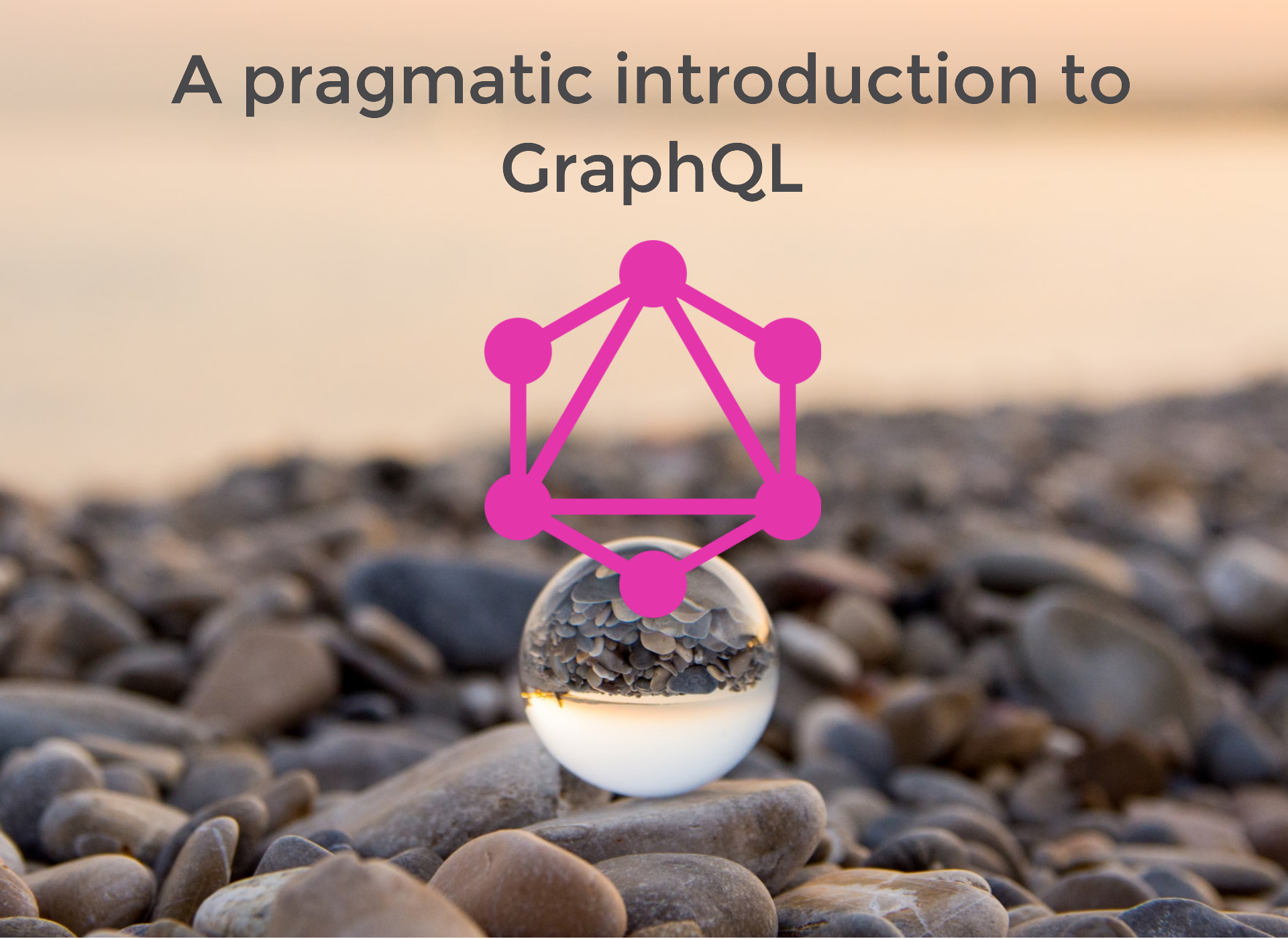
A pragmatic introduction to GraphQL



George Aidonidis

Software Engineer - Currently working as a frontend developer and in ❤️ with Javascript & React.

https://github.com/george-aidonidis

https://twitter.com/geoaido

george.aidonidis@gmail.com

What is GraphQL

A REST alternative

designed by Facebook

But why?

