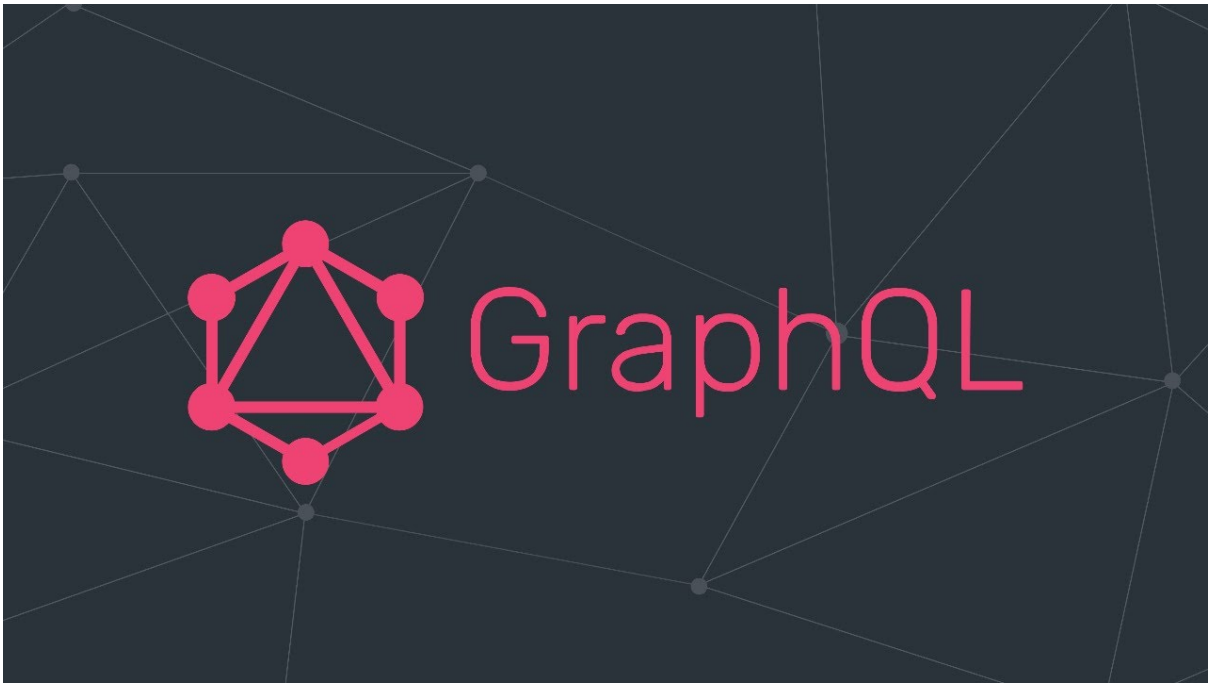
Universitat d'Alacant
Universidad de Alicante

GraphQL

APIs GraphQL en profundidad



Nombre: Sergio Orozco Palencia

Asignatura: Aplicaciones Distribuidas en Internet 2021/2022

Índice

Introducción a las APIs GraphQL

Introducción

El objetivo de este trabajo es explicar y difundir el funcionamiento de las APIs GraphQL, así como sus diferencias respecto a APIs REST tradicionales e incluso comparándolas con las bases de datos relacionales. GraphQL tiene su origen en Facebook, y ha sido liberado y transferido a la Fundación GraphQL, que forma parte de la Fundación Linux.

Primero daremos una definición de lo que es una API GraphQL y cuál fue su propósito original, y posteriormente daremos paso a la explicación más profunda de las mismas.

Además, también veremos algunas estadísticas de uso de las APIs GraphQL, para que veamos hasta qué punto son relevantes a día de hoy en el mundo del desarrollo web.

Al final del documento se incluyen una serie de referencias a algunos artículos y páginas de las que se ha extraído la información aquí presentada.

Definición de API GraphQL

> *"GraphQL es un lenguaje de consulta y manipulación de datos para APIs, y un entorno de ejecución para realizar consultas con datos existentes"*

Wikipedia

Es decir, esencialmente son como las APIs REST en el sentido de que su propósito es ofrecernos un estándar para consultar y manipular datos. Pero además, GraphQL también nos ofrece la posibilidad de desarrollar herramientas como GraphiQL fácilmente. GraphiQL tiene interfaz gráfica y nos permite realizar consultas para probar nuestra API de manera sencilla.

