# What is GraphQL?

A brief introduction to GraphQL along with features, benefits and how it differs from the commonly used REST architecture.

WE'LL COVER THE FOLLOWING



## GraphQL: A History #

**GraphQL** is an open source query language created by Facebook. Before GraphQL went open source in 2015, Facebook had used it internally for their mobile applications since 2012 as an alternative to the common REST architecture. As a result, network usage was reduced dramatically for Facebook's mobile applications because GraphQL made it more efficient with data transfers.

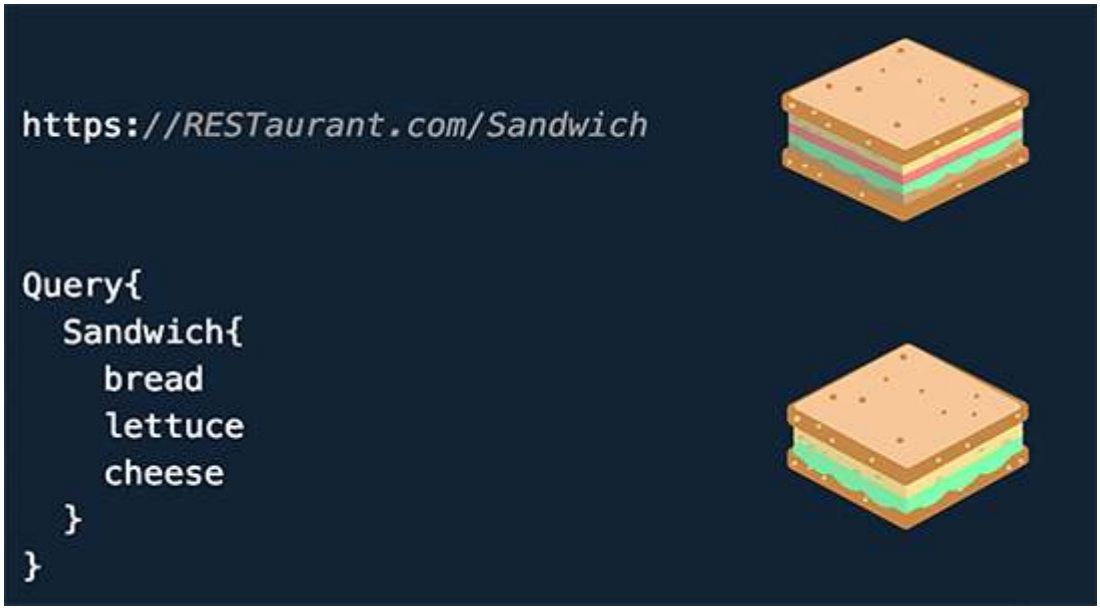
Let's say you want to fetch data from the Rest API and you say, "Hey REST API, give me the titles of the available courses on Educative". How this works is that you have a specific endpoint or URL (in our case, Educative) that you are hitting and that URL determines what data comes back. With the REST API, you fetch a URL and that URL is returning typically something like JSON or a Javascript object full of data. This results in either unwanted data that we must filter through to fetch our required data or multiple trips (requests) to cater to different queries. However, GraphQL is different.

## How is GraphQL Different?

Instead of an API where you hit a URL and accept whatever data comes back, GraphQL allows you to ask for specific data, giving clients more control over what information is sent.

### The Sandwich Comparison

Think of it like this; you want a sandwich with only bread, cheese, cucumbers, and lettuce. You walk into a RESTaurant where the only option on the menu is 'sandwich'; you place an order and receive a sandwich with bread, salami, lettuce, tomatoes, cucumbers, and cheese. You then remove everything you don't want, to be able to eat the sandwich you wanted; this is how the REST API works. However, when you visit GraphQL cafe, you realize you can specify which toppings you want in your sandwich and receive exactly what you wanted.




### Overfetching

In the RESTful architecture, the backend defines what data is available for each resource on each URL, while the frontend always has to request all the information in a resource, even if only a part of it is needed.

In the worst case scenario, a client application has to read multiple resources through multiple network requests. This is called *overfetching*. A query language like GraphQL on the server-side and client-side lets the client decide which data it needs by making a single request to the server.

### Specification, NOT Implementation

## Specification...NOT Implementation #

GraphQL is a query language, it is a way to get data from an API to your application hence, it is a **specification** rather than an **implementation**. Initially, Facebook open-sourced the GraphQL specification and its reference implementation in JavaScript. Now, along with Javascript, several libraries have been incorporated in implementation. The ecosystem around GraphQL is growing horizontally by offering multiple programming languages, but also vertically, with libraries on top of GraphQL like Apollo and Relay.

## Queries & Mutations #

Currently, GraphQL operations can be divided into two broad categories, a query (read) and mutation (write). Each of these operations is only a string that needs to be constructed according to the GraphQL query language specification.