Multiple trips to get our data

Over fetched payload

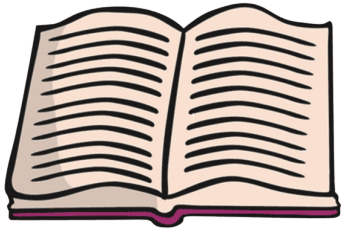
Facebook is behind it

Yes and no

It's just a specification

QL stands for Query Language

To use it we need an implementation



Anatomy 

Anatomy 🧐

```
query {
  getVideo(id: 1234) {
    title
    duration
    thumbnail {
      small
    }
  }
} # Here is a comment 🫘
```

Rest

```
GET https://example.dev/api/v1/media/id=1234
```

Response

```
{
  "data": {
    "getVideo": {
      "id": 1234,
      "title": "How to pet a lion 😺",
      "duration": 610,
      "thumbnail": {
        "small": "https://example.dev/thumb.jpg"
      }
    }
  }
}
```

Rest

Some 5KB JSON payload with everything about video with id 1234...

Query exchange

HTTP Request

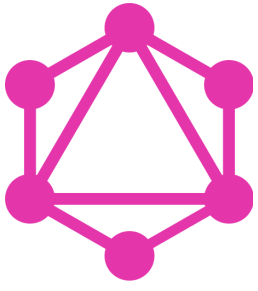
POST https://api.example.dev/graphql

Request payload: `{"query": "..."}`

Query exchange

```
curl \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{ "query": "{ getVideo(id: 1234) { title } }" }'
  https://api.example.dev/graphql
```

Query exchange

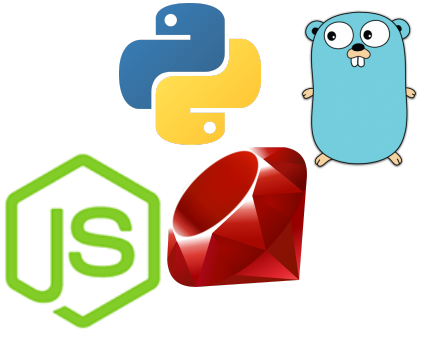


Schema

Or graphql's way to define types

Query exchange

Resolvers



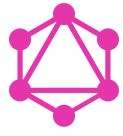
Just tell them what to return 😬

Query exchange

HTTP Request

POST https://api.example.dev/graphql

Request payload: `{"query": "..."}`



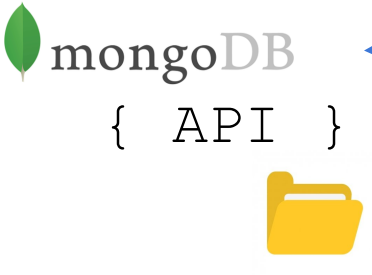
Schema

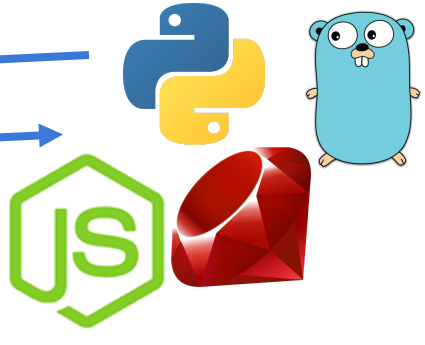
External sources

Resolvers

HTTP Response

Response payload:





```
{
  "data": {...}
}
```

Schema

Lets define a book

```
1 type Book {
2     id: String
3     title: String
4 }
```

How would the query look

```
query {
  books {
    title
    id
  }
}
```

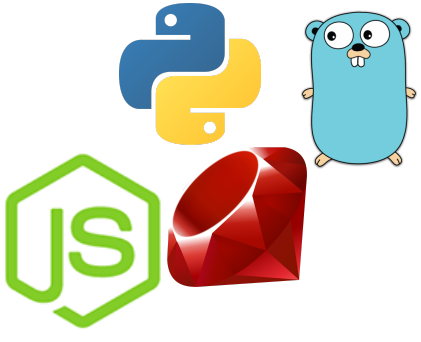
Schema

Lets define the **Query** schema

```
1 type Book {
2   id: String
3   title: String
4 }
5
6 type Query {
7   books: [Book]
8 }
```

Resolvers

Logic to get/edit our data



A Resolver will return our books

```
1 resolvers = {
2   Query: {
3     books: () -> {
4       return find(
5         '* in books'
6       )
7     }
8   }
9 }
```

Response

```
query {
  books {
    title
    id
  }
}
```

```
{
  "data": {
    "books": [
      {
        "title": "Harry Potter",
        "id": 123
      },
      {
        "title": "Jurassic Park",
        "id": 345
      }
    ]
  }
}
```

"Hey, don't we need authors?"

