

GraphQL



# WHAT IS GRAPHQL?

GraphQL is an application layer query language from Facebook.

GraphQL is a specification



Facebook  
Open Source



## MOTIVATION

2012

facebook



# MOTIVATION



Lee Byron

Facebook / GraphQL

Lee Byron is an Engineer at Facebook working on GraphQL. He's been making things at Facebook since 2008, including Immutable.js, Mobile & JavaScript.



## MOTIVATION

UP UNTIL 2012, NEWS FEED COULD ONLY BE REQUESTED AND DELIVERED AS HTML FROM OUR SERVERS. DURING THE EFFORT TO REBUILD NEWS FEED AS A NATIVE IOS VIEW WE HAD TO REVISIT THIS ARCHITECTURE TO GET RAW DATA

Lee Byron



# WHAT IS GRAPHQL?

<http://facebook.github.io/graphql>

## GraphQL

*Working Draft – April 2016*

### Introduction

This is a Draft RFC Specification for GraphQL, a query language created by Facebook in 2012 for describing the capabilities and requirements of data models for client-server applications. The development of this standard started in 2015. GraphQL is a new and evolving language and is not complete. Significant enhancement will continue in future editions of this specification.

### Copyright notice

Copyright (c) 2015, Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name Facebook nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

## GraphQL

- 1 Overview
- ▶ 2 Language
- ▶ 3 Type System
- ▶ 4 Introspection
- ▶ 5 Validation
- ▶ 6 Execution
- ▶ 7 Response
- ▶ A Appendix: Notation Conventions
- ▶ B Appendix: Grammar Summary



# WHAT IS GRAPHQL?

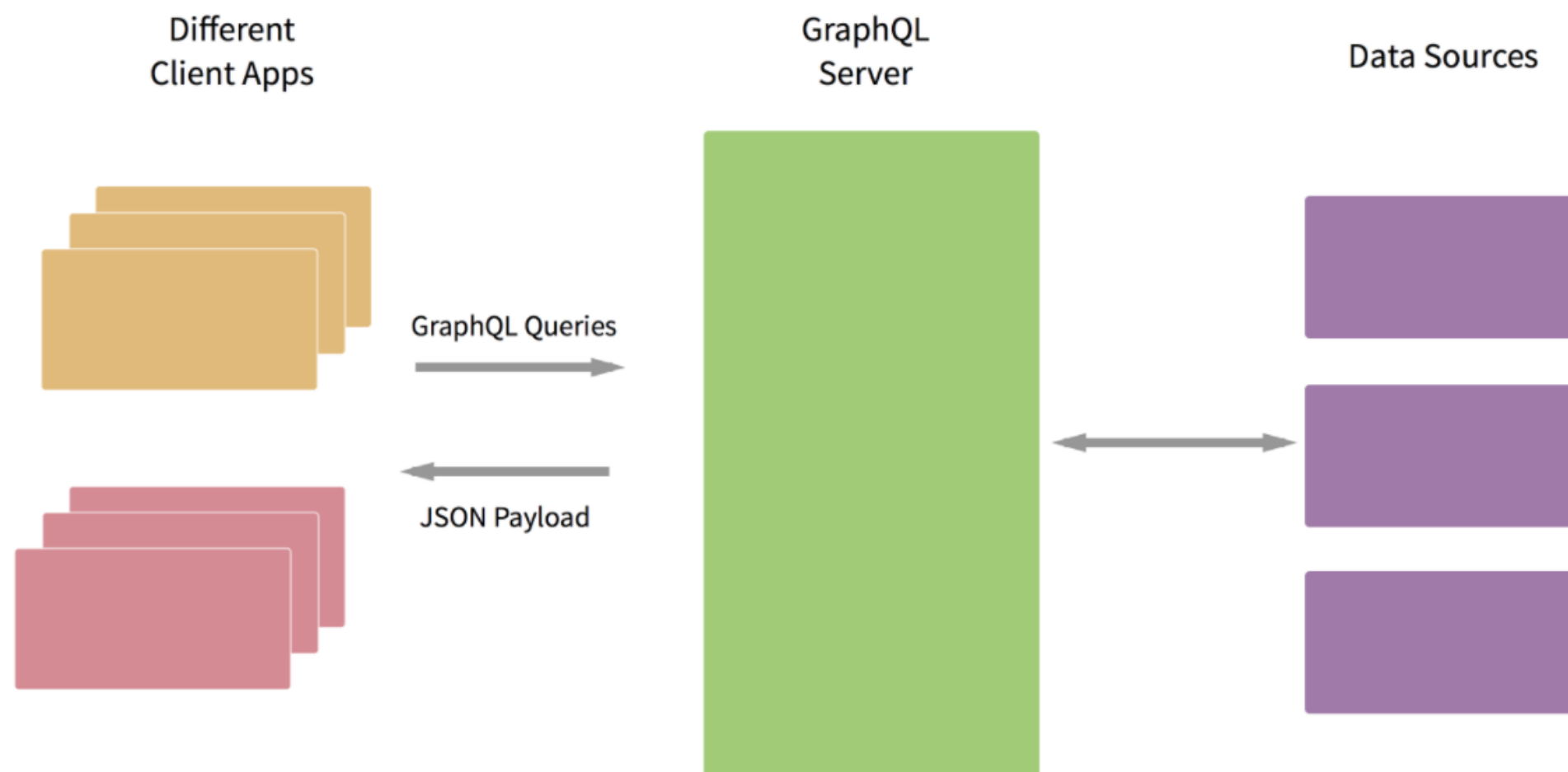
## Hello GraphQL

```
1 {  
2   hello  
3 }
```

```
{  
  "data": {  
    "hello": "world"  
  }  
}
```



