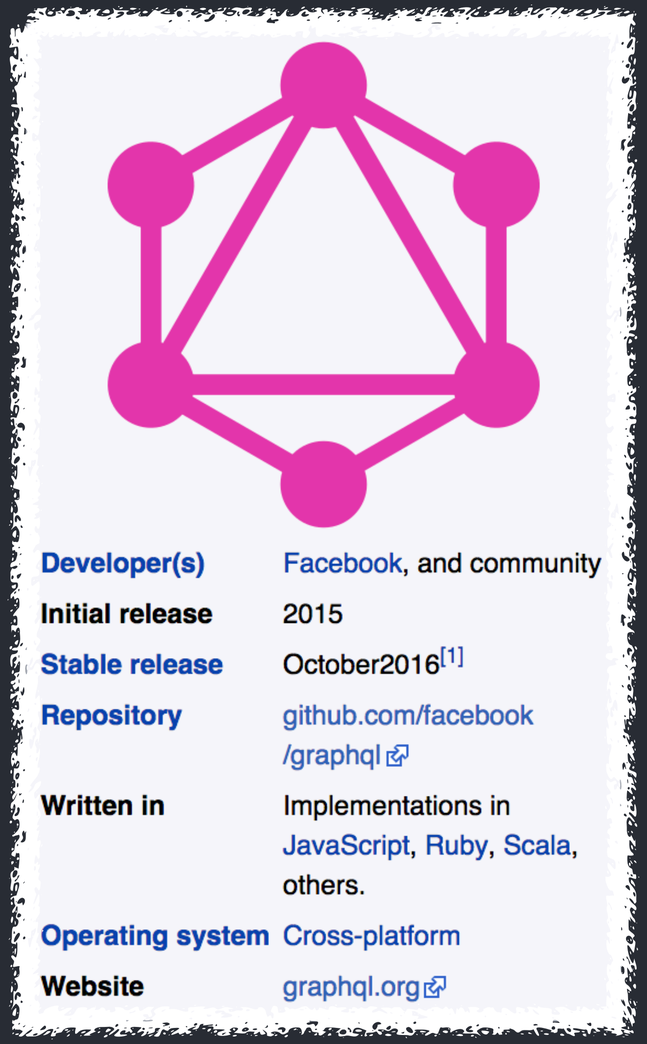
GraphQL

A REST alternative

Background - What is GraphQL?



Developer(s)	Facebook, and community
Initial release	2015
Stable release	October2016[1]
Repository	github.com/facebook/graphql
Written in	Implementations in JavaScript, Ruby, Scala, others.
Operating system	Cross-platform
Website	graphql.org

A query language for your API

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

— https://graphql.org/ —

TLDR

1. A data query language
2. An alternative to traditional REST APIs
3. A language-agnostic spec for APIs

Who uses it?







OICR
Ontario Institute
for Cancer Research

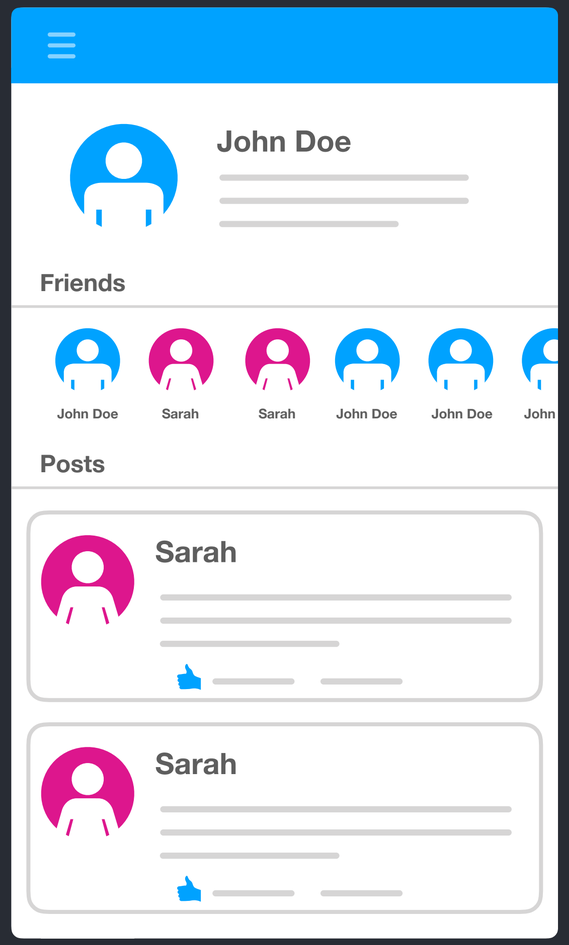

coursera

dailymotion

yelp

The problem

Traditional REST API model



'user/1234

```
{
  "id": 1234,
  "name": "John Doe",
  "email": "johndoe@email.com",
  "age": 23,
  "description": "some guy who like to code"
}
```