La propia fundación GraphQL la define como una "Query Language" para tus APIs, que además no está atada a ninguna base de datos ni motor de almacenamiento específico.

Explicación en profundidad de las APIs GraphQL

Vamos a seguir el principio del tutorial oficial de GraphQL para entender mejor cómo funciona.

Servicio GraphQL

Un servicio GraphQL se crea en primer lugar definiendo los tipos y los campos que van a formar parte de nuestra API, así como una serie de funciones para cada campo de cada tipo:

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}

function Query_me(request) {
  return request.auth.user;
}

function User_name(user) {
  return user.getName();
}
```

En el código de arriba se puede ver que hemos creado dos tipos "Query" y "User" con sus respectivos campos, y hemos definido dos funciones, una para el campo "me" de "Query" y otra para el campo "name" de "User".

Una vez el servicio GraphQL está en marcha, puede recibir "queries GraphQL" que validará y ejecutará. Primeramente comprobará que la "query" se refiere a los tipos definidos y no a otros que no existen, y ejecutará las funciones que correspondan para obtener el resultado final.

Por ejemplo al realizar la query siguiente:

```
{
  me {
    name
  }
}
```

Podría devolver el siguiente resultado en JSON:

```
{
   "me" :
  {
    "name" : "Pepito"
  }
}
```

Argumentos

Pero es que además, podemos pasarle argumentos en la propia query para solicitar un dato concreto:

```
{
  human(id: "1000")
  {
    name
    height
  }
}
```

En este caso se nos devolvería el humano con id 1000. De hecho, podemos incluir argumentos en todos los campos que queramos. Podemos pedir que nos devuelva la altura en pies en vez de centímetros (por ejemplo), o cualquier otra cuestión que hayamos definido en el "schema", sin necesidad de hacer múltiples llamadas diferentes como en las APIs REST.

Alias

Podemos hacer uso de alias para pasarle varios argumentos diferentes al mismo campo, como se ve en la siguiente query:

```
{
  empireHero: hero(episode: EMPIRE) {
    name
  }
  jediHero: hero(episode: JEDI) {
    name
  }
}
```

Y como vemos se devuelve un resultado con cada uno de los dos alias que hemos creado:

```
{
  "data": {
    "empireHero": {
      "name": "Luke Skywalker"
    },
    "jediHero": {
      "name": "R2-D2"
    }
  }
}
```

Nombre de las queries

Hasta ahora hemos utilizado la forma acortada de hacer una query, pero puede sernos útil el darles un nombre. Para ello no hay más que escribir antes de la primera llave:

```
query NombreQuery { ...
```

Variables

También podemos hacer uso de variables en nuestras queries, para ello no hay más que declararlas al principio de la misma y usarlas en su interior, y como se puede ver se les puede asignar un valor por defecto (Episode = JEDI):

```
query HeroName($episode: Episode = JEDI) {
  hero(episode: $episode) {
    name
  }
}
```

Mutaciones

Técnicamente es posible que una query cambie información en el servidor, pero no es aconsejable. Lo habitual y lo que se recomienda es emplear una mutación. Por ejemplo, para crear una nueva reseña se podría usar la mutación siguiente:

