneira simples, as consultas no GraphQL são como se obtém dados. Uma das melhores coisas do GraphQL é que você irá obter os dados exatos que deseja. Nem mais nem menos (Overfetching e Underfetching). Isso tem um enorme impacto positivo nas API

```
query {  
  getPost(id: "$id") {  
    id  
    title  
    desc  
    dateCreated  
    author {  
      name  
      email  
    }  
    comments {  
      id  
      title  
      desc  
      dateCreated  
      authorId  
      author {  
        name  
        email  
      }  
    }  
  }  
}
```



# Mutations

No GraphQL, as mutações são a maneira de modificar os dados no servidor e recuperar os dados atualizados. Você pode pensar como o CUD (Create, Update, Delete) do REST.

```
mutation {  
  DeletePost(id: "$id") {  
    deleted  
  }  
}  
  
mutation {  
  CreatePost(post: $post) {  
    id  
  }  
}
```



# Subscription

As **subscriptions** são a maneira de manter uma conexão em tempo real com um servidor. Isso significa que sempre que um evento ocorrer no servidor e sempre que esse evento for chamado, o servidor enviará os dados correspondentes ao cliente.

Ao trabalhar com assinaturas, você pode manter seu aplicativo atualizado para as últimas alterações entre diferentes usuários, isso normalmente é feito com Websocket.

```
subscription {  
  Comment(filter: $filter) {  
    id  
  }  
}
```



# Resolvers

Resolvers são funções responsáveis por, resolver um pedido e devolver o dado solicitado. É algo como a implementação de um endpoint REST.





# Ferramentas

- GraphQL
- GraphQL Voyager
- Apollo Client
- Editor GraphQL

GraphiQL

```
1 query TodoAppQuery($n: Int!) {
2   globalTodoList {
3     items(first:$n) {
4       edges {
5         node {
6           text
7           complete
8         }
9       }
10    }
11  }
12 }
```

QUERY VARIABLES


```
1 {
2   "n": 2
3 }
```


```
{
  "data": {
    "globalTodoList": {
      "items": {
        "edges": [
          {
            "node": {
              "text": "Release GraphiQL",
              "complete": true
            }
          },
          {
            "node": {
              "text": "Attend @Scale 2015",
              "complete": false
            }
          }
        ]
      }
    }
  }
}
```


GRAPHQL VOYAGER


CHANGE INTROSPECTION


Type List


Root **root**   
No Description


Film   
A single film.

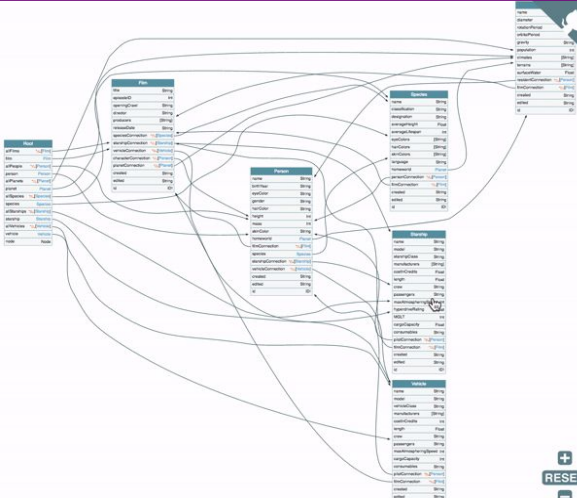
Person   
An individual person or character within the Star Wars universe.

Planet   
A large mass, planet or planetoid in the Star Wars Universe, at the time of 0 ABY.

Species   
A type of person or character within the Star Wars Universe.

Starship   
A single transport craft that has hyperdrive capability.

Vehicle   
A single transport craft that does not have hyperdrive capability



# Apenso.....

## Outros temas para estudar

- Fragments
- Interfaces
- Union types
- Aliases
- Default variables
- Directives



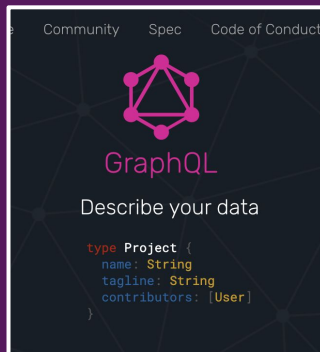
# Exemplos



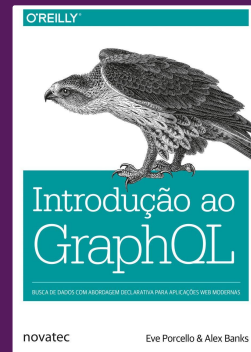
<https://github.com/justiandre/example-kotlin-springboot-graphql>



# Referências



[graphql.org](https://graphql.org)



O'Reilly  
*Introdução ao GraphQL*  
por Eve Porcello e Alex Banks



# Thanks!!!

André Justi

🐙 [/justiandre](#)

in [/in/andrejusti](#)

✉ [andre.justi@gmail.com](mailto:andre.justi@gmail.com)

Set/2019



mercado  
livre