# WHAT IS GRAPHQL?







## WHAT IS GRAPHQL?

With GraphQL, you can define your backend as a well-defined graph-based schema. Then client applications can query your dataset as they are needed.



## WHAT IS GRAPHQL?

So, you don't need to change your backend for data requirement changes in client apps. This simply solves one of the biggest problems in managing REST API.



## WHY GRAPHQL?

**Path  
Management  
Hell**



## WHAT IS GRAPHQL?

GraphQL also allows client applications to batch and fetch data very efficiently. For an example, have a look at the following GraphQL query:



# WHAT IS GRAPHQL?

```
{  
  latestPost {  
    _id,  
    title,  
    content,  
    author {  
      name  
    },  
    comments {  
      content,  
      author {  
        name  
      }  
    }  
  }  
}
```

This is a GraphQL query to fetch data for a blog post with comments and author information



# WHAT IS GRAPHQL?

Here's the result of the above query:

```
{
  "data": {
    "latestPost": {
      "_id": "03390abb5570ce03ae524397d215713b",
      "title": "New Feature: Tracking Error Status with Kadir",
      "content": "Here is a common feedback we received from our users ...",
      "author": {
        "name": "Pahan Sarathchandra"
      },
      "comments": [
        {
          "content": "This is a very good blog post",
          "author": {
            "name": "Arunoda Susiripala"
          }
        },
        {
          "content": "Keep up the good work",
          "author": {
            "name": "Kasun Indi"
          }
        }
      ]
    }
  }
}
```

WHAT IS GRAPHQL?

GraphQL

GraphQL▶Prettify

1 {  
2 allFilms{  
3 films {  
4 id  
5 title  
6 director  
7 }  
8 }  
9 }

{  
"data": {  
"allFilms": {  
"films": [  
{  
"id": "ZmlsbXM6MQ==",  
"title": "A New Hope",  
"director": "George Lucas"  
},  
{  
"id": "ZmlsbXM6Mg==",  
"title": "The Empire Strikes Back",  
"director": "Irvin Kershner"  
},  
{  
"id": "ZmlsbXM6Mw==",  
"title": "Return of the Jedi",  
"director": "Richard Marquand"  
},  
{  
"id": "ZmlsbXM6NA==",  
"title": "The Phantom Menace",  
"director": "George Lucas"  
},  
{  
"id": "ZmlsbXM6NQ==",  
"title": "Attack of the Clones",  
"director": "George Lucas"  
},  
{  
"id": "ZmlsbXM6Ng==",  
"title": "Revenge of the Sith",  
"director": "George Lucas"  
}  
]  
}  
}