```
mutation CreateReviewForEpisode($ep: Episode!, $review:
ReviewInput!) {
  createReview(episode: $ep, review: $review) {
    stars
    commentary
  }
}
```

Queries en paralelo, mutaciones secuenciales

En GraphQL las queries se procesan en paralelo, pero las mutaciones van secuencialmente una detrás de otra. Esto es importante porque es la diferencia más relevante entre mutaciones y queries. Por ejemplo, si tenemos una mutación que incrementa un número, esto nos garantiza que se incrementará ese número tantas veces como mutaciones haya, sin que se solapen unas con otras.

En definitiva

Hay muchas más cuestiones que podrían ser de interés según la circunstancia, pero la base de una buena API GraphQL son todos estos conceptos que hemos explicado hasta ahora. Ahora vamos a ver cuáles son las diferencias más palpables entre una API GraphQL y una API REST.

Comparativa API REST vs API GraphQL

Único recurso/s vs múltiples en la misma query

En una API REST convencional trabajamos con un único recurso o colección de recursos por cada petición (por ejemplo un array de posts, un post concreto o un usuario). De hecho, para cada uno de ellos tendremos un "endpoint" diferente (/posts, /posts/{id}, /users...).

Sin embargo, en una API GraphQL podemos solicitar múltiples recursos en una misma query, así como realizar modificaciones y crear nuevos mediante las mutaciones.

Over Fetching / Under Fetching

Otra cuestión con la que nos puede ayudar GraphQL es el traer exactamente los datos que se necesitan, ni más ni menos. Algo que ocurre con frecuencia al trabajar con APIs REST es que obtengas datos que no necesitas, o por el contrario te falte algún dato y tengas que hacer una petición adicional.

Por ejemplo: al solicitar la información de un usuario pueden llegarte datos que no necesitas como la fecha de nacimiento, el peso, la altura... Los términos que se emplean para describir una situación así son: Over Fetching, si traemos más datos de los que vamos a necesitar, y Under Fetching, si tenemos que hacer otra petición adicional porque nos falta algún dato.

Versionado de la aplicación