Schema

Lets define the Author schema

```
1 type Author {
2     name: String
3     id: Int
4     books: [Int]
5 }
```

Schema

```
type Book {
  id: String
  title: String
  author: Author
}
```

```
type Author {
  name: String
  id: Int
  books: [Int]
}
```

Resolvers

```
query {
  books {
    title
    id
    author {
      name
      id
    }
  }
}
```

```
resolvers = {
  Query: {
    books: () -> {
      return find(
        '* in books'
      )
    }
  },

  Book: {
    author: book -> {
      return find(
        'author in authors'
        where
        'author.bookId == book.id'
      )
    }
  }
}
```

Response

```
query {
  books {
    title
    id
    author {
      name
      id
    }
  }
}
```

```
{
  "data": {
    "books": [
      {
        "title": "Harry Potter",
        "id": 123,
        "author": {
          "name": "J.K. Rowling",
          "id": 1
        }
      },
      {
        "title": "Jurassic Park",
        "id": 345,
        "author": {
          "name": "Michael Crichton",
          "id": 2
        }
      }
    ]
  }
}
```

"What about reviews?"

Lets update our schema

Schema

Define a **Review** schema

```
1 type Review {
2     id: Int
3     score: Int
4     bookId: Int
5 }
```

Schema

```
type Book {
  id: String
  title: String
  author: Author
  review: Review
}
```

```
type Review {
  id: Int
  score: Int
  bookId: Int
}
```

Resolvers

```
query {
  books {
    author {
      name
      id
      author {
        name
        id
      }
      review {
        score
        id
      }
    }
  }
}
```

```
resolvers = {
  Query: {
    books: () -> {
      return find(
        '* in books'
      )
    }
  },

  Book: {
    author: book -> {
      return find(
        'author in authors'
         where
        'author.bookId == book.id'
      )
    },
    review: book -> {
      return find(
        'review in reviews'
         where
        'review.bookId equals book.id'
      )
    }
```

Resolvers

```
query {
  books {
    author {
      name
      id
      author {
        name
        id
      }
      review {
        score
        id
      }
    }
  }
}
```

```
resolvers = {
  Query: {
    books: () -> {
      return find(
        '* in books'
      )
    }
  },

  Book: {
    author: book -> {
      return find(
        'author in authors'
         where
        'author.bookId == book.id'
      )
    },
    review: book -> {
      return find(
        'review in reviews'
         where
        'review.bookId equals book.id'
      )
    }
```

Resolvers

```
query {
  books {
    author {
      name
      id
      author {
        name
        id
      }
      review {
        score
        id
      }
    }
  }
}
```

```
resolvers = {
  Query: {
    books: () -> {
      return find(
        '* in books'
      )
    }
  },

  Book: {
    author: book -> {
      return find(
        'author in authors'
         where
        'author.bookId == book.id'
      )
    },
    review: book -> {
      return find(
        'review in reviews'
         where
        'review.bookId equals book.id'
      )
    }
```

Resolvers

```
query {
  books {
    author {
      name
      id
      author {
        name
        id
      }
      review {
        score
        id
      }
    }
  }
}
```

```
resolvers = {
  Query: {
    books: () -> {
      return find(
        '* in books'
      )
    }
  },

  Book: {
    author: book -> {
      return find(
        'author in authors'
         where
        'author.bookId == book.id'
      )
    },
    review: book -> {
      return find(
        'review in reviews'
         where
        'review.bookId equals book.id'
      )
    }
```

Response

```
{
  "data": {
    "books": [
      {
        "title": "Harry Potter",
        "author": {
          "name": "J.K. Rowling"
        },
        "review": {
          "score": 10
        }
      },
      {
        "title": "Jurassic Park"
        "author": {
          "name": "Michael Crichton"
        },
        "review": {
          "score": 5
        }
      }
    ]
  }
}
```

But, what if we only need

only the authors

Resolvers

```
query {
  books {
    author {
      name
      id
    }
  }
}
```

```
resolvers = {
  Query: {
    books: () -> {
      return find(
        '* in books'
      )
    }
  },

  Book: {
    author: book -> {
      return find(
        'author in authors'
        where
        'author.bookId == book.id'
      )
    },
    review: book -> {
      return find(
        'review in reviews'
        where
        'review.bookId equals book.id'
      )
    }
```

Arguments

```
query {
  getBook(id: 123) {
    title
    author {
      name
    }
  }
}
```

Arguments

```
query {
  getBook(id: 123) {
    title
    author {
      name
    }
  }
}
```

Arguments

Schema

```
type Query {
  getBook(
    id: Int!
  ): Book

  books: [Book]
}
```