QUERY VARIABLES

< SchemaRootX

No Description

FIELDS

allFilms(after: String, first: Int, before: String, last: Int): FilmsConnection

film(id: ID, filmID: ID): Film

allPeople(after: String, first: Int, before: String, last: Int): PeopleConnection

person(id: ID, personID: ID): Person

allPlanets(after: String, first: Int, before: String, last: Int): PlanetsConnection

planet(id: ID, planetID: ID): Planet

allSpecies(after: String, first: Int, before: String, last: Int): SpeciesConnection

species(id: ID, speciesID: ID): Species

allStarships(after: String, first: Int, before: String, last: Int): StarshipsConnection

starship(id: ID, starshipID: ID): Starship

allVehicles(after: String, first: Int, before: String, last: Int): VehiclesConnection

vehicle(id: ID, vehicleID: ID): Vehicle

node(id: ID!): Node

# WHAT IS GRAPHQL?

- ▶ Declarative Query Language
- ▶ Hierarchical
- ▶ Product-centric
- ▶ Strong-typing
- ▶ Client-specified queries



## WHAT IS GRAPHQL?

Learn GraphQL Sandbox
Docs

```

1 {
2   posts{
3     title
4     summary
5     content
6     author {
7       name
8       twitterHandle
9     }
10  }
11 }

```

```

{
  "data": {
    "posts": [
      {
        "title": "New Feature: Tracking Error Status with Kadir",
        "summary": "Lot of users asked us to add a feature to set status for errors in the Kadir Error Manager. Now, we",
        "content": "Here is a common feedback we received from our users:\n\n> Hi, I have a suggestion. It would be great",
        "author": {
          "name": "Pahan Sarathchandra",
          "twitterHandle": "@pahans"
        }
      },
      {
        "title": "Understanding Mean, Histogram and Percentiles",
        "summary": "A short guide to means, histograms and percentiles and how we can use them in a real situation.",
        "content": "This is a short guide to remind you about means, histograms and percentiles in statistics. Then, we w",
        "author": {
          "name": "Arunoda Susiripala",
          "twitterHandle": "@arunoda"
        }
      },
      {
        "title": "Introducing Kadir Debug, Version 2",
        "summary": "Today, we are introducing a new version of Kadir Debug. It comes with many UI improvements and suppo",
        "content": "Today, I'm very excited to be launching Kadir Debug version 2. This version comes with a lot of UI +",
        "author": {
          "name": "Arunoda Susiripala",
          "twitterHandle": "@arunoda"
        }
      },
      {
        "title": "Sharing the Meteor Login State Between Subdomains",
        "summary": "In this blog we'll show you how we shared login state between our static web app and our Meteor app b",
        "content": "Most developers and companies use two different apps for the marketing website and for the app itself",
        "author": {
          "name": "Kasun Indi",
          "twitterHandle": "@indi"
        }
      }
    ]
  }
}

```

QUERY VARIABLES

# WHAT IS GRAPHQL?

GraphQL is a query language for your API,  
and a server-side runtime for executing  
queries by using a type system you define  
for your data

## WHAT IS GRAPHQL?

### GraphQL Operations

- ▶ Query (GET)
- ▶ Mutation (POST/PUT/DELETE)



## STORY

**A quick story before**



## STORY

**A Frontend Dev will start a new  
App and integrate with this  
REST API**

**SWAPI**

The Star Wars API



# STORY

## Root

The Root resource provides information on all available resources within the API.

### Example request:

```
http http://swapi.co/api/
```

### Example response:

```
HTTP/1.0 200 OK
Content-Type: application/json
{
  "films": "http://swapi.co/api/films/",
  "people": "http://swapi.co/api/people/",
  "planets": "http://swapi.co/api/planets/",
  "species": "http://swapi.co/api/species/",
  "starships": "http://swapi.co/api/starships/",
  "vehicles": "http://swapi.co/api/vehicles/"
}
```

### Attributes:

- **films** *string* -- The URL root for Film resources
- **people** *string* -- The URL root for People resources
- **planets** *string* -- The URL root for Planet resources
- **species** *string* -- The URL root for Species resources
- **starships** *string* -- The URL root for Starships resources
- **vehicles** *string* -- The URL root for Vehicles resources



## STORY

**Design send the first layout  
for our Frontend Dev**



# STORY

A mockup of a mobile application interface for Star Wars. The app has a dark blue header with a hamburger menu icon, the title 'Star Wars', a search icon, and a plus icon. Below the header are three tabs: 'Films' (selected), 'Planets', and 'Species'. The main content area displays a list of four film entries, each with a red circular icon containing a number, the film title, the director's name, and the release date. The entries are: 04 A New Hope (George Lucas, 1977-05-25), 05 Attack of the Clones (George Lucas, 2002-05-16), 06 Return of the Jedi (Richard Marquand, 1983-05-25), and 01 The Phantom Menace (George Lucas, 1999-05-19).