'/1234/friends

```
{
  "id": 2341,
  "name": "Justin Timberlake",
  "email": "justin@email.com",
  "age": 70,
  "description": "dude who sings"
},
{
  "id": 3453,
  "name": "Justin Richardson",
  "email": "justinsane@email.com",
  "age": 12,
  "description": "dude who codes"
},
...
]
```

/user/1234/posts

```
[
  {
    "text": "I'm feeling happy",
    "likes": [ {"user": "Steve"}, {"user": "Someone"} ],
    "comments": [
      { "author": "Steve", "text": "Me too" }
      { "author": "John", "text": "What's up" }
    ]
  },
  ...
]
```

- takes 3 network requests

- under & over fetching

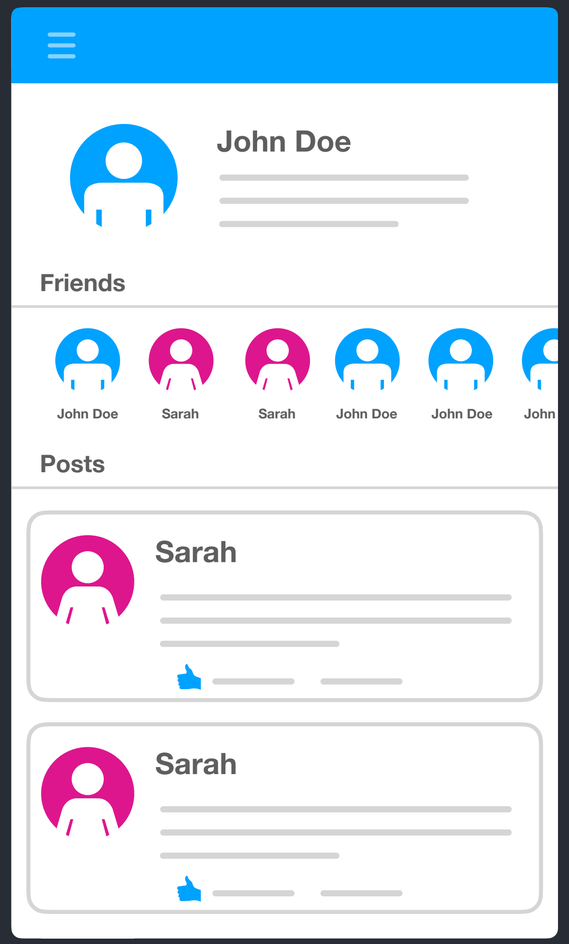
- how do you know what endpoint to hit?

- how do you know what data will be received?

...

The solution

GraphQL



/graphql

```
Query {
  user(id: 1234) {
    id
    name
    description
    friends {
      name
    }
    posts {
      text
      likes
    }
  }
}
```

```
{
  "id": 1234,
  "name": "John Doe",
  "description": "some guy who likes to code",
  "friends": [
    { "name": "Justin Timberlake"},
    { "name": "Justin Richardson"},
    ...
  ],
  "posts": [
    {
      "text": "I'm feeling happy",
      "likes": [{"user": "Steve"}, {"user": "Someone"}]
    },
    ...
  ],
}
```

- single end point to serve all data needs
- a standardized query language to perform "Queries" and "Mutations"
- a typed schema system

Languages with available implementation libraries

And more…

Developer Tool - GraphiQL

An in-browser “IDE” for exploring GraphQL APIs.
In other words, it’s Postman for GraphQL.

GraphiQL ▶ Prettify History

```
query($productId: String){
  product(id: $productId) {
    manufacturer {
      name
      products {
        id
      }
    }
  }
}
```

QUERY VARIABLES

```
{
  "productId": "iphone"
}
```

```
{
  "data": {
    "product": {
      "manufacturer": {
        "name": "Apple",
        "products": [
          {
            "id": "iphone"
          },
          {
            "id": "ipod"
          }
        ]
      }
    }
  }
}
```

Schema **Query** ×

Search Query...

