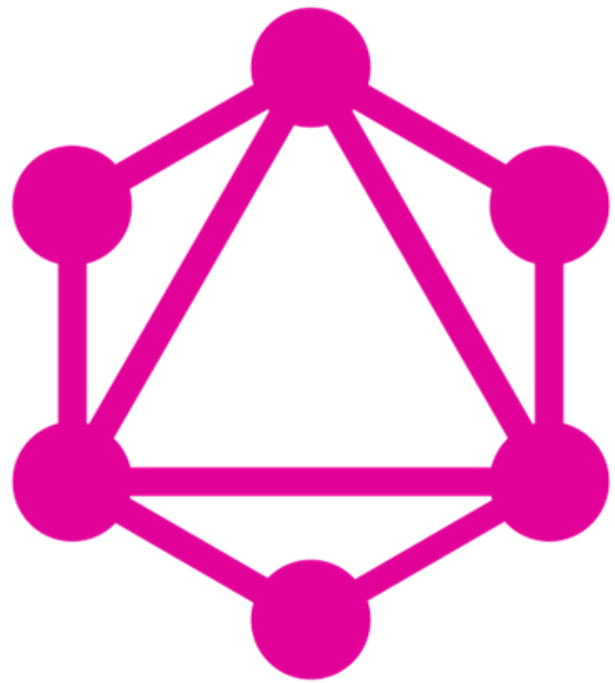

GraphQL

1. 두번 써봤어요(iOS+Node, React+Node)

2. 공부안하고 사용해서 더 공부해보고 싶었어요

탄생배경

- 2012년 facebook에서 개발하고 2015년에 오픈소스화
- **다양한 플랫폼**의 등장에 따른 데이터 제공 문제
- GitHub, Netflix, Medium, Twitter, Naver 등 다양한 곳에서 사용중

GraphQL?

- **A query language for your API**
- API 통신을 위해 사용되는 언어, 스펙
- DB랑 관련이 없어요
- 다양한 구현체가 있음
- 하나의 엔드포인트만 제공 `/graphql`

spec.graphql.org

The fields on the `query` root operation type indicate what fields are available at the top level of a GraphQL query. For example, a basic GraphQL query like:

Example № 34

```
query {
  myName
}
```

Is valid when the `query` root operation type has a field named "myName".

Example № 35

```
type Query {
  myName: String
}
```

Similarly, the following mutation is valid if a `mutation` root operation type has a field named "setName". Note that the `query` and `mutation` root types must be different types.

Example № 36

```
mutation {
  setName(name: "Zuck") {
    newName
  }
}
```

When using the type system definition language, a document must include at most one `schema` definition.

그래서 뭐가 좋은지??

- 원하는 데이터만 쿼리 언어로 요청하면 그것만 응답받는다
    - Data over fetching 문제 해결!
- 연관관계를 가지고 있는 데이터를 한번의 요청으로 받을 수 있다
    - 1+N 문제, Data under fetching 문제 해결!
- 라이브러리들이 좋은 문서화 툴을 자동으로 제공해줌. API문서를 작성할 필요가 없다

이제 한번 자세히 알아봐요

RESTful API vs GraphQL

GitHub API v3 (REST API)

- Request

```
https://api.github.com/users/YukJiSoo/repos
```

- Response

```
[
  {
    "id": 1296269,
    "node_id": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5",
    "name": "Hello-World",
    "full_name": "octocat/Hello-World",
    ...(more more more)
  }
]
```

원하지 않는 데이터까지 막 넘어와요

Data Over Fetching!!

GitHub API v4는...?

GitHub API v4 (GraphQL)

도구를 사용해봐요!

해결 😎

이제 다른 상황을 생각해봅시다

김개발씨는 포켓몬 고 게임을 운영하는 회사로 이직했어요🚀

입사 후 첫 프로젝트는 **포켓몬 도감** 만들기!

구현하게 된 기능은 검색한 포켓몬의 진화 전과 진화 후 정보를 함께 보여주기입니다.

(김개발씨는 프론트엔드 개발자🌈)

RESTful API를 사용한다면?

1. 특정 포켓몬 정보 요청

2. 응답 받은 데이터에서 진화 전과 진화 후 포켓몬 이름 확인

3. 또 요청

여러번 요청이 갔어요

1+N 문제, Data under fetching

GraphQL을 사용한다면?

해결 😎

좋쳐? 그럼 이제 어떻게 쓰는지 기본개념을 알아봐요

Schema

Query, Mutation

Resolver

Schema

서버와 클라이언트가 API통신을 위해 사전에 정의해 놓은 **데이터 타입의 집합**

도메인에 맞게 개발자들 끼리 서로 이야기하면서 정의

다양한 타입들이 있어요 (Object, Int, String, ID 등)

Pokemon Object 타입

```
type Pokemon {
  id: ID!
  name: String
  weight: Weight
  height: Height
  desc: String
  image: String
  evolvesFrom: Pokemon
  evolvesTo: Pokemon
}
```

👇👇👇 타입에 대해 더 자세히 알아보고 싶다면 아래 링크가 좋은 것 같더라구여 👇👇👇

https://code-masterjung.tistory.com/22

Query, Mutation

`Root Operation Types`

쿼리의 진입점이 되는 타입들

즉, Query와 Mutation에 선언된 필드들이 `entry point` 라고 생각하면 됨

Query

RESTful API에서의 `GET` 요청을 생각하면 됨

리소스를 읽어오는 작업을 수행

Mutation

RESTful API에서의 `POST`, `PUT`, `PATCH`, `DELETE` 요청을 생각하면 됨

리소스를 생성, 변경, 삭제하는 작업

Resolver

Schema를 정의하고 Query, Mutation에 응답해줄 필드를 추가했어요

그러면 이제 `데이터를 가져오는 로직` 을 구현해야 해요

그 코드를 **Resolver** 에서 작성해줍니다

기본적인 형태

```
const pokemonsQueryResolver = async (parent, args, context, info) => {
  const pokemons = await Pokemon.findAll();
  return pokemons;
};
```

Schema에 정의된 타입의 필드마다 매칭되는 resolver가 하나씩 존재해요

그래서 연쇄적으로 resolver가 실행되어서 데이터를 모은 후에 클라이언트에게 보내줘요

그러면 단점은?

- 기존의 Caching 방식을 사용할 수 없다
    - 이를 해결하기 위한 Client 라이브러리들이 존재한다
- 파일 업로드를 직접 구햔해야 한다
    - 파일 업로드는 REST API로 제공하면 된다