| Icon | Title                | Director         | Release Date |
|------|----------------------|------------------|--------------|
| 04   | A New Hope           | George Lucas     | 1977-05-25   |
| 05   | Attack of the Clones | George Lucas     | 2002-05-16   |
| 06   | Return of the Jedi   | Richard Marquand | 1983-05-25   |
| 01   | The Phantom Menace   | George Lucas     | 1999-05-19   |





## STORY

<http://swapi.co/api/films/>

### Film List

OPTIONS

GET



GET /api/films/

```
{
  "count": 7,
  "next": null,
  "previous": null,
  "results": [
    {
      "title": "A New Hope",
      "episode_id": 4,
      "opening_crawl": "It is a period of civil war.\r\nRebel spaceships, disguised as ordinary freighters, have stolen more than a dozen Imperial Star Destroyers. Now there is a civil war in the galaxy.",
      "director": "George Lucas",
      "producer": "Gary Kurtz, Rick McCallum",
      "release_date": "1977-05-25",
      "characters": [
        "http://swapi.co/api/people/1/",
        "http://swapi.co/api/people/2/",
        "http://swapi.co/api/people/3/",
        "http://swapi.co/api/people/4/",
        "http://swapi.co/api/people/5/",
        "http://swapi.co/api/people/6/",
        "http://swapi.co/api/people/7/",
        "http://swapi.co/api/people/8/",
        "http://swapi.co/api/people/9/",
        "http://swapi.co/api/people/10/",
        "http://swapi.co/api/people/12/",
        "http://swapi.co/api/people/13/",
        "http://swapi.co/api/people/14/",
        "http://swapi.co/api/people/15/",
        "http://swapi.co/api/people/16/",
        "http://swapi.co/api/people/18/",
        "http://swapi.co/api/people/19/",
        "http://swapi.co/api/people/81/"
      ]
    }
  ],
}
```

```
  "planets": [
    "http://swapi.co/api/planets/2/",
    "http://swapi.co/api/planets/3/",
    "http://swapi.co/api/planets/1/"
  ],
  "starships": [
    "http://swapi.co/api/starships/2/",
    "http://swapi.co/api/starships/3/",
    "http://swapi.co/api/starships/5/",
    "http://swapi.co/api/starships/9/",
    "http://swapi.co/api/starships/10/",
    "http://swapi.co/api/starships/11/",
    "http://swapi.co/api/starships/12/",
    "http://swapi.co/api/starships/13/"
  ],
  "vehicles": [
    "http://swapi.co/api/vehicles/4/",
    "http://swapi.co/api/vehicles/6/",
    "http://swapi.co/api/vehicles/7/",
    "http://swapi.co/api/vehicles/8/"
  ],
  "species": [
    "http://swapi.co/api/species/5/",
    "http://swapi.co/api/species/3/",
    "http://swapi.co/api/species/2/",
    "http://swapi.co/api/species/1/",
    "http://swapi.co/api/species/4/"
  ],
  "created": "2014-12-10T14:23:31.880000Z",
  "edited": "2015-04-11T09:46:52.774897Z",
  "url": "http://swapi.co/api/films/1/"
}
```



# STORY

```
1
2
3  {
4    allFilms{
5      films{
6        title
7        director
8        producers
9        releaseDate
10     }
11   }
12 }
13
14
```



# STORY

```
1
2  {
3    "data": {
4      "allFilms": {
5        "films": [
6          {
7            "title": "A New Hope",
8            "director": "George Lucas",
9            "producers": [
10             "Gary Kurtz",
11             "Rick McCallum"
12           ],
13            "releaseDate": "1977-05-25"
14          },
```

## QUERYING GRAPHQL

Let's write our first GraphQL query

```
{  
  latestPost {  
    title,  
    summary  
  }  
}
```

## QUERYING GRAPHQL

```
{  
  latestPost {  
    title,  
    summary  
  }  
}
```

```
{  
  "data": {  
    "latestPost": {  
      "title": "New Feature: Tracking Error Status with Kadir",  
      "summary": "Lot of users asked us to add a feature to set status for errors in the Kadir Error Manager. Now, we've that functionality."  
    }  
  }  
}
```

# QUERYING GRAPHQL

## Nested Querying

```
{  
  posts {  
    title,  
    author {  
      name  
    },  
    summary,  
    comments {  
      content  
    }  
  }  
}
```