Queries are used for data fetching and mutations are used to modify server-side data. In the example below, you will see that a query has the exact same shape as the result. This essential GraphQL feature always provides you with the expected results because it lets the server know exactly what the client is asking for.

```
//GraphQL Query:

query{
  course(id: "5"){
    id
    name
    author
  }
}

//Result of above Query:

"data":{
  "course":{
    "id": "5",
    "name": "Learn GraphQL with React",
    "author": "Robin Wieruch"
  }
}
```

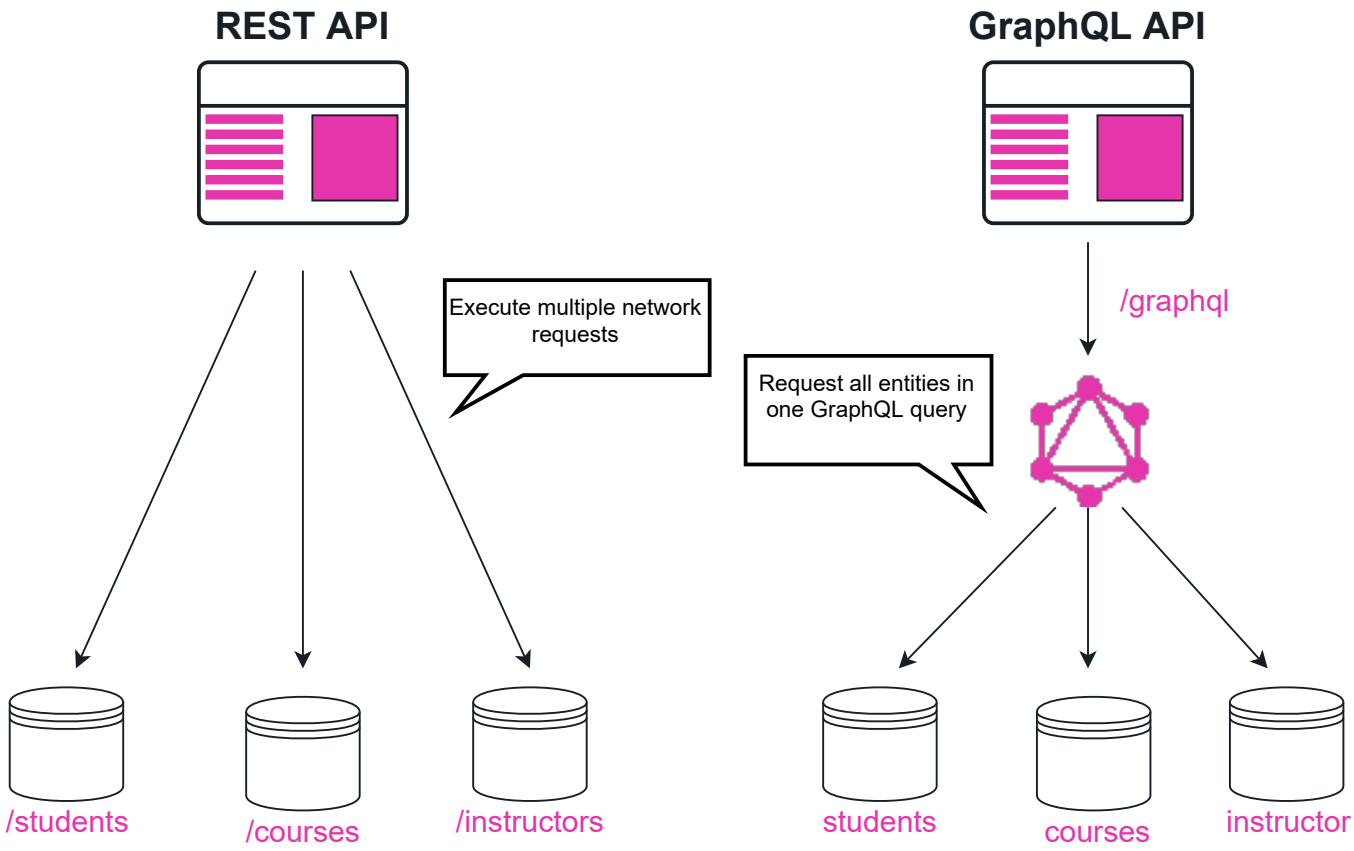
A Sample Query

Once this GraphQL operation reaches the backend application, it can be

interpreted against the entire GraphQL schema there and resolved with data for the frontend application. GraphQL is neither opinionated about the network layer, which is often HTTP, nor about the payload format, which is usually JSON. In short, it isn't opinionated about the application architecture in general.

## Relational Queries #

With GraphQL, we can make relational queries of multiple fields which results in us getting all the data required in one trip (query), unlike the REST architecture in which we would need to make multiple requests (one for each field).




In the example below, a query is requesting multiple resources (**author**, **article**) which are called **fields** in GraphQL. It also requests a particular set of nested fields (**name**, **urlSlug**) for the field **article**, even though the entity itself offers more data in its GraphQL schema (e.g. **description**, **releaseData** for article). A RESTful architecture would need at least two waterfall requests to retrieve the author entity and its articles, but the GraphQL query made it happen in just one query. In addition, the query only selected and sent the necessary fields instead of bringing back the whole entity.

Environment Variables

Environment Variables

| Key: | Value: |
| --- | --- |
| REACT_APP_GITHUB... | Not Specified... |
| GITHUB_PERSONAL... | Not Specified... |

```
// GraphQL Query:
author(id: "7") {
  id
  name
  avatarUrl
  articles(limit: 2) {
    name
    urlSlug
  }
}

// Result of Above Query:
{
  "data": {
    "author": {
      "id": "7",
      "name": "Robin Wieruch",
      "avatarUrl": "https://domain.com/authors/7",
      "articles": [
        {
          "name": "The Road to learn React",
          "urlSlug": "the-road-to-learn-react"
        },
        {
          "name": "React Testing Tutorial",
          "urlSlug": "react-testing-tutorial"
        }
      ]
    }
  }
}
```

A Sample Query

That’s GraphQL in a nutshell.

We will now move on to a brief description of the GraphQL server and client.