No Description

FIELDS

product(id: String): Product

manufacturer(id: String): Manufacturer

customer(id: String): Customer

Core concept - Schema

schema /ˈskēmə/

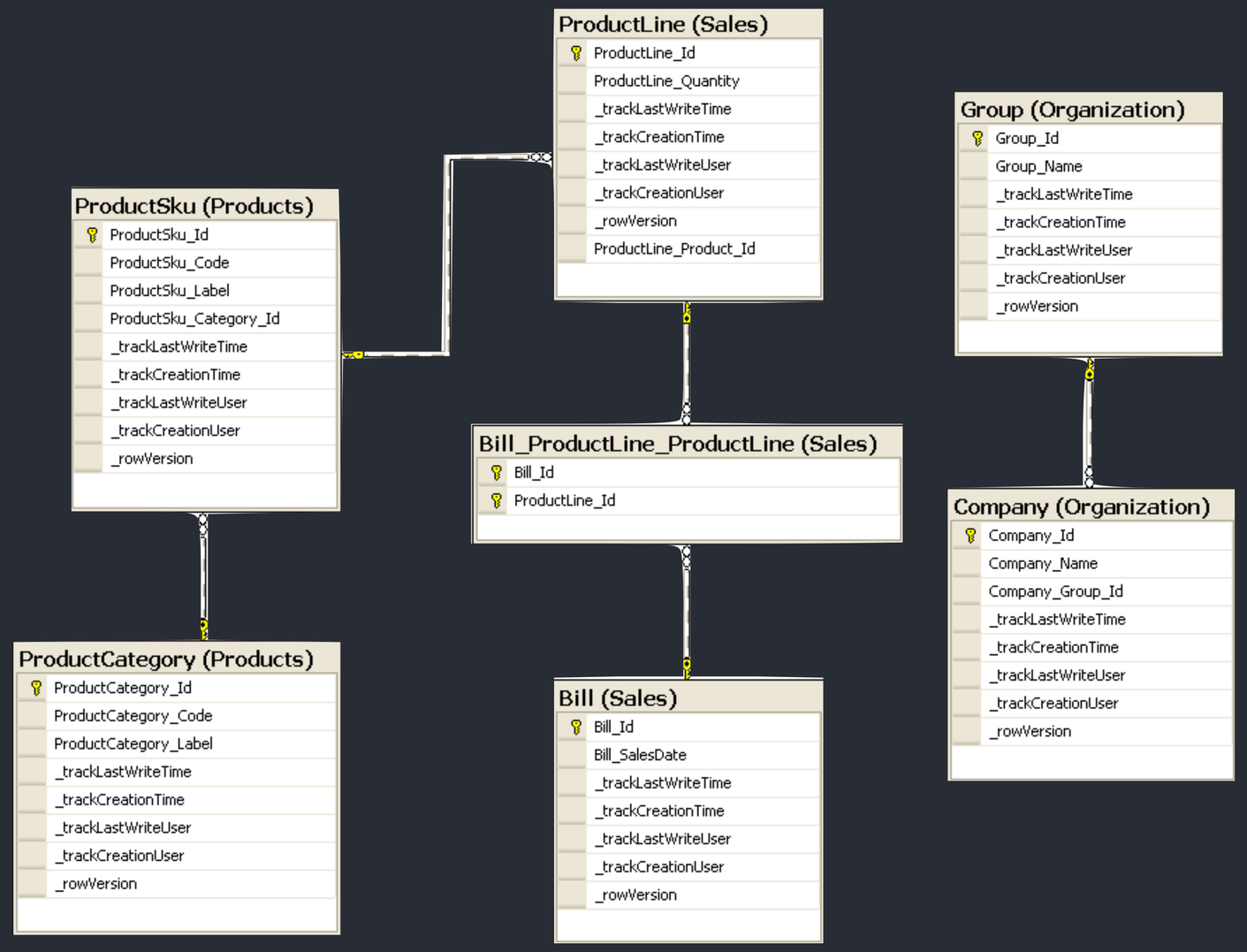
a representation of a plan or theory in the form of an outline or model.

In computer science, it is a blueprint that describes the shape of a data model, and is the “result” of data modelling.