Arguments

Schema

```
type Query {
  getBook(
    id: Int!
  ): Book

  books: [Book]
}
```

```
query {
  getBook(id: 123) {
    id
    title
    author {
      name
    }
  }
}
```

Resolver

```
Query: {
  getBook: args => find(
    'book in books'
     where
    'book.id == args.id'
  )
}
```

Arguments

Schema

```
type Query {
  getBook(
    id: Int!
  ): Book

  books: [Book]
}
```

```
query {
  getBook(id: 123) {
    id
    title
    author {
      name
    }
  }
}
```

Resolver

```
Query: {
  getBook: args => find(
    'book in books'
     where
    'book.id == args.id'
  )
}
```

Arguments

Schema

```
type Query {
  getBook(
    id: Int!
  ): Book

  books: [Book]
}
```

```
query {
  getBook(id: 123) {
    id
    title
    author {
      name
    }
  }
}
```

Resolver

```
Query: {
  getBook: args => find(
    'book in books'
     where
    'book.id == args.id'
  )
}
```

Or both

```
query {
  books {
    title
    id
  }
  getBook(id: 123) {
    id
    title
    author {
      name
    }
  }
}
```

Or both

```
query {
  books {
    title
    id
  }
  getBook(id: 123) {
    id
    title
    author {
      name
    }
  }
}
```

Multiple queries

```
query {
  getBooks(genre: COMIC) {
    id
    title
    genre
    author {
      name
    }
  }

  getBooks(genre: FANTASY) {
    id
    title
    genre
    author {
      name
    }
  }
```

Complex response

```
{
  "data": {
    "books": [
      {
        "id": 4356,
        "title": "Infinity space",
        "genre": "COMIC",
        "author": {
          "name": "Joe Doe"
        }
      },
      {
        ...
        "genre": "FANTASY"
      },
      ...
    ]
  }
```

There must be a better way

Alias

Nickname your queries 😎

Alias

```
query {
  top10ComicBooks: getBooks(genre: COMIC) {
    id
    title
    author {
      name
    }
  }

  top10FantasyBooks: getBooks(genre: FANTASY) {
    id
    title
    author {
      name
    }
  }
}
```

Alias

```
query {
  top10ComicBooks: getBooks(genre: COMIC) {
    id
    title
    author {
      name
    }
  }

  top10FantasyBooks: getBooks(genre: FANTASY) {
    id
    title
    author {
      name
    }
  }
}
```

Alias

```
query {
  top10ComicBooks: getBooks(genre: COMIC) {
    id
    title
    author {
      name
    }
  }

  top10FantasyBooks: getBooks(genre: FANTASY) {
    id
    title
    author {
      name
    }
  }
}
```

Alias

```
query {
  top10ComicBooks: getBooks(genre: COMIC) {
    id
    title
    author {
      name
    }
  }

  top10FantasyBooks: getBooks(genre: FANTASY) {
    id
    title
    author {
      name
    }
  }
}
```

```
{
  "data": {
    "top10ComicBooks": [
      "id": 4356,
      "title": "Infinity space",
      "author": {
        "name": "Joe Doe"
      },
      //...
    ]

    "top10FantasyBooks": [
      //...
    ]
  }
}
```

Operations

Query

Mutation

Subcription

Mutation

```
mutation {
  updateComicBook(id: 1234, title: "Finite space")
    id
    title
    author {
      name
    }
  }
}
```

Mutation

```
mutation {
  updateComicBook(id: 1234, title: "Finite space")
    id
    title
    author {
      name
    }
  }
}
```

Mutation

```
mutation {
  updateComicBook(id: 1234, title: "Finite space")
    id
    title
    author {
      name
    }
  }
}
```