GraphQL también es útil para facilitar el cambio de versiones y el añadido de nuevas funcionalidades, ya que como el cliente puede indicar en las queries exactamente los datos que quiere, no hará falta modificar a su vez el servidor, pues es algo integrado en el propio lenguaje y el servidor simplemente leerá e interpretará la query recibida.

Cuatro métodos (CRUD) vs POST

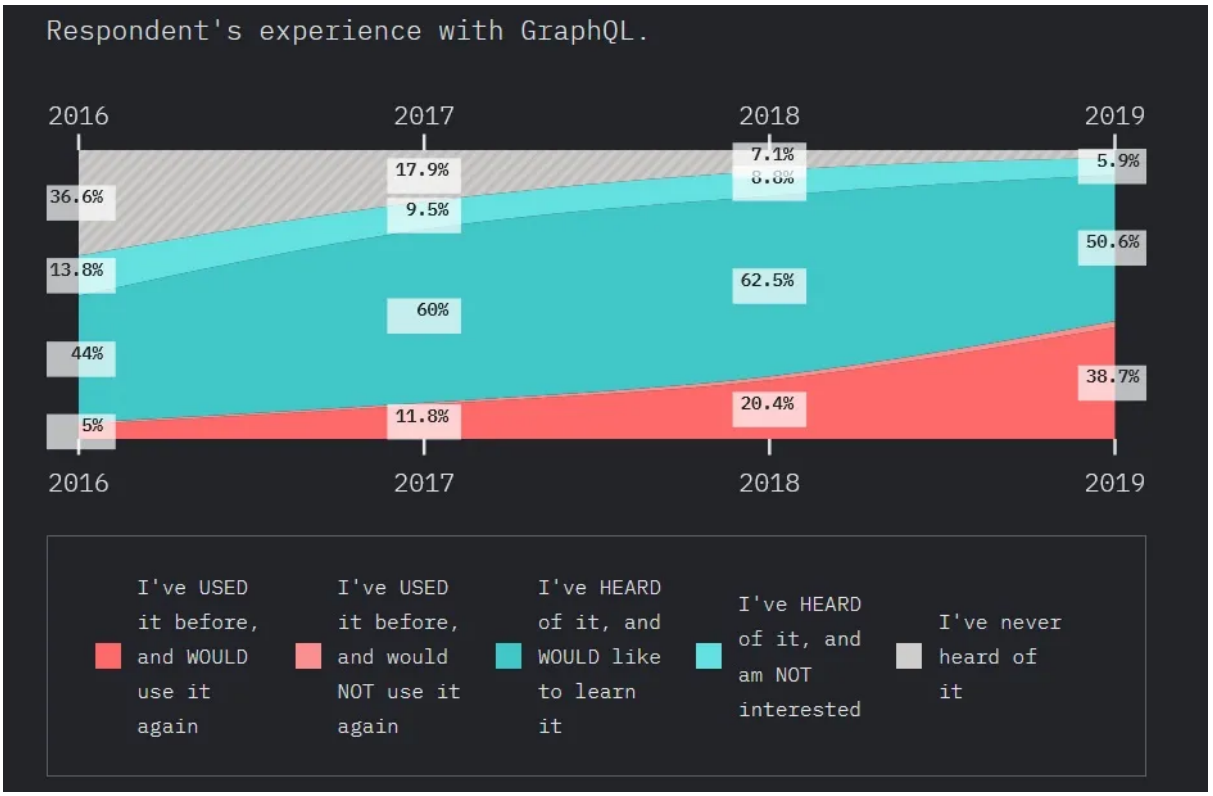
Mientras que en una API REST se utilizan cuatro métodos para cada una de las operaciones (GET, POST, UPDATE, DELETE), en GraphQL se emplea únicamente POST.

Estadísticas de uso de GraphQL y REST

Por último, vamos a visualizar el estado actual de GraphQL y REST mediante una serie de estadísticas más o menos recientes.

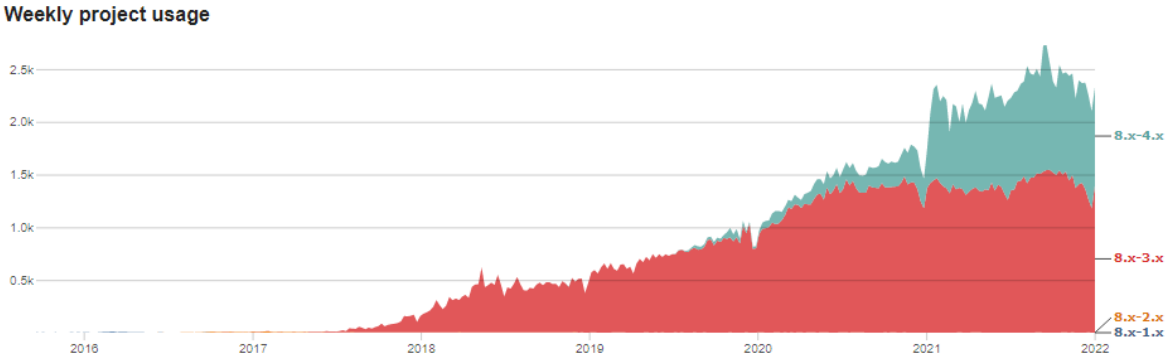
State of API 2020 Report

En esta gráfica se puede ver que hasta un 19% de desarrolladores de APIs emplean GraphQL, mientras que la "modalidad" más popular de REST llega hasta el 82%. Lógicamente las APIs GraphQL todavía están en pleno crecimiento, pero poco a poco se va notando su auge y cada vez son más los que la utilizan por su flexibilidad y sencillez.



State of Javascript 2019 Report

Como se puede ver, cada vez más gente ha usado o escuchado acerca de las APIs GraphQL, y la mayoría de los que las usan dicen que volverían a usarlas de nuevo en sus desarrollos.



Y por último en esta gráfica podemos ver el número de proyectos en los que se usa cada versión de GraphQL. Se puede observar que GraphQL no empezó a despegar verdaderamente hasta finales de 2017 y principios de 2018, y desde entonces no ha parado de crecer.

Bibliografía

GraphQL | A query language for your API

GraphQL - Wikipedia, la enciclopedia libre

▹ Diferencias entre una API REST y GraphQL (enmilocalfunciona.io)

GraphQL vs REST - A comparison (howtographql.com)

¿Por qué deberíamos abandonar REST y empezar a usar GraphQL en nuestras APIs? (genbeta.com)

So, What the Heck is GraphQL?. Your ultimate guide on why and how to... | by Karthik Kalyanaraman | Bits and Pieces (bitsrc.io)

GraphQL vs. REST - A Comprehensive Comparison (rakuten.net)

GraphQL vs REST in 2021: A Detailed Comparison - Devathon

Usage statistics for GraphQL | Drupal.org