Schemata are usually associated with but are not exclusive to databases.

Schemata in the wild

A SQL schema



Schemata in the wild

An Elasticsearch “mapping”

```
{
  "mapping": {
    "author": {
      "properties": {
        "name": {
          "type": "string"
        },
        "books": {
          "type": "nested",
          "properties": {
            "price": {
              "type": "integer"
            },
            "name": {
              "type": "string"
            },
            "title": {
              "type": "string"
            }
          }
        }
      }
    }
  }
}
```

A Graphql schema

```
type Author{
  name: String
}

type Quote {
  text: String
  author: Author
}

type Query {
  randomQuote: Quote
}
```

Anatomy of a GraphQL schema - Query & Types

```
type Author{
  name: String
}

type Quote {
  text: String
  author: Author
}

type Query {
  randomQuote: Quote
}
```

- Query describes what fields you can **“read”** from the API. Each field is of a certain Type.

- Types describe the set of possible data you can query for a given field. They can be made up of other Types to nest more fields under them. They are **<u>sent from the API to the client</u>**.

Example full response

```
{
  "data": {
    "randomQuote": {
      "text": "It is always the simple that produces the marvelous.",
      "author": {
        "name": "Amelia Barr"
      }
    }
  }
}
```

DEMO

Query & Types

Anatomy of a GraphQL schema - Mutation & Inputs

```
input UserInput {
  name: String!
  favoriteColor: String!
}

input FriendshipInput {
  from: String!
  to: String!
}

type Mutation {
  newUser(userData: UserInput!): User
  newFriendship(friendship: FriendshipInput): Friendship
}
```

- Mutation describes what you can **“write”** to the API. Mutations can take Inputs and respond with Types

- Inputs are like Types but are sent **<u>from the client to the API</u>**.

DEMO

Mutation & Inputs

Challenge Exercises - Warm Up

http://**localhost**:3000/randomQuotes/graphiql_game

Random Quote Generator

Query for a random quote and the name of its author

Challenge Exercises - Social Network

http://**localhost**:3000/socialNetwork/graphiql_game

Requirements

- Query for Justin's name, his favourite colour, and his friends
- For his friends, query for their names and their friends' favourite colour
- Achieve this in one query


```
{
  "data": {
    "user": {
      "name": "Justin",
      "favoriteColor": "red",
      "friends": [
        {
          "name": "John",
          "friends": [
            {
              "favoriteColor": "orange"
            },
            {
              "favoriteColor": "green"
            },
            {
              "favoriteColor": "red"
            }
          ]
        }
      ]
    }
  }
}
```

Challenge Exercises - Inventory System

http://**localhost**:3000/inventorySystem/graphiql_game

Requirements

- Query for Vineet and Alex's purchases
- For each purchase, retrieve the product name and the manufacturer
- For each manufacturer, retrieve the name and the name of the products they make
- Achieve all of this in one query, Vineet's data should be under the key "vineet", Alex's data under "alex"


```
{
  "data": {
    "vineet": {
      "purchases": [
        {
          "name": "MacBook Pro",
          "manufacturer": {
            "name": "Apple",
            "products": [
              {
                "name": "iPhone"
              },
              ...
            ]
          }
        },
        ...
      ]
    },
    "alex": {
      "purchases": [
        ...
      ]
    }
  }
}
```

Challenge Exercises - Social Network (mutations)

http://**localhost**:3000/_mutation_socialNetwork/graphiql_game

Let's build a social network dataset!

- Add yourself to the social network
- Add the person to the right and left of you as friends
- Query for your friends and your friends' friends

So why GraphQL again?

1

Query exactly what is needed

Reduced network traffic & more API reusability

Happier end-user

2

Self-documented APIs

Easier client-side integration

Happier UI devs

3

Built-in type checking

Only clean data hits server logic

Happier back-end devs

We just scratched the surface…

Schema

Query

Mutations

Fragments

Subscription

Variables

Custom scalar

Server implementations

Client libraries

…

Cool GraphQL resources

Some cool GraphQL APIs

https://github.com/APIs-guru/graphql-apis/blob/master/README.md

Apollo Launchpad - a “codepen” for prototyping Graphql APIs

https://launchpad.graphql.com/new

How to GraphQL - The Full Stack tutorial for GraphQL

https://www.howtographql.com/