Mutation

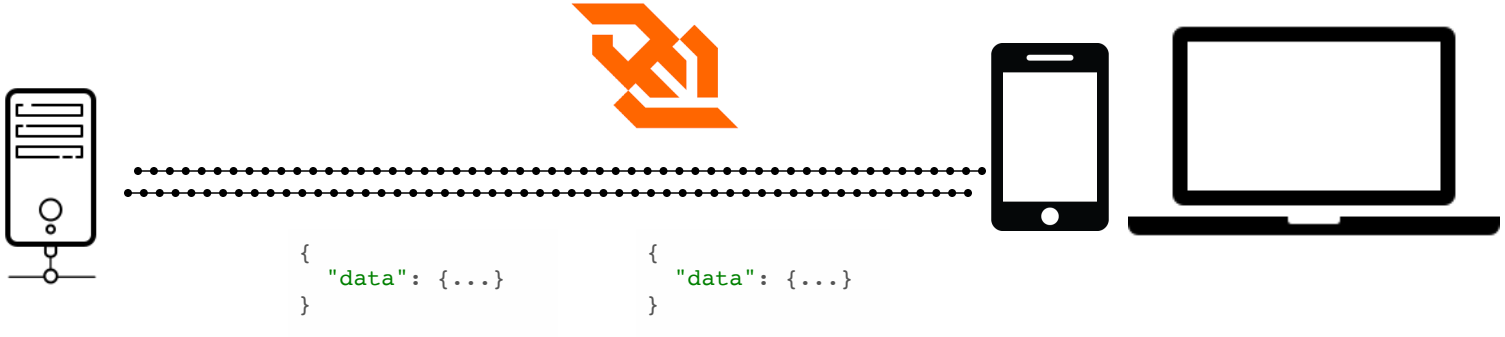
```
mutation {
  updateComicBook(id: 1234, title: "Finite space")
    id
    title
    author {
      name
    }
  }
}
```

Subscription

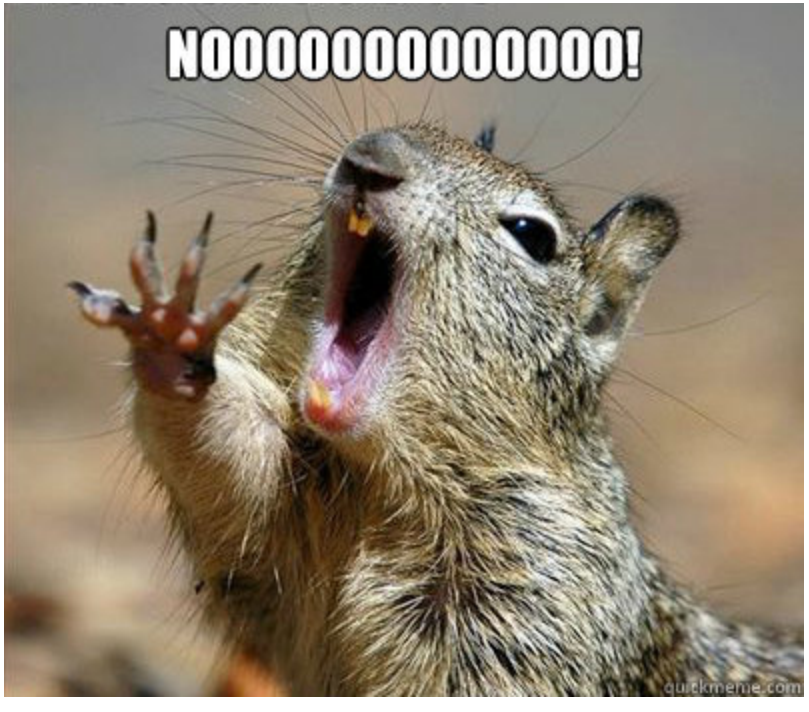
```
type Subscription {
  tweetLiked: Tweet
  newTweet: Tweet
}

type Tweet {
  id: Int!
  content: String!
}
```

```
subscription {
  tweetLiked {
    id
  }
  newTweet {
    id
    content
  }
}
```



Errors



Errors

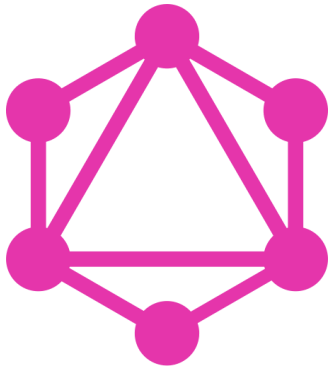
Status Codes

do not

describe errors

Errors

```
{
  "errors": [
    {
      "message": "Author for book with ID 1002 could not be fetched.",
      "locations": [ { "line": 11, "column": 7 } ],
      "path": [ "books", 1, "author" ]
    }
  ],
  {
  "data": {
    "books": [
      {
        "title": "Harry Potter",
        "author": {
          "name": "J.K. Rowling"
        },
        "review": {
          "score": 10
        }
      },
      {
        "title": "Jurassic Park",
        "author": null,
        "review": {
          "score": 5
        }
      }
    ]
  }
}
```



GraphQL

- Reduce over-fetching
- Typed system
- Documentation is a first-class citizen
- No back & forth requests
- Respond with specific data
- Available through a single endpoint


THANK YOU FOR
YOUR ATTENTION
PLEASE CLAP AND
DON'T ASK TOUGH QUESTIONS