## QUERYING GRAPHQL

### Arguments

```
{  
  recentPosts(count: 2) {  
    title,  
    comments(limit: 1) {  
      content  
    }  
  }  
}
```

## QUERYING GRAPHQL

### Multiple fields

```
{  
  latestPost {  
    title  
  },  
  
  authors {  
    name  
  }  
}
```



## QUERYING GRAPHQL

### Assigning a result to a variable

```
{  
  latestPost {  
    title  
  },  
  
  authors {  
    name  
  },  
  
  authors {  
    _id  
  }  
}
```

## QUERYING GRAPHQL

### Assigning a result to a variable

```
{
  "data": {
    "latestPost": {
      "title": "New Feature: Tracking Error Status with Kadir"
    },
    "authors": [
      {
        "name": "Arunoda Susiripala",
        "_id": "arunoda"
      },
      {
        "name": "Pahan Sarathchandra",
        "_id": "pahan"
      },
      {
        "name": "Kasun Indi",
        "_id": "indi"
      }
    ]
  }
}
```

## QUERYING GRAPHQL

### Assigning a result to a variable

```
{  
  latestPost: latestPost {  
    title  
  },  
  
  authorNames: authors {  
    name  
  },  
  
  authorIds: authors {  
    _id  
  }  
}
```

# INVOKING MUTATIONS

Mutations are the way to change the dataset behind GraphQL. A mutation is very similar to a field in a GraphQL query, but GraphQL assumes a mutation has side effects and changes the dataset behind the schema.

# INVOKING MUTATIONS

## First mutation

```
mutation {  
  createAuthor(  
    _id: "john",  
    name: "John Carter",  
    twitterHandle: "@john"  
  ) {  
    _id  
    name  
  }  
}
```

# INVOKING MUTATIONS

## First mutation

```
{  
  "data": {  
    "createAuthor": {  
      "_id": "john",  
      "name": "John Carter"  
    }  
  }  
}
```

# INVOKING MUTATIONS

## Multiple mutations

```
{
  "data": {
    "sam": {
      "_id": "sam",
      "name": "Sam Hautom"
    },
    "chris": {
      "_id": "chris",
      "name": "Chris Mather"
    }
  }
}
```

# FRAGMENTS

Fragments are the way to group commonly used fields and reuse them.

```
{
  arunoda: author(_id: "arunoda") {
    _id,
    name,
    twitterHandle
  },
  pahan: author(_id: "pahan") {
    _id,
    name,
    twitterHandle
  },
  indi: author(_id: "indi") {
    _id,
    name,
    twitterHandle
  }
}
```



## FRAGMENTS

So check this query. It's the same as above, but with fragments:

```
{
  arunoda: author(_id: "arunoda") {
    ...authorInfo
  },
  pahan: author(_id: "pahan") {
    ...authorInfo
  },
  indi: author(_id: "indi") {
    ...authorInfo
  }
}

fragment authorInfo on Author {
  _id,
  name,
  twitterHandle
}
```

# FRAGMENTS

## Fragments with nested fragments

```
{
  post1: post(_id: "03390abb5570ce03ae524397d215713b") {
    ...postInfo
  },
  post2: post(_id: "0176413761b289e6d64c2c14a758c1c7") {
    ...postInfo
  }
}
```

```
fragment postInfo on Post {
  title,
  content,
  author {
    ...authorInfo
  },
  comments {
    content,
    author {
      ...authorInfo
    }
  }
}
```

```
fragment authorInfo on Author {
  _id,
  name
}
```

## QUERY VARIABLES

### Using query variables

```
query getFewPosts($postCount: Int!) {  
  recentPosts(count: $postCount) {  
    title  
  }  
}
```

## QUERY VARIABLES

## Using query variables

The screenshot shows the GraphQL Sandbox interface at `sandbox.learngraphql.com`. The interface is divided into three main sections: a query editor, a query variables editor, and a response viewer.

**Query Editor:** The query is defined as follows:

```
1 query getFewPosts($postCount: Int!) {  
2   recentPosts(count: $postCount) {  
3     title  
4   }  
5 }
```

**Query Variables Editor:** The variables are defined as follows:

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

**Response Viewer:** The JSON response is displayed on the right side:

```
{  
  "data": {  
    "recentPosts": [  
      {  
        "title": "New Feature: Tracking Error Status with Kadir"   
      },  
      {  
        "title": "Understanding Mean, Histogram and Percentiles"  
      }  
    ]  
  }  
}
```